tech.net
working with windows
III./02.szám
1294 Ft

.NET kalauz:

tanuljunk .NET-et! De hogyan?

22. oldal

Nomádélet a XXI. században:

mobil IP

42. oldal

Rendezvénynaptár

43. oldal

ISSN 1586-5185
02

Ugróiskola: *felkészülés valami komolyabbra*

Szorgos munkával telt a január, s ez meglátszott a januári lapszám februári postázásakor is. Ahogy régi munkahelyemen mondták: ha jön a tanfolyam, még meghalni sem szabad az oktatónak. Nemhogy mással foglalkozni! Márpedig tanfolyam lett, ismét sikerült megmozgatnunk ötven-hatvan embert. Nagy szám ez? Hát bizony a Magyarországon fellelhető informatikusok-rendszergazdák-programozók számához képest elenyésző.
Mint már annyiszor az elmúlt két évben, most is oknyomozásba kezdtem, vajon mi a jelenség oka. Megkérdeztem például huszonvalahány közeli ismerősömet, milyen szaklapokat olvasnak, miből tanulnak. Több, mint 50%-ban az válaszolták, hogy „nem érünk mi rá ilyesmire"! Nem érnek rá olvasni! Oltaniuk kell a mindenütt felcsapó lángokat!

Hadd meséljem el egy régi élményemet: pár évvel ezelőtt főállású oktatóként dolgoztam egy hazai CTEC oktatóközpontban. Napi 8, *(néha, esti oktatással együtt 10)* heti 40, nem ritkán havi 180 óra telt oktatással. Hajtás közben nem éreztem sem fárasztónak, sem unalmasnak a munkát - de semmi másra időm nem jutott. Se olvasás, se gondolkodás. Mígnem egyszer, 1998 magasságában kisírtam egy amerikai konferenciautat magamnak. Nem emlékszem már, mi hajtott, azon túl, hogy még függőleges állapotában megmászhassam a World Trade Centert, de tény, hogy annyira az ügyön voltam, hogy idegességemben eltévesztettem a repülőgép indulásának napját, ennek folyományaképpen totálkárosra törtem az autómat, majd a roncsot hátrahagyva egy nap késéssel, és egy szál fogkefével elrepültem a Windows 2000 Technology Week rendezvényre Seattle-be.
Ez is kalandosnak tűnik, de mi volt ez ahhoz képest, hogy egy teljes hétig kiestem az örökös munkából! Napközben tanulás, délutántól CSÖND. Se munka, se barát, se email, se telefon, se SMS - csak a csönd. Szinte megrohantak a gondolatok! Nem így kéne tanítani, sok mindent nem úgy kéne szervezni, így és így lehetne mindent egy picit jobban csinálni...
Szinte kívülről *(és távolról)* láttam saját munkámat, és benne a nagy-nagy céltalanságot. Hazatérésem után akut konferenciálátogatóvá váltam, és például 2000-ben *(arányév!)* írd és mondd ötször szóltam meg konferenciaalibivel a világ előtt. Az az egy-egy hétnyi gondolkozási idő megváltoztatott engem. Lettek céljaim, és lett NetAcademia, és lett tech.net, és lett NATE. Ennyit tesz, ha valaki időnként leteszi a poroltót.

Nemrégiben végigültem Soczó Zsolt kollégámnál az ASP.NET tanfolyamot. Mint azt az msdev listán megírtam, a második napon elveszítettem a fonalat, de ismét nyertem három napot gondolkodásra. Meglepve tapasztaltam, hogy ha kisebb mértékben is, de a tanfolyam is kiragadott a napi robotból. Kevesebb gondolatom támadt ugyan, mert esténként csak visszaváltoztam irodakukaccá, majd apukává, de az a pár óra is élményszámba ment. Az ugróiskolával **ezt az élményt**, ennek egy falatkáját szeretnénk megmutatni mindenkinek. Adjon egy kis gondolkodási időt önmagának! Nem hosszú időt, nem marketingblablára, hanem az elvekre. Minden napi munka végezhető vak tapogatódzás mentén, de megfontolt cselekvéssorozattal messzebbre jutunk.
Sok olyan informatikus van, aki ezt a vezércikket sem olvassa, mert **nem ér rá**. Nincs öt perce. Nemhogy egy hete! A fokozatosság jegyében a NetAcademia felkínálja az Ugróiskola rendszert, ahol az előre kiadott tematika mentén ki-ki arra az alkalomra jön el, amelyik anyagát saját munkájában hasznosnak látja. A maximum másfél órás elfoglaltságot mind árban, mind időbeosztásban úgy pozicionáltuk, hogy a főnöktől sem engedélyt, sem anyagi támogatást kérni nem kell. Részletek a 44. oldalon.

Fóti Marcell
főszerkesztő

tech.net
working with windows

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

Kiadja és terjeszti a
NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

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

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

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

ISSN 1586-5185

2002 Február

.NET kalauz:

tanuljunk .NET-et! De hogyan?

Aki nekilát .NET-et tanulni gyakran úgy érzi magát, mint egy dzsungelharcos: vágja maga előtt a bozótot, küzd nagyon, de valójában fogalma sincs merre tart. Ez a kis útmutató azoknak nyújt segítséget, akik szeretnék élvezni a .NET fejlesztések előnyeit, és minél gyorsabban használható tudásra szeretnének szert tenni.

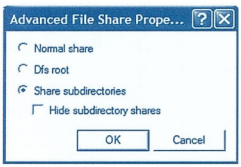
22. oldal

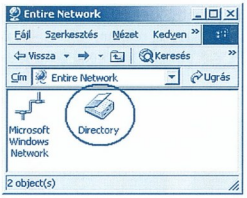
Farkasokkal táncoló

(IV. rész)

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

5. oldal





A trust leleplezése

Egy Windows 2000 Active Directory bevezetés kapcsán merült fel a Kedves Megrendelőben, hogy ő bizony egyszerre két azonos nevű munkaállomást szeretne csatlakoztatni a tartományba. Tudjuk, hogy ez nem megy. Ha annak is utánajárunk, hogy pontosan miért nem közelebb kerülünk a meghatalmazotti kapcsolatok lényegéhez, s kiderülhet, a Professional változatok nem is túl sokban különböznek a Szerverektől. Izgalmas utunkon Secure Channeleken bújuk át, SAM-ban kotorászunk, no és persze hiszünk (trust).

8. oldal

RIS

(IV. rész)

A RIS kétféle telepítőkészlettel képes dolgozni. Az egyik a Riprep segítségével készített, a másik a CD-ROM, vagy Script alapú telepítőkészlet. A Microsoft RIS-sel csupán a Windows XP/2000 Professional telepítését támogatja, sem korábbi operációs rendszerek, sem a Windows 2000 Server telepítését nem, azonban egy kis trükkel Windows 2000 Servert meg tudjuk etetni a risetuppal, így CD-ROM alapú telepítőkészletet tudunk készíteni a kiszolgálókhoz is.

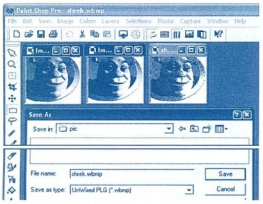
12. oldal

Internettörvény

vagy az elektronikus kereskedelem szabályozása?

A 2001. év karácsonyán az információs társadalom tagjai is megkapták a maguk ajándékát a törvényhozástól – az Internettörvényt *(2001. évi CVIII. törvény)*. Kérdés, hogy valóban Internettörvény született-e, azaz a jogalkotó ténylegesen kiterjesztette-e a jogi szabályozás hatósugarát a háló valamennyi jelenségére, vagy csupán az elektronikus kereskedelem, a gazdasági kapcsolatok körén belül maradt.

16. oldal



ASP Suli

Wap alapok

Előző számunkban bemutattuk a WML nyelv alapjait. Megtudtuk, hogy a wapos szolgáltatások kiszolgáló oldalról gyakorlatilag alig különböznek a hagyományos webes szolgáltatásoktól – ilyen alapon néhány apró beállítás után akár az Internet Information Services is képes wapos kiszolgálóként üzemelni. Akkor pedig már annak sincs akadálya, hogy a WML oldalakat dinamikusan, az ASP segítségével állítsuk elő!

18. oldal

.NET Akadémia

(II. rész) - .NET alaptípusok

Az előző részben áttekintettük a .NET alapjait madártávlatból, valamint láttuk hogyan kell mezőket, metódusokat és jellemzőket definiálni egy osztályban. Ebben a részben megnézzük az osztályok további jellemzőit, körbejárjuk, hogy miért nagyon fontos az érték és referencia típusok megkülönböztetése. Menet közben érintjük a .NET egyik igen fontos sajátosságát, a nemdeterminisztikus objektum-életciklust is.

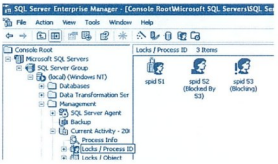
26. oldal

XMLgessünk

Az xml nagyszerű eszköz adatok átvitelére, heterogén rendszerek közötti információcserére, de csapnivaló adattárolásra. Redundáns és csak lassan lehet belőle kihámozni a tényleges információtartalmat. Hatékony adattárolásra már régóta léteznek megfelelő eszközök, az adatbázisok. A legtöbb adatbázis azonban relációs elvekre épült, téglalap alakú információhalmazokban gondolkodik, amelyek között kapcsolatok vannak definiálva. Hogyan valósítható meg az átjárás két világ között, a lehető leghatékonyabb módon? Erre keresem a választ a .NET eszközeivel.

30. oldal

Zárolás, livelock, deadlock



Az SQL Server zárairól lapunk 2001 márciusi számában már olvashattak Soczó Zsolt tollából *(28. oldal, Transact SQL VI. rész)*. Végigvette a zárak típusait, a tranzakcióizolációs szinteket, majd a cikk végén könnyelműen megígérte, hogy legközelebb a deadlock következik. Még egy év sem telt el, s íme a folytatás.

34. oldal

MsDownload

- itt a végleges .NET Framework!

2001 október közepétől a Netacademia Kft üzemelteti Magyarország egyik legnagyobb letöltőközpontját. A Microsoft Magyarország megbízásából elkészült oldalak az eddig megszokott tartalommal, de eltérő formában jelentkeznek. Az innen letölthető programok közül csemegézünk ebben a rovatban.

38. oldal

Dupla KV

RAID, Windows XP

39. oldal

Rendezvénynaptár

2002 tavaszán is folytatódik a TechNet Szemináriumok sorozata. A Microsoft Magyarország rendszermérnökei és vendégeik az előadások során áttekintik a kereskedelmi webhelyek biztonságát növelő technológiákat; kitérnek arra, hogy a Windows platform milyen eszközökkel segíti a vállalati webes alkalmazások fejlesztését; betekintést adnak az SQL adatbázisok adminisztrációjába, valamint ismertetik a hamarosan megjelenő Windows .NET Server és Project 2002 újdonságait. Mindezt természetesen a jól ismert „100% technológia, 0% marketing" szabály betartásával teszik.

43. oldal

Nomádélet a XXI. században: **mobil IP**

Az emberi szabadságvágy egyik ősidők óta jelenlevő megnyilvánulási formája a helyváltoztatás, illetve a cselekvés minőségét *(kényelem, sebesség stb.)* is figyelembevevő mobilitás. Amit a honfoglaló őseinknek a lovak és szekerek jelentettek, azt a XX. század embere számára az autó, a vonat, a repülő testesítette meg. Az utazás ma már az egyszerű halandók mindennapi életének is részévé vált.

42. oldal

Farkasokkal táncoló *(IV. rész)*

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

Az erőforrásokról általában

Minden erőforrás létrehozásának hasonló a menete: az első panelen megadjuk a nevét, a leírását, a csoportot, amelyhez tartozni fog, és az erőforrás típusát. Igazából ez a legbonyolultabb panel a művelet során.

A következő lépés, hogy megadjuk, milyen állomások láthatják vendégül az erőforrást *(preferred owner)*. A beállításokat a csoportjától örökli, jobb meghagyni a felajánlott állapotot. Ezután a függőségek következnek, erről a témáról az előző cikkben bőven volt szó. Az utolsó panel erőforrásról erőforrásra változik, itt kell ugyanis az egyes típusok speciális, csak rájuk jellemző adatait megadni. Van azonban olyan erőforrás, ahol nincs is ilyen panel, mert egyszerűen nem kell megadni speciális beállítást.

A legördülő listából összesen 15 erőforrástípus közül lehet választani, ám ez csalóka szám. A mappa megosztásból van mindjárt három altípus, de csak egy szerepel a listában. Van azután ún. „általános szolgáltatás", meg „általános alkalmazás", amely nem más, mint olyan szervizek clusteresítése, amelyek fejlesztésekor még nem gondoltak erre a fontos technológiára. Ezekből annyi lehet, mint égen a csillag. Szóval a 15 senkit ne tévesszen meg. A következőkben minden erőforrástípusra szánunk néhány mondatot, a fontosabbakat pedig részletesebben is megvizsgáljuk.

A létrehozás tulajdonképpen nem más, mint egy létező könyvtár kacifántos megosztása.

A fizikai lemez *(physical disk)* erőforrás

Nagyon fontos, de nagyon egyszerűen létrehozható erőforrás. A varázsló utolsó lapján csak annyi dolgunk van, hogy megadjuk a lemezmeghajtó betűjelét, pontosabban kiválasztjuk a lehetséges betűjelet. A nem közös és a már korábban kiválasztott, ilyen erőforrásként megadott lemezek nincsenek a listában. Az is előfordulhat, hogy egy általunk odagondolt lemez nem jelenik meg. Ekkor meg kell győződni arról, hogy:

1. Mindkét állomás azonos betűjelet adott-e a lemeznek?
2. A lemezen van-e egyedi jel *(disk signature)*?
3. Az egyedi jel létéről mindkét állomás biztosan tud-e?
4. A diszk nem dinamikus-e véletlenül?
5. Minden gyártóspecifikus eszközmeghajtót telepítettünk?

Az első feltételt könnyű ellenőrizni a lemezkezelővel. A második feltétel automatikusan teljesül, ha legalább egyszer elindítottuk a lemezkezelőt az erőforrás létrehozása előtt, és hagytuk, hogy a felbukkanó varázsló rátegye a „disk signature"-t az új lemezekre. Ugyanez a varázsló ebben a körben dinamikus lemezekké konvertálná a diszkeket, de ezt ne engedjük! A cluster lemezkezelője *(a clusdisk eszközmeghajtó)* csak egyszerű *(basic)* lemezeket kezel.

Az utolsó feltételt saját tapasztalataim alapján magam találtam ki. A lemezek – ahogy erről korábban már volt szó – valójában nem igazi lemezek, hanem valamilyen lemeztömb logikai felosztásai. Ez a lemeztömb, különösen ha diszkalrendszerről van szó, számos gyártóspecifikus eszközmeghajtót is igényelhet, amelyek telepítése nem triviális. Alapszabály tehát, hogy a fizikai lemezerőforrások létrehozása előtt győződjünk meg arról, hogy a fürt alatt dohogó lemezrendszer jól működik-e, és semmilyen megmagyarázhatatlan „tulajdonsága" sincs. Olyan ez, mint a házépítés: csak jó alapokon épül jó otthon.

Az IP cím *(IP address)* erőforrás

A lemezekhez hasonlóan szintén nagyon egyszerű és nagyon fontos erőforrás. A létrehozásához csak egy pontos címre és egy alhálózati maszkra van szükség. A virtuális gép minden egyéb TCP/IP beállítását az őt futtató állomástól veszi – egyszerűbb így az élet. Gyakori hiba, hogy az alhálózati maszkot rosszul adja meg a rendszergazda – a hatása éppen az, mint egy normál szerver esetén.

A hálózati név *(network name)* erőforrás

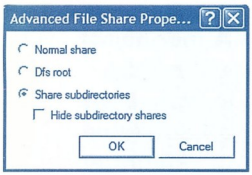
A hálózati név nem kevésbé fontos erőforrás, mint az előző kettő volt. Ez teszi lehetővé, hogy a virtuális szerverre név szerint hivatkozhassunk. A név – sajnos – NetBIOS név. A Windows 2000 ugyan regisztrálja a WINS mellett a DNS Servernél is, ám ez sovány vigasz. Az igazság a Q235529 cikkben van: a Windows 2000 cluster szolgáltatása és a NetBIOS egyelőre elválaszthatatlanok egymástól.

A közös mappa *(file sharing)* erőforrás

Az első három erőforrás ahhoz szükséges, hogy egy virtuális kiszolgáló egyáltalán létrejöhessen. A közös mappa viszont már „igazi" szolgáltatás. Három alfaja is van.

A létrehozás tulajdonképpen nem más, mint egy létező könyvtár kacifántos megosztása. Ez annyiból áll, hogy a könyvtárnak már léteznie kell, a megosztás pedig az erőforrás létrehozásával történik. Az eltávolítást épp fordítva kell elvégezni: előbb meg kell szüntetni az erőforrást *(ezzel megszűnik a megosztás)*, majd lehet törölni a mappát. Azért írom le ezt a triviálisnak tűnő lépéssort, mert kezdetben számtalanszor magam is eltévesztettem, és előbb töröltem a könyvtárat, minthogy megszüntettem volna az erőforrást. A következménye az volt, hogy a virtuális szerver, vagyis az erőforrás-csoport ész nélkül elkezdett egyik állomásról a másikra vándorolni, hátha ezzel orvosolhatja a „hibás" állapot-

ba került erőforrását. Nem sikerült neki... A tanulság, hogy kiszolgálófürt esetén még kell tanulni: a megosztásokat a Cluster Administrator segítségével kell létrehozni és megszüntetni.
A varázsló utolsó paneljén van egy „Advanced" jelzésű gomb, amellyel előcsalogatható a háromféle megosztás létrehozását lehetővé tévő ablak. Ez az:



Fáljmegosztás típusok a fürtön

A normális megosztás pontosan az, ami a neve. Ugyanilyen megosztást tud minden Windows 2000 termék, ez csupán a rendelkezésre állásában jobb. Mivel semmi különbség nincs, a szabályok sem különböznek: javasolt a megosztás jogosultsági beállításait alapon hagyni, és NTFS jogosultságokkal szabályozni a felhasználók hozzáférését az állományokhoz. Egy fontos tudnivaló azért van: a clusterszolgáltatás fiókjának legalább olvasási jogokkal kell rendelkeznie a közös mappán. Ha ez nincs így, akkor nem képes az erőforrást működőképes állapotba hozni. Ez persze nem a létrehozáskor jelent problémát, hanem amikor egy biztonsági felülvizsgálat során megvonják tőle a jogokat.
Az első változatnál érdekesebb a „share subdirectories" típusú megosztás, különösen akkor, ha az alkönyvtárakat automatikusan elrejtjük. Mire is jó ez? A Windows NT 4.0 minden verziója küzd azzal a problémával, hogy csak megosztásokhoz képes csatlakozni, megosztások alatti almappákhoz már nem. Ez pedig probléma a felhasználói könyvtáraknál, mert a szegény felhasználó elveszik a sok könyvtár között, és nem találja a sajátját. Szemfüles rendszergazdák úgy segítettek magukon, hogy megosztották a felhasználók könyvtárait is, és hogy ne sorakozzon több száz megosztott mappa a tallózó szolgáltatásban, ezért a könyvtárak megosztási nevei kaptak egy $ jelet, vagyis rejtve maradtak. No, ez a szolgáltatás pont ezt a funkciót végzi, csak most már automatikusan. Az igazat megvallva kellett is ez. Korábban ugyanis az ilyen megosztásokat is erőforrásként lettünk volna kénytelenek definiálni, az eredeti fürtkiszolgáló szoftver verzió azonban csak 900, a jelenlegi pedig csak 1600 erőforrást kezel, s bizony akadt olyan hely a világon, ahol ez korlátot jelentett. Arról nem is beszélve, hogy ilyen sok erőforrás terhelte is a clustert rendesen, míg most a megosztott alkönyvtárak már nem erőforrások többé, így minden a helyére került. Minden? No, ha majd csak Windows 2000, vagy XP munkaállomások lesznek a hálózaton, akkor lehet faragni a scriptet, amely megszünteti a rengeteg rejtett megosztást, és a normál almappához csatolja a felhasználókat.
Egy lehetséges hibára szeretném felhívni még a figyelmet: ha a Windows NT-hez mellékelt „User Manager for Domains" eszközzel hozzuk létre a felhasználó könyvtárát, akkor annak jogosultsági listájában egyedül a felhasználó lesz benn teljes eléréssel, és semmi-senki más. Ezért aztán a fürtkiszolgáló fióknak sem lesz elegendő joga a könyvtárat automatikusan megosztani. Ez éppen az n+1. ok, hogy ne NT-t használjunk. *(Épp most vonja ki a piacról a Microsoft az NT4-et – a szerk.)*
Nem esett még szó a DFS-megosztásról. A betűszó annyit tesz, hogy elosztott fájlrendszer, és arra való, hogy a rendszergazdák egyetlen közös névtérbe rendezhessék a megosztásaikat, egyfajta óriási virtuális megosztásrendszert létrehozva. A technikának az az előnye, hogy a jól megszerkesztett névtér állandó maradhat, míg alatta a megosztások és kiszolgálók tetszőlegesen mozoghatnak, változhatnak. Ez a szolgáltatás már a Windows NT 4.0-ban is jelen volt, igaz, nem a CD-n, hanem webről letöltve lehetett telepíteni. A gyenge pont pedig az volt, hogy a DFS gyökere *(amelyből csak egy lehetett)* kártyavárként döntötte össze a névteret, ha nem volt elérhető - Achilles sarok volt tehát. Emiatt nem is vált igazán népszerűvé a DFS.
A Windows 2000 azonban változtatott a helyzeten kétféleképpen is. Egyrészt megjelent a tartományalapú DFS-rendszer, amely a névteret az AD-ban tárolja, ami így már redundánssá vált. Másrészt megjelent a fürtszolgáltatásban a DFS megosztás létrehozása. Miért volt szükség erre, amikor ott az AD? Két ok miatt is. Egyrészt nem mindenki akar Active Directory szolgáltatást, másrészt ehhez az AD-val integrált DFS-hez vagy Windows 2000-es ügyfelek kellenek, vagy AD klienst kell telepíteni minden egyes korábbi Windows verzióra.
A fürtszolgáltatás rendelkezésre állása pont az Achilles sarkot szünteti meg, de kompatibilis a korábbi NT verziókkal is. Az ilyen DFS gyökérmegosztást „Stand alone" DFS-nek hívják, megkülönböztetve az AD-val integrálttól. Minden szépsége ellenére ennek a szolgáltatásnak is vannak korlátai, nevezetesen:

- Csak egyetlen DFS megosztást lehet létrehozni a teljes fürtön.
- Nem lehet egyszerre AD-val integrált DFS-t és Stand alone DFS-t létrehozni a kiszolgálófürtön.

Ez azt jelenti, hogy bizony a DFS megosztásnak helyet adó virtuális kiszolgáló neve jó időre beleég a rendszerünkbe.
Van még két súlyos DFS korlát a fürtszolgáltatásnál.

- Az AD-val integrált DFS létrehozásakor a gyökérkönyvtár nem lehet a fürt közös lemezein.
- A fürt közös lemezein található megosztások, amelyek részei a DFS névtérnek, nem használhatják a Windows 2000 fájlreplikációs szolgáltatását.

Az első problémát még csak-csak át lehetne hidalni egy nem túl elegáns megoldással, de a második korlát bizony mellbevágó. Ez ugyanis azt jelenti, hogy a közös névtérbe szervezett, a vállalat eltérő telephelyein eltérő módon használt könyvtárrendszert nem lehet automatikusan replikálni. *(Például ugyanazon elérési utat használva az egyik telephelyen létrehoznák a termelési jelentést, az automatikusan átreplikálódna a központba, ahol a vezetőség a WAN vonalakat nem terhelve nézegethetné azokat.)* Jelenleg csak az Exchange 2000 nyújthat megoldást a problémára saját mappáinak replikációjával. No és a .Net Server, ha megjelenik...

A DFS betűszó annyit tesz, hogy elosztott fájlrendszer, és arra való, hogy a rendszergazdák egyetlen közös névtérbe rendezhessék a megosztásaikat, egyfajta óriási virtuális megosztásrendszert létrehozva.

A fürtkiszolgáló fizikai lemezeinek és közös mappáinak érdekes problémája a hibajavítás, de erről nem itt, hanem egy másik cikk keretében szeretnék írni. A hasznos olvasnivalókat a cikk végén soroltam fel.

A nyomtatási sorkezelő *(print spooler)* erőforrás

A közös mappáknál nem kevésbé érdekes a közös nyomtatók létrehozásának feladata. Az alapvető tudnivalók:

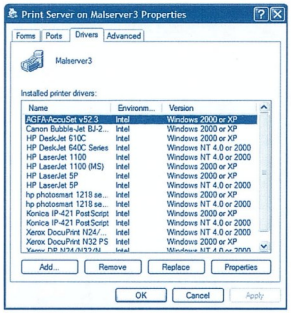
A nyomtatási sorkezelő csupán alapja a fürtszolgáltatás hálózati nyomtató megosztásának. Létrehozásával azt tesszük lehetővé, hogy **lehessen** megosztott, nagy rendelkezésre állású nyomtatónk.

Egy virtuális gépen csak egy spooler erőforrást hozhatunk létre, de az akármennyi hálózati nyomtatót is kiszolgálhat. A nyomtató csak akkor lehet közös, ha hálózatról *(értsd: TCP/IP alapon)* el lehet érni, és nem csatlakozhat párhuzamos porton az egyik vagy a másik állomáshoz.

A hálózati nyomtatónak szüksége van a TCP/IP nyomtatási szolgáltatásra *(esetleg a UNIX vagy más nyomtatási szolgáltatásra környezettől függően)*, ezeket tehát telepíteni kell az erőforrás létrehozása előtt mindkét kiszolgálóra. Végezetül hozzunk létre a kiszemelt virtuális gépünk egyik fizikai lemezén egy „Spool" könyvtárat. A továbbiakban feltételezzük, hogy van már egy virtuális szerverünk névvel, IP-címmel és egy fizikai lemezzel.

Jöhet az erőforrás: az első lapon válasszuk ki a „Print Spooler" erőforrástípust, és adjunk nevet neki. A következő lapon hagyjuk változatlanul a lehetséges állomásokat. A függőségek paneljén tegyük függővé a sorkezelőnket a hálózati névtől és a fizikai lemeztől, végezetül az utolsó lapon adjuk meg a korábban létrehozott Spool könyvtárat, ahol a nyomtatószerver az átmeneti állományait tárolhatja. Ezzel kész a sorkezelő, de még csak az út felénél járunk. Nincs ugyanis még sem portunk, ahová az adatot küldenénk, sem meghajtónk, sem közös nyomtatónk. De most lesz!

Tallózzunk a hálózaton az egyik fürtállomásunkhoz. Lépjünk be a nyomtatók mappába. Válasszuk ki a „Fájl" menü „Server Properties" menüpontját.



Nyomtatómeghajtók az egyik állomáson

A Driver fülön adjuk hozzá a leendő nyomtatónk meghajtóit. Amint az a képen is látható, többféle nyomtató eszközvezérlő létezik, ha tehát többféle kliensünk van, mindegyik rendszer meghajtóját hozzá kell adnunk. Ez némely Windows NT 4-es meghajtónál gondot okozhat, ezért egy másik gépen érdemes kísérletezgetni a nyomtatók meghajtóival. Ha valaki végképp nem boldogul, ajánlom figyelmébe a Fixprnsv.exe segédprogramot és a Q247196 cikket. Mivel a téma nem igazán clusterspecifikus, ezért az ennél a fázisnál felmerülő problémákat most nem tudjuk felsorolni és megoldani.

Ha megvannak a meghajtók az első node-on, akkor ugyanezt a műveletet el kell végezni a másik állomáson is. Azért kell ezt megtenni, mert a nyomtatómeghajtók nem a közös lemezterületen találhatók, hanem az állomások print$ megosztásaiban. Figyelem! A meghajtók, és később a frissítéseik is legyenek azonosak a két állomáson, különben megjósolhatatlan furcsaságokba ütközhetünk a későbbiekben.

Az eszközkezelők után jöhetnek a közös nyomtatók. Tallózzunk ezúttal a virtuális szerverünkhöz. Látható, hogy a megosztások között létrejött egy „Nyomtatók" speciális mappa, amelybe belépve pontosan ugyanazokat a feladatokat végezhetjük el, mint egy hétköznapi Windows 2000 nyomtatószerveren. Nosza, adjuk hozzá azt a nyomtatót! Amikor a varázsló a port kiválasztásához ér, adjunk meg nyugodtan új portot, válasszuk a „Standard TCP/IP"-t, vagy azt, amelyik a mi környezetünkben a megfelelő. Adjunk egyedi nevet a portnak. Visszajutunk az előző ablakhoz, válasszuk ki az újonnan létrehozott kaput, és lépjünk tovább. A Windows 2000 alatt kicsit könnyebb a helyzetünk, mert a port létrehozását csak egyszer kell elvégezni, a konfigurációs adatokat ugyanis a cluster hive alatt tárolja a fürt, tehát közös részen. A korábbi verzióknál ez még nem így volt, ott előbb helyi nyomtatókat kellett létrehozni mindkét állomáson, hogy létrejöjjenek a portok. No, ez már a múlté.

A meghajtók, és később a frissítéseik is legyenek azonosak a két állomáson, különben megjósolhatatlan furcsaságokba ütközhetünk a későbbiekben.

A meghajtók között válasszuk ki a már egyszer megadottat, így a következő panelen lehetőségünk lesz megtartani az előzetesen feltett drivereket. Ezután már csak nevet kell adnunk a nyomtatónak, és rendelkeznünk kell, hogy a rendszer ossza meg a felhasználók felé. Ha van AD címtárunk, akkor publikálhatjuk a nyomtatót a címtárba is. Ezzel el is készültünk. A biztonság kedvéért javaslom, hogy teszteljük a létrehozott nyomtatót, majd mozgassuk át a másik állomásra a virtuális szervert, és végezzük el újra a tesztet. Így lehetünk csak biztosak, hogy működik az átkapcsolás. Azt is ellenőrizzük le, hogy minden kliensünk képes nyomtatni, valamint, hogy a Windows 2000 és Windows XP ügyfelek látják az Active Directoryban a nyomtatónkat. Ezzel megteremtettük a megbízható, központosított nyomtatás alapjait.

Végezetül néhány Knowledge Base cikket sorolok fel, amelyek eligazítást nyújthatnak a cikk által érintett problémák esetén.

Q185212	Cluster Server Does Not Support More Than 900 Shares
Q186496	Securing a Common Folder
Q194831	SP4 Cluster Shares Must Be Reset to Recognize Added Subdirectories
Q208243	Folder Share-Level Permissions Override Subfolder Permissions
Q224967	How to Create File Shares on a Cluster
Q254219	Security Considerations When Implementing Clustered File Shares
Q257389	Microsoft Cluster Server Does Not Share Folders Automatically
Q229733	During Cluster Failover Print Queue Monitoring May Stop

Lepenye Tamás (MCSE 2000)
lepenyet@mal.hu

A trust leleplezése

Egy Windows 2000 Active Directory bevezetés kapcsán merült fel a Kedves Megrendelőben, hogy ő bizony egyszerre két azonos nevű munkaállomást szeretne csatlakoztatni a tartományba. Tudjuk, hogy ez nem megy. Ha annak is utánajárunk, hogy pontosan miért nem, közelebb kerülünk a meghatalmazotti kapcsolatok lényegéhez, s kiderülhet, a Professional változatok nem is túl sokban különböznek a Serverektől. Izgalmas utunkon Secure Channeleken bújuk át, SAM-ban kotorászunk, no és persze hiszünk (trust).

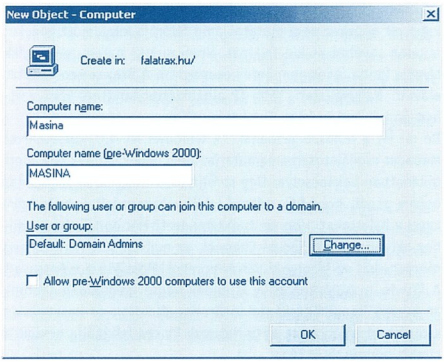
A tartományhoz csatlakozás menete

Először tekintsük át a sokak által naponta végrehajtott folyamatot. A csatlakozás kétféleképpen végezhető el:

- vagy mindent a munkaállomás oldaláról hajtunk végre *(és meg kell adnunk egy tartományi erős ember nevét és jelszavát)*
- vagy a munka oroszlánrészét rábízzuk egy megfelelő jogosultságú tartományi felhasználóra

A csatlakozáskor létrejön egy nagyon fontos objektum a **címtárban** *(nem a munkaállomáson!)*: a számítógépfiók, más néven Computer Account Object *(CAO. Ennek létrehozásához kell egyébként megadnunk egy erős ember nevét és jelszavát)*.

Ha mindent a munkaállomás oldaláról hajtunk végre, a CAO létrejön, és azonnal felhasználódik, a folyamat a háttérben, csendben lezajlik. Ha a másik utat járjuk, választ kaphatunk arra, hogy miért nem „lóghat" két azonos nevű munkaállomás egy tartományon. Hozzuk létre a CAO-t a tartományban:

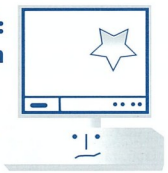


Computer Account Object létrehozása Windows 2000 tartományvezérlőn. Feltétlenül adjuk meg, hogy ki lesz képes felhasználni! (Change gomb)

Mint látható, itt a MASINA nevű masina számára készítjük elő a terepet. Nyilvánvaló, hogy ezen a ponton még nem dőlt el, hogy pontosan melyik gép lesz az az egyetlenegy, amelyik a későbbiekben használni fogja. Ha van két, egyformán MASINA nevet viselő munkaállomásunk, most még mindkettő rávetődhet – de csak az egyik, a „gyorsabbik" győzhet. Vajon miért?

A Windows NT/2000 tudathasadása

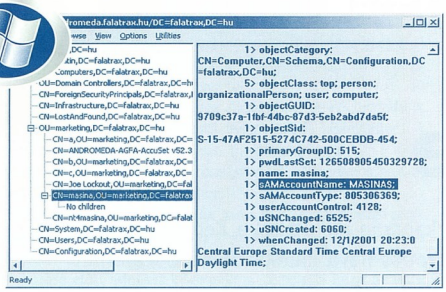
Egy Windows 2000 valójában két személyiség nevében ténykedik a hálózaton. Egyfelől itt van a bejelentkezett felhasználó, Józer Jolán, akinek van ugyan egy-két joga és feladata, de ő csak vendég a gépen. Sokkal jobban a géphez van nőve a saját tudatállapota. Akár be van személyesen jelentkezve valaki, akár nincs, az NT/2000 önmaga is tesz-vesz a hálózaton. Képzeljük el így:



A Windows 2000 személyiségei. Ha nincs bejelentkezve hús-vér ember, a gép akkor is VALAKI!

Az ábra enyhén csúsztat annyiban, hogy nem a teljes operációs rendszer fut kernel módban, sőt, jobbára csak az eszközmeghajtók, és az oprendszer lelke *(Hardware Abstraction Layer)*, de mivel most éppen a gép lelkéről beszélünk, ennyi tárgyi tévedés még „belefér". Miután megállapítottuk a két személyiséget, megvizsgálhatjuk, hogy hogyan jelennek meg a hálózaton. Józer Jolán attól az aki, hogy helyesen megadta nevét és jelszavát. Nem meglepő ezek után, hogy MASINA is azért az, aki, mert bootoláskor **megadja nevét és jelszavát a tartományvezérlőnek**. Ez ugyanis a hálózati azonosítás alapja, akár gép az „illető", akár ember.

A gép bejelentkezési neve nyilván a gép nevéből származik *(egészen pontosan egy dollárjel hozzábiggyesztésével)*. Ha belekukkantunk az Active Directoryba a telepítőlemezen található Support Tools eszközkészlet LDP.EXE eszközével *(pontos használatáról a tavaly májusi számban olvashattak)*, ezt láthatjuk:



A sAMAccountName a számítógépfiók bejelentkezési neve. Ha a jelszavára is kíváncsiak vagyunk...

A neve tehát megvan. De mi lehet a jelszava? Nyilván van jelszava, mert ha nem lenne, már régesség ettől a biztonsági réstől lenne hangos az Internet. És az emberekhez hasonlóan egyvalaki tudja csupán ez a jelszót: maga a gép! Csatlakozás közben a MASINA$ nevű illető csöndben megváltoztatja az *(akkor már saját)* jelszavát valami random jelsorozatra. És **ezzel** a lépéssel záródik be az a kiskapu, amin át újabb, azonos nevű gép juthatna a tartományba. Ha jön egy újabb MASINA nevű számítógép, akkor az – a helyes jelszó ismerete híján – üres jelszóval próbálna csatlakozni, ami már nem megy!
Esetleg búykál bennünk a kérdés, hogy vajon a munkaállomás hol is tárolja a saját jelszavát. Józer Jolánról tudjuk hol tárolja: vagy a buksijában, vagy kiragasztja a monitorra. És a MASINA? ő bizony a regisztrációs adatbázisban dugdossa előlünk, itt:

```
HKLM/SECURITY/Policy/Secrets/$MACHINE.ACC
```

Egy REG_NONE *(!)* típusú változóban. Ja, hogy ehhez nem férünk hozzá? De hát mondtam, hogy dugdossa! :)

A Secure Channel

Minden NT alapú, tartományi tag operációs rendszer már a rendszertöltés folyamán bejelentkezik a tartományba. A bejelentkezés után ez a kapcsolat egészen az operációs rendszer leállításáig megmarad. Mintha a gép egy kiskaput nyitna magának a tartomány felé, hogy a későbbiekben, amikor majd mindenféle Józer Jolánok hitelesítését, és egyéb feladatokat kell elvégezni, szabadon ki-be sétálhasson rajta. A kiskapu hivatalos neve biztonságos csatorna, azaz Secure Channel. Történelmi tény, hogy a Windows NT 4.0 a Service Pack 4 javítócsomag megjelenéséig nem volt képes titkosítani a Secure Channel forgalmát: minden adat, mely azon jött-ment *(Access Tokenek stb.)*, olvasható formában került ki a kábelre.

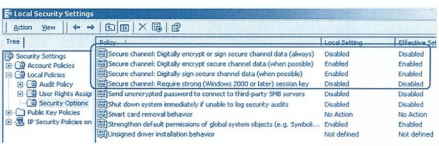
Minden NT alapú, tartományi tag operációs rendszer már a rendszertöltés folyamán bejelentkezik a tartományba.

Kitérő: mivel a Windows 9x ágnál nincs számítógépfiók, nincs Secure Channel sem. A mai napig titkosítatlanul áramlik - a jelszón kívül - minden hitelesítési adat ezekre a gépekre. Vállalati hálózaton nincs helyük!

SP4 előtt a Secure Channel biztonsága annyiban merült ki, hogy erősen bíztunk benne: a gép indulásakor létrejött TCP csatorna két végpontját senki sem képes elmozdítani, és magára irányítani. Mivel azonban a neten sorra jelennek meg a gonoszabbnál gonoszabb toolkitek, nem meglepő, hogy a mai operációs rendszereken némiképp magasabbra került a léc:

- a csatorna forgalma valóban titkosított
- a végpontok eltéríthetetlenségéről digitális aláírással gondoskod*(hat)*unk

Az alábbi ábrán egy megkímélt állapotban lévő, soha agyon nem menedzselt tartomány alapértelmezett biztonsági beállításait láthatjuk, mely szerint - ha a túloldali gép ezt lehetővé teszi *(when possible)* – titkosít ÉS aláír minden adatot, ami a csatornában folyik.



Egy tartományvezérlő alapértelmezett Secure Channel beállításai. Ha lehetséges (when possible), a csatorna adatai titkosított és aláírt formában közlekednek

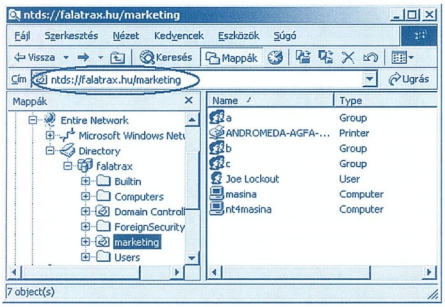
Erősíteni oly módon lehetne a biztonságon, hogy kötelezővé tesszük mind a digitális aláírást, mind a titkosítást – és elbúcsúzunk szeretett Windows 9x-einktől *(legfelső sor a keretben)*. További érdekes lehetőség a Windows NT 4.0 lerúgása, mely a keret legalsó opciójával érhető el: strong kulcs = NT4 kizárás! Már a kereten kívülre szorult, csak megemlítem: a Windows 2000 alapértelmezésben nem engedélyez clear text *(ilyenkor titkosítatlan még a jelszó is)* bejelentkezést, így SaMBa kiszolgálóinkon lehet, hogy frissíteni kell egyet – vagy az itt látható opciót engedélyezni, bár én ezt nem tenném.

Mire lehet jó még a Secure Channel?

Amikor a felhasználó erőforrást keres a hálózaton, jobbára a Network Neighborhoodon kattintgat szegény. Sokszor hiába teszi, mert a jó öreg Browse szolgáltatás hajlamos percek óta bekapcsolt gépeket **nem** mutatni, míg félórája kinyomott gépeket minden további nélkül felkínál. Hogy miért? Az egy olyan cikk témája lenne, amit nem érdemes megírni. A Browse szolgáltatás elavult. Nagyvállalati, több IP alhálózattal tarkított környezetben ennyit tud és kész. Ezt kapja a józer.
De mi jár a rendszergazdának? A Windows NT 4.0 – féle Server Manager csodálatosan megmutatja, mely gépek vannak egy adott pillanatban bekapcsolva. Úgy sorolja fel kicsi okos ablakában az összes gépet, hogy kék színnel jelöli a bekapcsoltakat, és szürkével a halottakat. Honnan tudja mindezt? Hát onnan, hogy amihez aktuálisan van Secure Channel, az nyilván fut. Amihez meg nincsen, hát az bizony ki van kapcsolva. Most jön a svédcsavar! A Windows 2000-ben hála Istennek végre megszabadulhatunk az amúgy is működésképtelen Browse áldatlan és haszontalan szolgáltatásaitól, mert a My Network Places felkínálja nekünk a címtár böngészését is.

A My Network Places elágazik! A Microsoft Windows Network a jó öreg, és megbízhatatlan Browse, míg a könyvecske a címtár kutatását teszi lehetővé

Már csak egy lépésnyire vagyunk a Server Manager által mutatott valós képtől. Hagyjuk a csudába a Microsoft Windows Networköt, és menjünk bele a Directoryba! Csodálatos, hierarchikus világ tárul majd a szemünk elé, melyben ott lesznek a cég szervezeti egységei, felhasználói, nyomtatói és számítógépei. S ha egy csöpp szerencsénk van, a kikapcsolt számítógépek szürkék, míg a bekapcsoltak kékek lesznek, mint annak idején az NT4 Server Managerében...



Figyeljük meg az Explorer által mutatott címet! Az NTDS:// (NT Directory Services) arra utal, hogy a címtárban, az Active Directoryban kutakodunk. Én, Józer Jolán, hozzáférek a címtárhoz!

Megérkeztünk a címtárba. Látunk két kék gépet: az egyik a már jól ismert MASINA, a másik pedig az NT4MASINA nevet viseli. Ha kék, akkor fut nemde? Sajnos nem. Az Active Directory sok nélkülözhetetlen szolgáltatást hozott ugyan az NT4-hez képest, de most rábukkantunk egyre, amit elveszített. **Nem** tudja megmutatni, hogy egy géphez van-e Secure Channel, azaz fut-e. :-(Mindig minden gép ikonja kék.

Ennek a sajnálatos malfunkciónak legalább két oka van.

1. Az Active Directoryt nem arra találták ki, hogy illékony adatokat tároljunk benne. Egy munkaállomás állapotinformációja sajnos túl dinamikus adat ahhoz, hogy értelme legyen betenni a címtárba. Többszáz gépes hálózaton óriási, és jobbára felesleges replikációs forgalmat generálna. Ezen a Windows .NET Server fog segíteni, ahol különleges és ravasz replikációs lehetőségek jelennek meg a címtárban.
2. Hol is van a Secure Channel vége? Ha van öt tartományvezérlőm, akkor...

Hol az alagút vége?

A Secure Channel fizikai valójában nem más, mint egy TCP csatorna, mely „örök időkig" fennáll. Ha van mondjuk öt tartományvezérlőm, vajon hogy alakul a csatorna? Mivel TCP csatornáról van szó, nyilvánvaló, hogy maximum két végpontja lehet. „Kirojtosodott", háromvégű TCP csatorna nincs, és nem is lehet *(a TCP szekvenciaszámok miatt)*. Emiatt nem meglepő, hogy a Secure Channel a munkaállomás és **valamelyik** DC között húzódik. Egyetlenegy DC áll a túlvégen. Arra a kérdésre nehezen tudnánk kapásból válaszolni, hogy az öt közül pontosan **melyik** az, de egy támpontunk lehet: a Windows 2000 munkaállomások hozzájuk közel eső, saját telephelyükön lévő DC-t keresnek, és – ha csak lehet – ahhoz csatlakoznak bootoláskor. Az NT4-ről ez korántsem mondható el. Ha nem bűvészkedünk Resource Kit bigyókkal, akkor az NT4 a leghamarabb válaszoló DC-vel lép nászra, jöjjön a válasz akár a földgolyó túloldaláról.

Ha feltelepítjük a Windows 2000 CD kincseit a Support\Tools könyvtárból, lesz egy NlTest.EXE-nk, amivel a Secure Channel kezelése is megoldható. Nem mintha állandóan babrálni kellene, de ha valakit érdekel, sok egyéb hasznos funkció mellett az NlTest ezt nyújtja a Secure Channel kezeléséhez:

Kapcsoló	Funkció *(munkaállomásról futtatva!)*
/SC_QUERY:<Domain Name>	Kiírja a Secure Channel adatait.
/SC_RESET:<Domain Name>[\<DcName>]	Kikényszeríti a Secure Channel menet közbeni lebontását és újraépítését. Így lehet visszairányítani a csatornát, ha véletlenül távoli DC-hez került
/SC_CHANGE_PWD:<DomainName>	A Computer Account jelszavának átállítása.

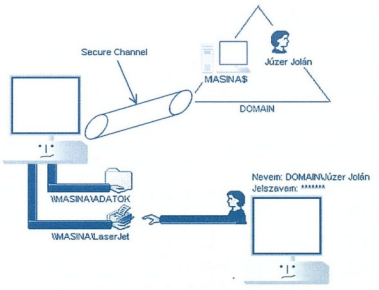
A Secure Channel valódi szerepe

Sokat beszéltünk a Secure Channelről, éppen csak az értelme hiányzik még. Ha csak Józer Jolán bejelentkeztetéséhez lenne rá szükség, akkor nem lenne rá szükség, Jolika bejelentkezhetne önmaga, anélkül, hogy *(előtte jóval)* a gép bejelentkezne. Így dolgozik a Windows 9x, és nem dől össze a világ.

A biztonságos csatorna valójában nem a titkosítás miatt kell *(láttuk, az NT4 még nem sokat bíbelődött a csatorna titkosításával)*, hanem amiatt, hogy a központi címtár pontosan tudja a „vonal" túlsó végén lévő „egyed" azonosságát. Erre azért van szükség, mert a gép – függetlenül attól, hogy Jolán dolgozik-e rajta – megosztásokat publikál*(hat)* a hálózaton, és ennek kapcsán időről időre tartománylekérdezéseket kell végeznie, hogy megállapíthassa, vajon a csatlakozni szándékozónak megvan-e ehhez a szükséges jogosultsága.

A Secure Channel fizikai valójában nem más, mint egy TCP csatorna, mely „örök időkig" fennáll.

Az alábbi ábrán egy kicsit nagy a nyüzsgés, de remélem jól szemlélteti, ahogy a MASINA *(szinte)* minden egyes megosztott erőforrás elérésekor kényszerűen a tartományvezérlőjéhez rohangál *(amelyikhez bootoláskor a Secure Channelt nyitotta)*. A tartományvezérlő pedig azért válaszol neki minden teketória és ellenőrzés nélkül akár a legvadabb címtárlekérdezésre is, mert a MASINA már réges-régi jó ismerőse neki. Csatornája csak az ismerősöknek van.



MASINA nevű munkaállomásunk a Secure Channelen keresztül kérdezgeti a címtárat, vajon az erőforrásait odaadhatja-e az erre ácsingózóknak.

A tartomány alapértelmezésben - kompatibilitási okokból - sajnos nemcsak a csatornásoknak, hanem akárkinek, még Anonymousnak is

válaszol, hisz ez a feltétele annak, hogy Windows 9x-ek is használhassák a címtáradatbázist – de ezt **le lehet tiltani**, és ez a lényeg! Az én hálózatom ne borítsa oda a címtárat mindenféle jöttmentnek, csakis azonosított felhasználóknak, legyen az akár gép, akár ember!

A meghatalmazotti kapcsolat

A meghatalmazotti, vagy trust kapcsolat lényege, hogy ha egy tartomány hitelesít egy felhasználót, akkor az ő hitelesítését más tartományok elfogadják, s a felhasználónak jogosultságok adhatók. Ha idegen tartományban kopogtat a felhasználó, akkor az ottani DC-k nem kínlódnak az azonosítással, hanem megkeresik azt a tartományt, amely felelős a felhasználóért, és felkérik a név+jelszó ellenőrzésére.

A folyamat szimulálható a fentebb emlegetett NlTest.EXE programocskával. A /WHOWILL és a /FINDUSER kapcsolókkal meg lehet állapítani, melyik tartomány tudná bejelentkeztetni a parancssorban megadott felhasználót. A /FINDUSER még csak nem is kér tartománynevet: egy puszta loginnév alapján kideríti, hol lehetne őt bejelentkeztetni.

Úgy is fogalmazhatunk, hogy a trustra lépett tartományok „idegen" címtárakkal is érkezhetnek, vagyis nemcsak saját, hanem távoli címtár alapján is képesek dönteni a felhasználó beengedéséről.

Most vizsgáljuk meg a tartományhoz csatlakoztatott munkaállomást ebből a szempontból. Bejelentkezéskor két címtár közül választhatunk: a helyi SAM, és a tartományi címtáradatbázis is rendelkezésünkre áll. Vagyis a munkaállomás „idegen" címtár alapján dönt a felhasználó beengedéséről. Mi ez, ha nem trust? Ez bizony trust, ha nem is a javából, de a gyengébbjéből.

Trust tartományvezérlők között

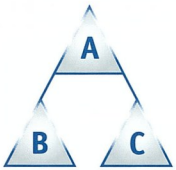
A kihalófélben lévő Windows NT4 tartományok között kézzel lehetett meghatalmazotti viszonyt felépíteni.

Meghatalmazotti kapcsolat három NT4 tartomány között. „B" felhasználói tevékenykedhetnek „A" tartományban is. „C" és „B" felhasználói kölcsönösen dolgozhatnak egymás tartományában is. De mi a helyzet „A" és „C" között?

... a Windows NT Workstation és Windows 2000 Professional olyan tartományvezérlők, melyeknek kapcsolatépítési képességeit a minimális szintre szorították.

A nyíl iránya nem tévedés: „A" tartomány a nyíllal jelölt tartományban **megbízik**. Ennek folyománya, hogy „B" felhasználói átsétálhatnak „A" tartományba, és ott bejelentkezhetnek, erőforrásokhoz kapcsolódhatnak. „B" és „C" között kétirányú, kölcsönös a bizalom: ezt két egyirányú trust létrehozásával lehet kialakítani. A nyíl tehát olyan tartományokra mutat, amelyek *(account)*forrásként szerepelnek a trustban. Ezek a kézi kapcsolatok nem tranzitívek, ami magyarul azt jelenti, hogy „A" és „C" mindaddig nem hisz egymásnak, amíg közöttük közvetlen kapcsolat nincs.

Windows 2000 esetén a gyermektartományok telepítésekor automatikusan alakul ki a trust, bár továbbra is lehetőség van kapcsolatok kézi kialakítására az Active Directory Domains and Trusts eszköz segítségével. Így ábrázoljuk az automatikus meghatalmazotti kapcsolatot:



Meghatalmazotti kapcsolat három Windows 2000 tartomány között. Itt mindegyik kapcsolat kétirányú és tranzitív, azaz „B" és „C" között is van „átjárás"

A meghatalmazotti kapcsolat alapja nem más, mint *(minő meglepetés!)* egy-egy **Secure Channel** a két tartomány vezérlői között. A két domain gépei egymáshoz éppúgy bejelentkeznek, mint azt a munkaállomásnál láttuk. Egy terminológiai különbség van csupán: azt a fiókot, mellyel bejelentkeznek egymáshoz, nem Computer Accountnak, hanem Interdomain Trust Accountnak hívják. A neve éppúgy dollárjellel képződik, csak nem a gépnévből, hanem a tartomány nevéből. Ennek jelszavát a DC-k közösen kezelik.

Foglaljuk most össze a sok-sok operációs rendszer-változat címtárkapcsolat-képességeit egy táblázatban:

	Címtár	Kialakítható kapcsolatok száma	A kapcsolat típusa	Mire képes?	Tranzitív?
Workstation és Professional	SAM	1	Kézi, egyirányú	Cél	Nem
NT 4	SAM	Sok	Kézi, egyirányú	Forrás és cél	Nem
Server és Advanced Server	Active Directory	Sok	Automatikus, kétirányú	Forrás és cél	Igen

Igen szembetűnő a hasonlóság a munkaállomás-változatok és az NT4 között: a különbség a kialakítható kapcsolatok számában és irányában rejlik! *(A munkaállomás „címtárában" nem lehet megbízni, az ott létrehozott felhasználók és csoportok lokálisak, kitörni nem tudnak.)*

Tehát a Windows NT Workstation és Windows 2000 Professional olyan tartományvezérlők, melyeknek kapcsolatépítési képességeit a minimális szintre szorították.

Ehhez jön még a maximálisan tíz felhasználói kapcsolat a megosztott erőforrásokra, a végrehajtási szálak kezelésének különbözősége, a lemezkezelési lehetőségek *(RAID)* megnyirbálása – de ezzel gyakorlatilag vége is a sornak.

S hogy mi a fenti ismeretek gyakorlati haszna? Vizsgázáskor jól jön, ha annak látjuk a világot, ami, és nem dőlünk be a furfangos kérdéskészítők szándékos szédítéseinek!

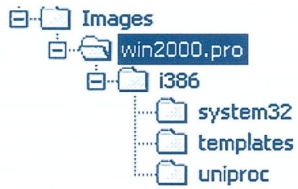
Fóti Marcell
marcellf@netacademia.net

RIS
(IV. rész)

A RIS kétféle telepítőkészlettel képes dolgozni. Az egyik a Riprep segítségével készített, a másik a CD-ROM, vagy Script alapú telepítőkészlet.
A Microsoft RIS-sel csupán a Windows XP/2000 Professional telepítését támogatja, sem korábbi operációs rendszerek, sem a Windows 2000 Server telepítését nem, azonban egy kis trükkel a Windows 2000 Servert meg tudjuk etetni a risetuppal, így CD-ROM alapú telepítőkészletet tudunk készíteni a kiszolgálókhoz is.

CD-ROM alapú telepítőkészlet

Ez a telepítőkészlet csak kicsit különbözik attól az esettől, amikor felügyelet nélküli telepítőkészletet használunk egy kiszolgálón megosztott könyvtárból, amelyhez egy DOS-os MSClient segítségével csatlakozunk. RIS esetén jó esetben nem kell bootfloppy ehhez *(ha muszáj, a RIS-es rbfg.exe-vel elkészíthető bootfloppyt használunk)*. A gyalogos válaszfájlos telepítéstől annyiban tér el, hogy itt SIF kiterjesztésű a válaszokat tartalmazó állomány, s tartalma kismértékben eltér a hagyományos unattend.txt-től.
Ez a típusú telepítőkészlet az alapértelmezett, a risetup.exe első futtatásakor menthetetlenül létre is jön egy csomag a RIS kiszolgálón a Remoteinstall\Setup\English\Images egy alkönyvtárában.



Az első telepítőkészlet a RIS Sever telepítésekor elkészül

Minden, ami a win2000.pro könyvtár alatt található, ehhez a telepítőkészlethez tartozik. Alatta létrehozhatunk egy OEM nevű könyvtárat, benne további könyvtárakat és állományokat, amelyek telepítéskor az operációs rendszerrel együtt a merevlemezre kerülnek. Például ha C:\myfiles könyvtárba szeretnénk rakatni fájlokat a telepített, kész gépre, akkor a OEM könyvtár alatt létrehozunk egy C könyvtárat *(a kötet neve alapján)*, benne a myfiles könyvtárat, abba pedig belepakoljuk amit le akarunk juttatni a telepítővel az ügyfélgépre.

Extra fájlok lejuttatása a telepítés során

Ez például akkor hasznos, ha egy batch fájlt akarunk meghívni az első bejelentkezés után a további programok feltelepítéséhez, de a lokális administrator/rendszergazda jelszava nem egyezik meg a tartománybelivel *(ennek minden helyen így kellene lennie)*, és a válaszfájlban nem is akarjuk azt megadni. Így szokás lejuttatni az ügyfelekre olyan eszközmeghajtókhoz szükséges állományokat is, amelyek nem részei az operációs rendszer telepítőkészletének.
Alapértelmezésben csak az operációs rendszer telepítését tartalmazza egy ilyen telepítőkészlet, de könnyen alakíthatjuk úgy a telepítést, hogy az a szükséges programokat is felpakolja rögtön az operációs rendszer után. Ehhez a SIF állományt kell módosítani, valamint a programokat elő kell készíteni a felügyelet nélküli telepítéshez. Az Office2000/XP-hez például az Office Resource Kitben található telepítő varázslóval gyárthatunk konfigurációs állományokat, amelyek segítségével parancssorból telepíthető az Office. Az Internet Explorer Administration Kit segítségével olyan telepítőkészlet készíthető az IE-hez, mely ráadásul teljes körű beállítási lehetőséget is nyújt *(proxy, dial stb.)*. Telepítőkészletek gyártásához a Wininstall LE, vagy a Systems Management Server 2.0-ben található SMS Installer használható. Mindkettővel telepítőcsomagba gyűrhetők az egyébként nem ilyen telepítésű, régi alkalmazások. Akár a sysdiffet is használhatjuk felügyelet nélküli telepítőkészletek gyártásához, vagy telepíthetjük a programokat Group Policy segítségével is. De térjünk vissza az operációs rendszerhez.

Alapértelmezésben csak az operációs rendszer telepítését tartalmazza egy ilyen telepítőkészlet, de könnyen alakíthatjuk úgy a telepítést, hogy az a szükséges programokat is felpakolja rögtön az operációs rendszer után.

A CD-ROM alapú telepítőkészlet nagy előnye, hogy rendkívül rugalmasan változtatható, többféleképp telepíthetjük az operációs rendszert ugyanazzal a telepítőkészlettel, csupán a válaszokat tartalmazó állományban – a SIF fájlban – kell változtatnunk. Helykímélő megoldás, mert a programokhoz is csak egyszer kell létrehozni a telepítőkészletet, azt külön batch fájlokkal szabályozva különböző végeredményhez juthatunk. Hátránya a Riprep alapúval szemben, hogy a kívánt eredmény eléréséhez kicsit többet kell foglalkozni a programok előkészítésével.

Csalás: Windows 2000 Server telepítőkészlet létrehozása

CDROM alapú telepítőkészletet a risetup segítségével tudunk

létrehozni, azonban ez Windows XP/2000 Professional telepítőkészlet létrehozását engedélyezi csupán. Be kell csapni a varázslót ahhoz, hogy megegye a Windows 2000 Server telepítőkészletet is.

Elkészítés az alábbi recept szerint:

- A Windows 2000 Server telepítő CD-ről másoljuk az I386 könyvtárat a merevlemezre.
- Egyszerű szövegszerkesztővel nyissuk meg a txtsetup.sif fájlt miután levettük az írásvédettséget.
- A **[SetupData]** részben a **ProductType** értékét állítsuk 1-ről 0-ra, majd mentsük az állományt.
- Futtassuk a risetup.exe-t, és a már ismert módon a varázsló segítségével készítsük el a kiszolgálón a telepítőkészletet.
- Keressük meg a txtsetup.sif fájlt az elkészített telepítő készlet állományai között, és állítsuk vissza a ProductType értékét 1-re.

Ezzel elkészült a telepítőkészlet, amelyet használhatunk RIS-sel, azonban mivel kiszolgálóról van szó, a hozzá tartozó SIF fájlban módosításokat kell végrehajtanunk – lásd később.

Riprep alapú telepítőkészlet

Ebben az esetben nem teszünk mást, mint a teljesen előkészített minta ügyfélgép pontos lenyomatát másoljuk a kiszolgálóra olyan formában, hogy az a másik munkaállomáson használható legyen *(klónozás)*.

A Riprep alapú lenyomatok több helyet foglalnak, mint a CD-ROM alapúak, mert ezek az ügyfél merevlemezének másolatai, amely nemcsak az operációs rendszert, hanem minden telepített programot is tartalmaznak.

Ahhoz, hogy egy kiszolgálón Riprep alapú telepítőkészletet tároljunk, már legalább egy CD-ROM alapú telepítőnek ott kell lennie. A nyelvi verziónak és a Service Pack verziónak is meg kell egyeznie. További szükséges feltétel, hogy az ügyfélnek, amelyről a telepítőkészletet készítjük, sajnos csak egy partíciója lehet. Ha több partíciója van, akkor figyelmeztető üzenet jelenik meg és a program csak a rendszerpartíciót *(amelyen a Winnt könyvtár található)* másolja le. Ebből következik, hogy az indító-*(boot)* és a rendszerpartíciónak *(system)* egynek kell lennie. Figyeljünk arra is, hogy a célszámítógépnek legalább akkora vagy nagyobb merevlemeze legyen, mint amelyről a lenyomatot készítjük, valamint ugyanaz legyen a HAL-ja *(ACPI vagy nem ACPI)*.

Alaphelyzetben a Riprep telepítőkészletből felhúzott gépek esetén a telepítés nem vizsgálja a hardvert. Ha ezt mégis szeretnénk, akkor használjuk a –pnp parancssori kapcsolót

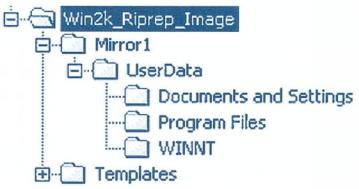
Riprep alapú telepítőkészlet létrehozásához az előkészített, **teletelepített** munkaállomásról kapcsolódjunk ahhoz a RIS kiszolgálóhoz, amelyen tárolni akarjuk a telepítőkészletet. A Reminst megosztás alól az Admin\I386 könyvtárból futtassuk a Riprep.exe programot. Ez az előkészítő varázsló *(Remote Installation Preparation Wizard)*, amely a következőket teszi:

- Ellenőrzi, hogy vannak-e nyitott állományok. Ha talál ilyet, felszólít, hogy zárjuk be azokat és indítsuk újra a programot.
- Ellenőrzi, hogy vannak-e olyan ismert vagy ismeretlen szolgáltatások, amelyek esetleg írhatnak a lemezre. Ha talál ilyen futó szolgáltatást, akkor felszólít annak leállítására.
- Bekéri annak a RIS kiszolgálónak a nevét, amelyen a lenyomatot tárolni akarjuk.
- Bekéri azt az alkönyvtárat, ahova létre akarjuk hozni a lenyomatot.
- Bekéri a megjelenítendő leírást és kiegészítő szöveget *(Friendly Description, Help text)*
- Létrehoz egy alapértelmezett Riprep.sif fájlt a SysPrep felügyelet nélküli beállításait használva.

Miután minden kérdést megválaszoltunk, a Riprep előkészíti a gépet a telepítőkészlet létrehozásához, majd elkezdi a kiszolgálóra másolni a fájlokat. Amikor ezzel végzett, kikapcsolja a munkaállomást. Ha újraindítjuk, egy varázsló fog elindulni, mintha csak a Sysprepet futtatnánk.

A Riprep telepítőkészlet elkészítése

A kiszolgálón a Riprep az alábbi könyvtárstruktúrát hozza létre:



Riprep telepítőkészlet (klón) a RIS kiszolgálón

Három fájl is létrejön, melyek a Riprep alapú lenyomatról tartalmaznak információt *(Riprep.log, Bootcode.dat és Imirror.dat)*:

- Riprep.log: Ez a fájl a Riprep.exe futásának naplója. Az esetleges hibaüzenetek *(például nyitott vagy titkosított fájlok miatt)* ide kerülnek bejegyzésre olyan információk kíséretében, mint a kiszolgáló neve, leírása stb. Ez a fájl az I386 könyvtárban található.
- Bootcode.dat: Ez a fájl a számítógép bootszektorát tartalmazza. Ez az egyik oka annak, hogy a telepítendő számítógépben legalább akkora merevlemezre van szükség, mint amiről a lenyomat készült. A fájl az I386\Mirror1 könyvtárban található.
- Imirror.dat: Ez a fájl a Ripreppel klónozott számítógépről tartalmaz információt. Nem olvasható *(hacsak nem készítünk róla egy hexa-dumpot)*. A meghajtó betűjeléről, a telepítési könyvtárról, a hardver absztrakciós réteg típusáról *(HAL)*, az ARC útvonalról és hasonlókról tartalmaz információkat.

Alaphelyzetben a Riprep telepítőkészletből felhúzott gépek esetén a telepítés nem vizsgálja a hardvert. Ha ezt mégis szeretnénk, akkor használjuk a –pnp parancssori kapcsolót a telepítőkészlet létrehozásakor **(riprep –pnp)**. Ezután működni fog a Plug and Play kiértékelés, így különböző alkatrészek esetén is működni fog a klón.

Ha egyszer elkészítettük a Riprep alapú telepítőkészletet, abban változtatni nem lehet. Ha bármit meg szeretnénk változtatni a beállításokon, akkor a mintaként használt gépről újra és újra el kell készíteni a telepítőkészletet. Sok helyet foglal, bár a SIS jótékony munkája nyomán minden állomány igazából csak egyszer foglalja a helyet a kiszolgálón. Kevésbé rugalmas megoldás ez, cserébe kevesebb nyűggel jár az elkészítése.

A RIS válaszfájlok

Az egyes munkaállomások telepítéséhez a válaszfájl a telepítőkészletek minta válaszfájljából jön létre a munkaállomás telepítése előtt közvetlenül CIW *(Client Installation Wizard)* képernyőkön megadott adatok beépítésével.

A mintafájlokat fontos előkészíteni, ugyanis ezzel tehetjük valóban felügyelet nélkülivé a telepítést, hisz az ebben található válaszok

irányítják a munkaállomások telepítését. Minden telepítőkészlethez legalább egy minta válaszfájl tartozik. Az alapértelmezett mintafájl az első Risetup első futtatásával jön létre, a telepítőkészlet I386\Templates könyvtárában ristndrd.sif néven.
Ha ennek segítségével indítunk egy telepítést, akkor az nem fog működni felügyelet nélkül, mert egy fontos dolog, a telepítéshez szükséges termékkód hiányzik belőle. Ezt az egy adatot mindenképp érdemes beletenni a [UserData] szakaszba a következő formában:

```
ProductID="XXXXX-XXXXX-XXXXX-XXXXX-XXXXX"
```

Egy telepítőkészlethez több ilyen minta válaszfájl is tartozhat, így tudunk ugyanabból a forrásból többféle végeredményre jutni. Ehhez nem kell mást tenni, mint a meglevő SIF fájlt lemásoljuk ugyanabba a könyvtárba, majd azt módosítjuk, és elmentjük bármilyen néven SIF kiterjesztéssel. Egyszerű, nem? Készíthetünk ilyen válaszfájlt a Setup Manager Wizard segítségével is, de a meglevőt is megváltoztathatjuk vele. Ez a varázsló a **Windows 2000 Support Tools** része. Ha nem akarjuk telepíteni, akkor a Windows 2000 telepítő CD-n levő \support\tools\deploy.cab fájlból szedjük ki a **setupmgr.exe** fájlt és máris varázsolhatunk. Mindenkinek ajánlom figyelmébe az ugyanitt található **unattend.doc** fájlt is, amelyben megtalálható a válaszfájlokban használható paraméterek magyarázata. Nézzük, mit tartalmaz egy ilyen válaszfájl és mit érdemes megváltoztatni benne.
A SIF fájl nem más, mint egy felügyelet nélküli telepítési válaszfájl a RIS-sel való telepítéshez szükséges kiegészítésekkel. Néhány fontosabb bejegyzés:

A [Data] szakasz

Ez a mintafájl első szakasza, amelyben a Risetup.exe helyezi el a következőket:

- floppyless = "1"
  Ebből tudja a telepítő, hogy hálózati telepítés zajlik.
- msdosinitiated = "1"
  Azt jelenti, hogy a telepítés nem CD-ROM-ról vagy floppylemezről indult.
- OriSrc="\\%SERVERNAME%\RemInst\%INSTALLPATH%\%MACHINETYPE%
  A telepítő fájlok elérési útvonala. A változók akkor kapnak értéket, amikor az ügyfél ténylegesen használja a mintát. A BINL szolgáltatás a fájlban lévő változókat helyettesíti a kiszolgáló által megadott értékekkel *(például a Servername helyére a RIS kiszolgáló neve kerül)*. Így a minta a szervezet bármely RIS kiszolgálóján használható. Ennek az értéknek a megadásával megkíméljük a felhasználót a telepítési útvonal mindenkori megadásától.
- OriTyp = "4"
  Ez az érték azért van 4-re állítva, mert távoli telepítésről van szó. Ha a telepítés helyi CD-ROM-ról indul, az érték 5.

Ezeket az értékeket a RIS kiszolgáló állítja be, és módosításukra nincs szükség.

A [SetupData] szakasz

A minta [SetupData] szakasza nem RIS-specifikus, minden felügyelet nélküli válaszfájlban használható. Ez a szakasz a telepítő szöveges módjára vonatkozik és két értéket ad meg:

- OsLoadOptions = "/noguiboot /fastdetect"
  Utasítja a Telepítőt, hogy a telepítést grafikus felhasználói felület *(splash screen)* nélkül indítsa, és használjon „fastdetect"-et.
- SetupSourceDevice="\Device\LanmanRedirector\%SERVERNAME%\RemInst\%INSTALLPATH%"

Elérési útvonal a szöveges módú telepítéshez.

Ismét megjegyzem, hogy a változókat a BINL helyettesíti be.

Az [Unattended] szakasz

Bár ez a szakasz nem RIS specifikus, mégis van pár dolog, amit itt érdemes beállítani az alapértelmezéstől eltérően.

- FileSystem = LeaveAlone
  Ez a paraméter mondja meg, hogy a partíció, amire telepítünk NTFS legyen-e, vagy sem. Érdemes ezt az értéket ConvertNTFS-re állítani.
- ExtendOEMPartition = 0
  Alapértelmezésben ez 0, de ha 1-re állítjuk, akkor kiterjeszti a partíciót amekkorára csak lehet *(csak NTFS partíciókkal tudja ezt megtenni, tehát a FileSystem paramétert ez esetben ConvertNTFS-re kell állítani)*.

A [UserData] szakasz

Alapértelmezésben így néz ki:

```
[UserData]
FullName = "%USERFIRSTNAME% %USERLASTNAME%"
OrgName = "%ORGNAME%"
ComputerName = %MACHINENAME%
```

Ristndrd.sif UserData szakasza

Látható, hogy itt is csupa változóból állnak az értékek, amelyek a konkrét telepítés során kapnak valódi tartalmat.
Már az előbb szóltam arról, hogy ide be kell rakni egy ProductID nevű paramétert, mert anélkül a telepítő meg fog állni és várni fog a kulcs megadására.

A [GuiUnattended] szakasz

Az alapértelmezett beállítások mellett itt lehet megadni a lokális Administrator jelszavát, valamint azt is, hogy belépjen-e a telepítés befejezése után az első boot után további parancsok végrehajtása végett, vagy sem.
Annak érdekében, hogy a helyi Administrator automatikusan belépjen az első boot után, adjuk hozzá ehhez a szakaszhoz az AutoLogon paramétert 1-es értékkel, majd az AutoLogonCount paraméterrel adjuk meg hányadik bootig lépjen be automatikusan.

```
[GuiUnattended]
    AdminPassword=po98iu7
    AutoLogon=Yes
    AutoLogonCount=1
    OEMSkipRegional=1
    TimeZone=%TIMEZONE%
    OemSkipWelcome=1
```

Egy átalakított [GuiUnattended] szakasz

- OEMSkipReginal = 1
  Ez a paraméter biztosítja, hogy a telepítő grafikus szakaszában kimaradjanak a Regionális beállításokra vonatkozó képernyők. Létrehozható egy **[RegionalSettings]** szakasz is, amellyel szabályozhatóak a Regionális beállítások.
- OEMSkipWelcome = 1
  Ez a beállítás biztosítja, hogy a telepítés során kimaradjon a legelső üdvözlő képernyő.
- TimeZone = %TIMEZONE%
  Ennek a paraméternek magunk is adhatunk értéket. Minden időzónának van egy két, vagy három számból álló kódja. Ha így hagyjuk, ahogy fentebb látható, akkor a kiszolgáló időzónáját fogja felvenni a munkaállomás.

A [RegionalSettings] szakasz

Ez nem szerepel az alapértelmezett válaszfájlban. Az alábbi pél-

dán az látható, hogy az operációs rendszer és a felhasználók alapértelmezett területi beállítása a magyar, valamint a billentyűzet kiosztások közül a magyar és az angol van telepítve.

```
[RegionalSettings]
    LanguageGroup=1,2
    SystemLocale=0000040e
    UserLocale=0000040e
    InputLocale=040e:0000040e, 0409:00000409
```

Magyar területi beállítások

A LanguageGroup kódokról bővebben a már emlegetett unattended.doc-ban lehet olvasni, csak annyit hogy ez adja meg a telepítendő nyelvi csoportokat. Ahhoz, hogy magyar területi beállítás legyen, kell a Central Europe nyelvi csoport is, ez a 2-es kódú, míg az egyes a Western Europe and United States nyelvi csoport.
A SystemLocale és UserLocale paraméterhez is kódokat kell rendelni, amelyeket a Microsoft weblapjain találhatunk meg. A fenti példánál maradva a 409 végű az angol, a 40e végű a magyar. Az InputLocale paraméternek megadott első érték lesz az alapértelmezett, esetünkben a magyar.

A [Components] szakasz

Ő sem található meg az alapértelmezett válaszfájlban, de érdemes létrehozni, ugyanis így tudjuk szabályozni, hogy mely eszközök településenek, illetve ne településenek az operációs rendszerrel.
Nagyon hosszú lenne a sor, ha mindet felsorolnám, inkább utalok újból az unattend.doc fájlra. Azért néhány példa:
A hyperterminál letiltása:

- hypertrm = off

Néhány játék letiltása *(de akkor mire jó az egész? A szerk.)*:

- minesweeper=off
- pinball=off
- solitaire=off

A [GuiRunOnce] szakasz

A CD-ROM alapú telepítőkészleteknél fontos ez a rész, itt adhatjuk meg az első automatikus belépés utáni parancsot például az alábbi formában:

- Command0="c:\Install.bat"

A példában szereplő Install.bat-ot az ismertetett OEM módszerrel lejuttatjuk az ügyfélre, a batch segítségével pedig egy hálózati meghajtóról telepíthetjük a szükséges alkalmazásokat.

A [RemoteInstall] szakasz

- Repartition = Yes vagy No
  Ha Yes az érték, akkor a munkaállomásról minden partíciót letöröl a telepítő, majd létrehoz egy újat. Ezt a paramétert elhelyezhetjük a válaszfájlok **[Unattended]** szakaszába is. Ha No az értéke, akkor a telepítő megáll ott, ahol a telepítés során a partícionálást lehet megtenni.
- UseWholedisk= Yes vagy No
  Yes érték esetén az egész merevlemez felhasználásra kerül.

Az [Oschooser] szakasz

Az **[Oschooser]** szakasz néhány olyan változót tartalmaz, amelyek megjelennek az operációs rendszer kiválasztásakor. Amikor először futtatjuk a Risetup.exe fájlt, vagy új lenyomatot helyeztünk el a kiszolgálón, az akkor megadott név és leírás kerül a fájlnak ebbe a szakaszába. Alapértelmezett beállítások a következők:

- Description ="Microsoft Windows 2000 Professional"
  Operációs rendszer választásakor ezt a nevet fogja látni a felhasználó.
- Help ="Automatically installs Windows 2000 Professional on the computer without prompting the user for input.
  A CIW-ben megjelenő leírás.
- LaunchFile="Installpath\Machinetype\Templates\Startrom.com"
  A megadott fájl letöltésére és futtatására utasítja a munkaállomást. A fájl nem Win32® API formátumú futtatható állomány, hanem rendszerindító fájl *(boot image)*, mint például a Startrom.com, vagy más független szoftverfejlesztők által készített PXE program.
- ImageType = "Flat"
  A lenyomat típusát jelzi. Ne változtassuk meg, mivel a CD-ROM és a Riprep alapú SIF-ek nem felcserélhetőek.
- Version="5.0"
  Az operációs rendszer verziója.

Az itt említett változók közül az első kettő kivételével ne változtassuk a többit, mert nem cserélhetők fel a Riprep alapú és a CD-ROM alapú telepítéshez használt válaszfájlok.
Az utolsó részben megnézzük hogyan lehet több telephely esetén a RIS kiszolgálókat szinkronban tartani, valamint az Ügyfél telepítő Varázsló *(CIW)* képernyőket fogjuk átszabni.

Dorner Csilla
MCSE

Internettörvény vagy az elektronikus kereskedelem szabályozása?

A 2001. év karácsonyán az információs társadalom tagjai is megkapták a maguk ajándékát a törvényhozástól – az Internettörvényt *(2001. évi CVIII. törvény)*. Kérdés, hogy valóban Internettörvény született-e, azaz a jogalkotó ténylegesen kiterjesztette-e a jogi szabályozás hatósugarát a háló valamennyi jelenségére, vagy csupán az elektronikus kereskedelem, a gazdasági kapcsolatok körén belül maradt.

Kérdésünkre a választ a törvény hatályát szabályozó szakasz elemzése adja meg:

A törvény az elektronikus kereskedelmi szolgáltatást olyan információs társadalommal összefüggő szolgáltatásnak definiálja, amelynek célja áruk, illetőleg szolgáltatások üzletszerű értékesítése, beszerzése, cseréje.

Mint látjuk, a kereskedelem hagyományos fogalma tevődik át az információs társadalmi szintre. A teljességhez tehát elengedhetetlen az információs társadalommal összefüggő szolgáltatás fogalmi meghatározása is, amely a törvény értelmezésében:

„elektronikus úton, távollevők részére, ellenszolgáltatás fejében nyújtott szolgáltatás, amelynek igénybevételét a szolgáltatás igénybevevője egyedileg kezdeményezi, továbbá mindazon ellenszolgáltatás nélkül, elektronikus úton, távollevők részére, az igénybevevő egyedi kezdeményezésére nyújtott szolgáltatás, amelyek a szolgáltató, illetve az igénybevevő részéről nem az Alkotmány által biztosított véleményszabadság gyakorlásának körébe tartoznak."

E kissé szokatlan megfogalmazásból látszik, hogy addig, amíg az e-commerce *(elektronikus kereskedelem)* témaköre alatt hagyományosan a B2B *(business to business)* vagy B2C *(business to consumer)* kapcsolatot értjük, addig a törvény a C2C, vagyis **az állampolgárok egymás közötti kapcsolatát is a szabályozás hatálya alá vonta.** *(!)*

Az, hogy ténylegesen mi tartozik a véleménynyilvánítás szabadsága körébe, nyilván a bírói gyakorlat függvényében alakul majd ki.

A B2B kapcsolat hagyományosan a nagykereskedelem, ahol az üzleti élet egymással tartós kapcsolatban álló résztvevői egymás között speciális zárt hálózaton, egyedi kommunikációs szabályok szerint, kidolgozott szerződéses keretek között kötik meg a jogügyleteket. E körben tehát kevésbé szükséges a fokozott jogi védelem. A B2C nyilvános hálózaton, a fogyasztók részére történő értékesítés, ahol jellemzően a szolgáltató által diktált egyoldalú szerződéses feltételek *(Általános Szerződési Feltételek)* alapján zajlik az üzletkötés, ismeretlen felek közötti alkalmi jogügyletek jönnek létre. Hagyományosan ez az a terület, amely speciális jogi szabályozásért kiált, hiszen itt van kiszolgáltatva a fogyasztó, és itt kell a jogalkotónak konkrét szabályokkal biztosítania a védelmet. Ezért is tűnik az európai szabályozástól eltérő magyar definíció kissé szerencsétlennek, mivel nem ez a jogalkotói szándék látszik a szövegezés mögött. Ellenkezőleg, **a törvény nem a speciális védelem biztosítására, hanem az Internet lehető legszélesebb körű szabályozására törekszik** akkor, amikor az ingyenesen, nem ellenszolgáltatás fejében nyújtott információs társadalmi szolgáltatást is hatálya alá vonja.

A törvény másrészt kiveszi hatálya alól az elektronikus levélcím fenntartását, mint információs szolgáltatást, és az elektronikus magánlevelezést.

A területi hatály is érdekesen került meghatározásra, mivel a Magyar Köztársaság területéről nyújtott információs társadalmi szolgáltatás mellett az ide irányulót is a szabályozás körébe kívánja vonni. A törvényértelmező rendelkezése szerint azt, hogy a Magyar Köztársaság területére irányul-e a szolgáltatás elsősorban **a tartalomból, a használt nyelvből, a feltüntetett fizetőeszközből** kell*(ene)* megállapítani.

A törvény hatálya alá tartozó szolgáltatókat meglehetősen széleskörű adatszolgáltatási kötelezettség terheli. Elektronikus úton közvetlenül és folyamatosan közzé kell tenniük a nevüket, képviselőjük nevét, lakcímét, székhelyét, telephelyét, elérhetőségeit *(különösen az e-mail címet)*, amennyiben a tevékenység végzéséhez valamilyen nyilvántartásba vételi kötelezettség teljesítése szükséges, úgy a nyilvántartásba vevő hatóság megnevezését és a nyilvántartásba vételi számot, engedélyköteles tevékenység esetén az engedélyező hatóságot és az engedély számát, a szakmai és etikai előírásokra való hivatkozást azok elérhetőségének megjelölésével, a szolgáltató adószámát, szakmai kamarai tagságát, tudományos szakmai fokozatát és megszerzésének helyét, akkreditáló okiratának adatait, elérhetőségét, minősítését, a fogyasztóvédelmi szabályoknak megfelelő tájékoztatást.

> ***... a törvény a C2C, vagyis az állampolgárok egymás közötti kapcsolatát is a szabályozás hatálya alá vonta. (!)***

A törvény már ismertetett módon kiszélesített hatálya azonban azt eredményezi, hogy ha nem soroljuk a véleménynyilvánítás szabadsága körébe tartozónak a hobbi site-ok számottevő részét, akkor pl. kedvenc receptjeit a hálón ingyenesen hozzáférhetővé tevő akadémikusnak is **minden felsorolt információt közölnie kell magáról.**

Az adatszolgáltatási kötelezettség körében felsorolt engedélyre vonatkozó információadást külön szükséges kiemelnünk, mivel

az európai szabályozással összhangban a törvény kifejezetten kizárja azt, hogy magát az információs társadalmi szolgáltatás folytatását külön engedélyhez kössék. Ilyen szolgáltatást bármely természetes, illetve jogi személy, vagy jogi személyiség nélküli szervezet végezhet. Az előírt engedély ez esetben tehát arra az alaptevékenységre *(pl. csomagküldő tevékenység, on-line újság kiadása stb.)* vonatkozik, amelyet elektronikus úton, távollevők részére, ellenszolgáltatás fejében nyújtanak.

A felsorolt, és a szolgáltató személyével összefüggő adatszolgáltatási kötelezettségen túl a törvény további, a szolgáltatás természetével összefüggő információk folyamatos közlését írja elő, így a szolgáltatás ellenértékhez kötését, az ellenérték mértékét, a teljesítés módját ugyancsak folyamatosan, elektronikus úton közvetlenül hozzáférhetővé kell tenni. A tájékoztatásnak egyértelműnek és közérthetőnek kell lennie, és arra is ki kell terjednie, hogy az ellenérték a közterheket is tartalmazza-e, illetve az igénybevevőhöz eljuttatás költségeit fedezi-e.

A törvény szankcionálja azokat a szolgáltatókat, akik nem tesznek eleget közlési kötelezettségüknek, mivel a **Hírközlési Felügyelet 500000.-Ft-ig terjedő bírságot szabhat ki** a kötelezettségét megszegő szolgáltatóval szemben. Természetesen a szolgáltató nem mentesül az egyéb jogszabályban, a tevékenységre specifikusan vonatkozó előírások betartása alól sem, így halmozott bírság is elképzelhető, ha pl. a kereskedőt a Fogyasztóvédelmi Felügyelőség is elmarasztalja.

Az elektronikus kereskedelem témakörében az egyik legérzékenyebb kérdés, hogy milyen pillanattól jön létre a szerződés, és ehhez milyen aktív tevékenység szükséges a felek részéről. Ha egyszerű klikkeléssel a fogyasztó kötelezettséget vállalna az oldalon hirdetett áru átvételére és ellenértékének kifizetésére, számos pert lehetne indítani az ily módon létrejövő szerződések megtámadására, amikor a megtámadási ok a tényleges szerződéses akarat hiánya, tévedés vagy megtévesztés lenne. Az Európai Unió 2000/31 Irányelvének jogi megoldásával összhangban ezért a magyar jogi megoldás is több lépcső egymásra épüléséhez köti a szerződés létrejöttét. Első feltétel, hogy a fogyasztó még az ajánlat elküldését megelőzően hozzáférhessen az általános szerződési feltételekhez oly módon, hogy azt tárolhassa és bármikor előhívhassa. A szolgáltató továbbá előzetesen köteles tájékoztatni a fogyasztót a szerződés megkötéséhez szükséges technikai lépésekről, arról, hogy azt írásba foglalt szerződésnek minősíti-e, a szolgáltató iktatja-e a szerződést, az iktatott szerződés utóbb hozzáférhető lesz-e, a fogyasztó az esetleges hibás adatbevitelt utóbb hogyan tudja korrigálni, a szerződés nyelvéről, valamint arról a magatartási kódexről, amelynek a szolgáltató alávetni magát. A felsorolásból két tényezőt érdemes kiemelni: Az írásbeliséghez egyes jogszabályok jogkövetkezményeket fűznek, így bizonyos szerződések érvényességéhez elengedhetetlen azok írásba foglalása. Az elektronikus úton kötött szerződések tehát nem a jogszabály erejénél fogva minősülnek írásbelinek, hanem csak akkor, ha azokat a szolgáltató akként kezeli. Az elektronikus aláírásról szóló törvény szerinti minősített vagy fokozott biztonságú aláírással ellátott okiratok nyilvánvalóan írásbelinek is minősülnek, más esetben azonban az írásbeliséghez fűzött jogkövetkezmények már nem állapíthatók meg ilyen egyértelműen. A másik sajátosság a magatartási kódexre hivatkozás, amely a co-regulation, azaz az állami- és az önszabályozás együttességének előrevetítése, ugyancsak az Európai Unió irányelvével összhangban.

A törvény szabályozása kogens, amely azt jelenti, hogy **a felek közös akarattal sem térhetnek el a törvény előírásaitól**. Azonban a B2B viszonyban, vagyis az üzleti szférán belüli szerződéses kapcsolatok körében a jogalkotó maga is megengedi az eltérést a szolgáltatót terhelő speciális információ szolgáltatási kötelezettségek vonatkozásában.

A fogyasztó további biztonságát szolgálja az, hogy számára lehetőséget kell biztosítani az adatbeviteli hibák azonosítására és kijavítására. Ha ilyen lehetőséget nem biztosít a fogyasztó számára a szolgáltató, úgy a fogyasztó ajánlatához nincs kötve.

A szerződés létrejöttéhez a polgári jog általános szabályai szerint is a két fél egybehangzó akaratnyilvánítása szükséges. Az akaratnyilvánítást azonban nem szükséges a feleknek egyidejűleg vagy azonos helyszínen megtenniük, a nyilatkozatok időpontja és módja is eltérhet egymástól. A távollevők közötti szerződéskötések tipikusan ilyen időben és helyszín tekintetében különböző nyilatkozatokkal jönnek létre. Az üzleti életben ezért alakult ki hagyományosan az ajánlat – és annak elfogadása, mint két egymástól eltérő nyilatkozat, amelyek együttesen hozzák létre a szerződést a felek között. Az elektronikus úton kötött szerződéseknél tehát a szolgáltató ajánlattételi felhívást tesz közzé, amelyre a fogyasztó a megfelelő technikai úton megteszi a maga ajánlatát. Ez többnyire e-mail küldéssel vagy egyszerű klikkeléssel történik. A fogyasztó ajánlatához ezt követően **48 óráig kötve van**, vagyis ha a szolgáltató ez időtartam alatt visszaigazolja az ajánlat megérkezését, vagyis azt elfogadja, a szerződés létrejön. Az ajánlat és annak visszaigazolása akkor tekinthető a másik félhez megérkezettnek, amikor az számára hozzáférhetővé válik. Tehát nem az elolvasás vagy feldolgozás ténye, hanem a hozzáférhetővé válás a döntő. A jogalkotó a szubjektív tudomásszerzés helyett az objektív megérkezést tette relevánssá. E tekintetben természetesen harmonizál a megoldás a hagyományos polgári jogi megoldásokkal, hiszen a hagyományos postai úton küldött ajánlat esetében sem annak elolvasása, hanem a címzettnek történő elküldése és az oda való megérkezése a szerződést keletkeztető tény.

A B2B körben itt is megengedett az eltérés, tehát a felek megállapodhatnak abban, hogy az ajánlat visszaigazolását nem igénylik, vagy hogy a szolgáltató nem biztosítja az igénylőnek az adatbeviteli hibák korrigálásának lehetőségét. Az ajánlat és a visszaigazolás megérkezésére vonatkozó előírásoktól azonban még e körben sem térhetnek el a felek.

A törvény hatályának elemzésekor említettekkel szemben az elektronikus úton kötött szerződésekre vonatkozó szabályokat kiterjeszti a jogalkotó az elektronikus levelezés vagy azzal egyenértékű kommunikáció *(e-mail, spam)* útján létrejövő szerződésekre is.

Az elektronikus kereskedelem körén túl – amint azt a már elemzett jogalkotói szándék is mutatja – az Internet számos egyéb kérdésre próbál meg jogalkotói választ adni a törvény. Így az amerikai és európai szabályozással összhangban rendezi a szolgáltatók egyes tevékenysége esetén felmerülő felelősség kérdését, továbbá a szerzői jog jogsérelem esetére bevezeti a notice and take down eljárást.

E speciális rendelkezések ismertetését a következő számban találják meg.

Dr. Mayer Erika
Nádas & Mayer Ügyvédi Iroda

ASP Suli:
Wap alapok

Előző számunkban bemutattuk a WML nyelv alapjait. Megtudtuk, hogy a wapos szolgáltatások kiszolgáló oldalról gyakorlatilag alig különböznek a hagyományos webes szolgáltatásoktól – ilyen alapon néhány apró beállítás után akár az Internet Information Services is képes wapos kiszolgálóként üzemelni. Akkor pedig már annak sincs akadálya, hogy a WML oldalakat dinamikusan, az ASP segítségével állítsuk elő!

Mielőtt nekikezdenénk...

... megismételném az előző számban már leírtakat. A wapos fejlesztés egyik legnagyobb problémája, hogy a szintaktikai és egyéb hibák általában csak a WML kódot lefordító átjárón, a WML gateway-en derülnek ki. Ha a tesztelést hagyományos telefonról végezzük, ez a gateway a mobilszolgáltató szerverén működik, és a hibák pontos jellegéről nem kapunk értesítést sem a telefon, sem pedig a kiszolgáló irányába. Ennél sokkal jobb, ha olyan eszközöket használunk a teszteléshez, amelyek a szemünk láttára elvégzik a gateway funkciókat is. Egy ilyen, nagyon jól használható fejlesztőkészlet az Openwave **[2]** címéről letölthető SDK és emulátor, amelyet cikkünk során is használni fogunk. Az SDK 4.1-es verziója rövidebb, kompaktabb, jól használható, az 5.0-s pedig szinte egy teljeskörű fejlesztői környezetet ad a kezünkbe. A Nokia fejlesztői weboldaláról **[4]** letölthetők a cég telefonjainak Java alapú emulátorai és a hozzá tartozó fejlesztői környezet is, de ezek inkább csak utólagos nézegetéshez jók, mert – hogy is fogalmazzam finoman – kicsit instabilak.

Tartalomtípusok

A legfontosabb talán az, hogy az IIS-t meg kell tanítanunk a WML-es tartalomtípusok kezelésére. Hiába van ugyanis a WML tartalom már a kiszolgálón, az első wapos próbálkozás érdekes, eddig valószínűleg ritkán látott hibaüzenettel válaszol:

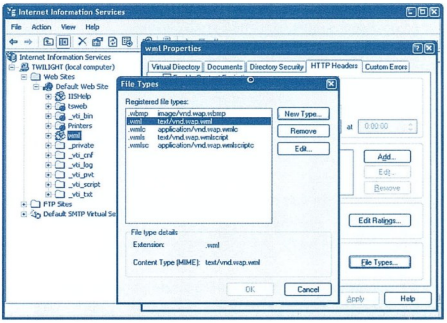
```
HTTP Error 406 - Not acceptable
```

A böngésző ugyanis *(esetünkben a telefon)* minden kérésben meghatározza az általa kezelni képes tartalomtípusokat *(mint például: text/vnd.wap.wml)*. Ez a hibaüzenet akkor keletkezik, ha a kérés alapján a kiszolgáló biztos benne, hogy nem képes az ügyfél számára emészthető tartalomtípust produkálni – például mert egyiket sem ismeri. A wapos környezetben legelterjedtebb tartalomtípusok a következők:

Kit.	MIME típus	Leírás
.wml	text/vnd.wap.wml	WML kód
.wbmp	image/vnd.wap.wbmp	WBMP képformátum
.wmls	text/vnd.wap.wmlscript	WML script kód
.wmlc	application/vnd.wap.wmlc	WML bájtkód
.wmlsc	application/vnd.wap.wmlscriptc	WMLScript bájtkód

Az elterjedtebb WAP-os tartalomtípusok

A táblázat utolsó két sorában a bájtkódra előre lefordított WML és WMLScript oldalak MIME típusa található, ezeket nem szükséges feltétlenül felvennünk, hacsak nem használjuk. A többit viszont igen: válasszuk ki a kiszolgálón a kívánt mappát, majd a tulajdonságlap HTTP Headers oldalán kattintsunk a File Types gombra, és sorra vegyük fel a fenti tartalomtípusokat.



WML tartalomtípusok az IIS-ben

A kiterjesztésekkel a webkiszolgálónak segítünk, de majd látni fogjuk, hogy egy-egy tartalomtípust nemcsak a kiterjesztés alapján, hanem – természetesen – ASP-ből is beállíthatunk, így nem lesz gond, ha a mobiltelefon épp egy .asp oldalt próbál megnyitni.

Az első lépések

Kezdjük a munkát először egy statikus oldallal. Ha az menni fog, már bátran próbálkozhatunk a dinamikus működéssel. Lássunk egy Helló Világ példát! *(A példaprogramok – szokás szerint – letölthetők az* **[1]** *címről.)*

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
↳ 1.1//EN" "http://www.wapforum.org/
↳ DTD/wml_1.1.xml">

<wml>
<card id="card1" title="WML teszt">
  <p align="center">Helló Világ!</p>
</card>
</wml>
```

Mentsük el a fenti kódot mondjuk index.wml néven. *(Ha kell, ne felejtsük el átállítani a könyvtárak alapértelmezett dokumentumát erre,*

különben csak a közvetlen hivatkozások fogják megtalálni az oldalakat!). Ha valami ilyesmit látunk, a beállítások sikerültek. Ha nem, akkor ellenőrizzük, milyet hibát jelez a toolkit *(szintaxishiba esetén nyilván a kódot, adattípus-probléma esetén pedig az IIS beállításait nézzük át).*

A dolog persze akkor igazán élvezetes, ha az ember rögtön igazi telefonról, élesben is kipróbálja. Ilyenkor érdemes indítani egy Network Monitort, és figyelni a kiszolgáló forgalmát.

A kommunikáció

A kiszolgálóhoz érkező kérés egy valós próbálkozás esetén a következő volt:

```
GET /wml/ HTTP/1.1
Accept: application/vnd.wap.wmlc
Accept: application/vnd.wap.wmlscriptc
Accept: image/vnd.wap.wbmp
Accept: application/vnd.wap.wtls-ca-cert
Accept: image/gif
Accept: text/plain
Accept: application/x-NokiaGameData
Accept: text/vnd.wap.wml
Accept: text/vnd.wap.wmlscript
Accept-Charset: ISO-8859-1
Accept-Charset: UTF-8;q=0.8
Accept-Charset: ISO-10646-UCS-2;q=0.6
Accept-Language: hu
Host: localhost
User-Agent: Nokia3330/1.0 (04.30)
Via: WAP 1.2 212.51.126.2:9201
Pragma: no-cache
```

A kérésből látható, hogy a böngésző tisztességesen be is mutatkozik *(User-Agent sor, esetünkben egy Nokia 3330).* Ennek később még lesz jelentősége, ez az információ ugyanis bekerül a Request objektum ServerVariables kollekciójába, és segítségével nagyszerűen szét lehet majd válogatni a valódi böngészőktől illetve a mobil készülékekről érkező kéréseket. A Via: fejlécben a wap gateway típusa *(itt még az IP címe és a használt port is)* látható. Érdemes még megfigyelni a feldolgozható MIME típusokat *(Accept: fejlécek; érdekességképpen megemlíteném pl. az image/gif típust!),* illetve az elfogadott karaktertáblák *(Accept-Charset:)* listáját is. A tapasztalat azt mutatja, hogy bár a telefon nem jelzi, hogy képes lenne a közép-európai *(ISO-8859-2)* kódtábla kezelésére, általában nincs gondja vele. Ha ez mégsem menne, tisztább és egyszerűbb rögtön UTF-8 kódolással dolgozni. A kiszolgálótól érkező válaszban tulajdonképpen semmi különös nincs, egyetlen dolgot leszámítva: a Content-Type fejléc a szabályos WML tartalomtípust jelzi:

```
Content-Type: text/vnd.wap.wml
```

Képek előkészítése

A WML-ben általánosan használható speciális képformátum, a .wbmp előállításához többféleképpen is nekifoghatunk. A wapos fejlesztői csomagok többsége tartalmazza a megfelelő képkonvertáló eszközöket *(vagy akár Interneten keresztül is konvertálhatunk),* de a **[3]** címről letölthető egy ingyenes plug-in, amely Photoshop illetve Paint Shop Pro bővítményként funkcionál. A komponens leírás szerinti telepítése után a kívánt képet egyszerűen csak el kell mentenünk .wbmp formátumban.

Működésben a .wbmp plug-in

Ügyeljünk arra, hogy bár a program képes gyakorlatilag tetszőleges méretű képek mentésére, a készülékek korlátozott képességeit is figyelembe kell vennünk. Az egyik határ a képek bájtmérete: a legtöbb jelenleg elterjedt telefon legfeljebb 1.3 kilobájt méretű képeket kezel. A másik korlátozás pedig a képek mérete *(képpontban),* ezt nyilván a telefonok kijelzőjének felbontása korlátozza. A Nokia fejlesztői fórumának weboldaláról **[4]** letölthető dokumentációk alapján a nálunk ismertebb Nokia készülékek alapján ezek az értékek az alábbiak *(természetesen ezek az információk megtalálhatók más gyártók készülékeihez is):*

Típus	Képernyő	Maximális képméret	Maximális bájtméret
3330	84x48	78x30	2,8kB
5510	84x48	78x30	2,8kB
6210	96x60	96x...	1,3kB
6510	96x65	92x45	2,8kB

Néhány elterjedt Nokia telefon karakterisztikája

A legtöbb készülék képes a grafika görgetésére *(legalábbis képnézegető üzemmódban, bár lehetséges, hogy csak függőleges irányba),* ezért ami igazán korlátozza a lehetőségeinket, az a kép szélessége *(általában nem érdemes 70 képpontnál szélesebb grafikával próbálkozni),* valamint a bájtméret. Ha elkészültünk, a kész .wbmp állományt tegyük fel a webkiszolgálóra, majd gyártsunk hozzá egy WML fájlt is:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
↳ 1.1//EN" "http://www.wapforum.org/
↳ DTD/wml_1.1.xml">

<wml>
<card id="card1" title="WBMP teszt">
  <p align="center">
    <img src="shrek.wbmp" alt="Shrek"/>
  </p>
</card>
</wml>
```

Jöhet az ASP!

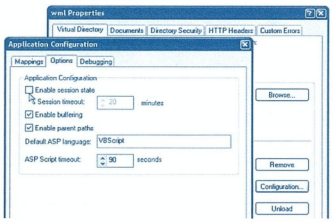
Ha a natív WML fájlok kiszolgálása már nem okoz gondot, elkezdhetjük a dinamikus oldalak készítését. Az ASP oldalakat feldolgozó motor – ha ezzel ellentétes utasítást nem kap –, tartalomtípusként a text/html-t állítja be, amit a wapos ügyfél természetesen nem képes feldolgozni. Ezért mindig első dolgunk legyen a megfelelő tartalomtípus beállítása *(default.asp):*

```
<%
  Response.ContentType = "text/vnd.wap.wml"
%>

<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
↳ 1.1//EN" "http://www.wapforum.org/
↳ DTD/wml_1.1.xml">

<wml>
<card id="card1" title="ASP teszt">
  <p align="center"><%=Now()%></p>
</card>
</wml>
```

Ha ezzel megvagyunk, nagy baj már nem lehet. Nyissuk meg az oldalt, próbáljuk ki, mi történik: ha minden jól ment, a telefon képernyőjén megjelenik az oldal lekérésének időpontja *(köszönhetően a kódban található <%=Now()%> sornak)*.
Ha megnézzük a kiszolgáló által adott válasz fejléceit, láthatjuk, hogy abban szerepel egy, az alábbihoz hasonló sor:

```
Set-Cookie: ASPSESSIONIDGQQQGEZC=
↳ JJINAFJAKOHKICFFGIMACDCK; path=/
```

A kiszolgáló tehát sütivel bombázza a telefonunkat. Az ASPSESSIONIDxxxxxxxx nevű cookie egyébként – ha emlékszünk még – arra jó, hogy a kiszolgáló felismerhesse az azonos munkameneten belül feldolgozandó kéréseket. Ennek köszönhető többek között, hogy az ASP kódban használhatjuk a Session objektumot, mint tárolót. Sok telefon azonban ma még nem kezeli a cookiekat, így ebben a tekintetben bizony vissza kell mennünk az elsőmásodik generációs böngészők szintjére. Ha ez így van, sajnos nagy az esély rá, hogy az ASP Sessionállapotok kezeléséről is le kell majd mondanunk. Ha viszont nem használjuk, a munkamenetek kezelését akár le is tilthatjuk, ezzel egyrészt megszüntetjük az automatikus cookie-küldést, másrészt pedig biztosítjuk, hogy még véletlenül sem csábulunk el programozás közben :-)



A munkamenetek kezelése ASP alkalmazásonként letiltható

A cookie-t nem támogató böngészőkhoz létezett egyébként megoldás régebben: egy telepíthető ISAPI filter minden kimenő oldalt ellenőrzött, majd a cookie-ba nem rakható információt utólag csatolta az oldalban található hivatkozások *(URL-ek)* végére. A bejövő kérésekben pedig ugyanezeket felismerte, és így létrejöhetett a munkamenet. A probléma ezzel a megoldással az, hogy az URL-ek bővítése meglehetősen megnöveli a kód méretét *(hiszen minden egyes hivatkozás végére odakerülne a cookie-k összes adata)*, amit a WML amúgy is szűkös keretei miatt nem engedhetünk meg magunknak. Kénytelenek vagyunk tehát más, állapotfüggetlen megoldást találni. *(Vagy várunk egy kicsit, míg elterjednek az „okosabb" telefonok: Nokia 3330 például már kezeli a munkamenet-cookie-kat)*.

Ki kopog?

Sok wapos szolgáltatás ugyanazon a címen érhető el mobiltelefonon, mint hagyományos böngészőn keresztül. Ilyenkor a kiszolgáló válogatja szét és irányítja az ügyfeleket a megfelelő tartalomtípushoz, és ehhez a kérésbe kódolt információkat használja fel.
A legkézenfekvőbb mód a böngésző típusának felismerése lenne, ha már olyan szépen bemutatkoznak, például *(de lásd még:* **[5]***)*:

```
Nokia6210/1.0+(05.02)
Nokia3330/1.0+(04.30)
Alcatel-BE4/2.0+UP/4.1.19e
EricssonR320/R1A
EricssonT39/R201
Mozilla/1.22 (compatible; MMEF20; Cellphone;
↳ Sony CMD-Z5)
```

A biztos azonosításhoz ez akár kevés is lehet *(lásd az utolsó készüléket, ami megtévesztő módon Mozilla-ként jelentkezik be)*. Ez azért nem meglepő, mert azon az adott telefonon bizony a Microsoft Mobile Explorer fut. A böngésző típusának *(azaz a Request.ServerVariables("HTTP_USER_AGENT") értékének)* vizsgálata mellett, vagy inkább előtt jobban tesszük, ha annak nézünk utána, hogy a böngésző képes-e kezelni a WML formátumot. Ehhez a HTTP_ACCEPT fejlécek között keressünk például a „vnd.wap.wml"-re *(példa később)*. Ha ezt megtaláltuk, biztosak lehetünk abban, hogy a kérés wapra vonatkozik. Ha nem, még mindig ellenőrizhetjük a HTTP_USER_AGENT string első néhány karakterét, és az alapján dönthetünk, mit adunk válaszként. A detect.asp példakód a fenti két trükk segítségével azonosítja a böngészőt, és emészthető választ ad vissza *(WAP-os kérésre WML-t, hagyományos kérésre pedig HTML-t)*.

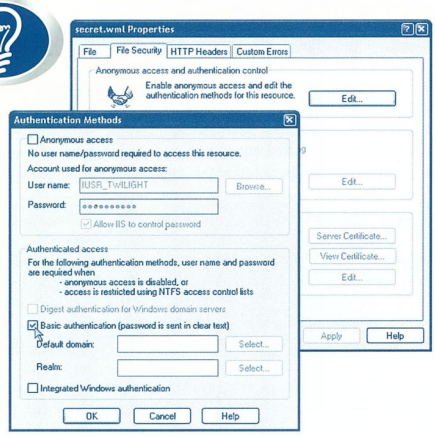
Átirányítás

A HTTP protokoll támogatásának köszönhetően wapon keresztül is működik a böngésző átirányítása, tehát bátran használhatjuk a Response.Redirect() metódust is. Nincs más dolgunk, mint az alapértelmezett ASP oldal elejébe beilleszteni az alábbi kódot – ami felismeri a WML-t kezelni képes böngészőket – plusz egy átirányítást:

```
sAccept =
↳ LCase(Request.ServerVariables("HTTP_ACCEPT"))

If InStr( sAccept, "vnd.wap.wml" ) <> 0 Then
  Response.Redirect "wmlindex.asp"
End If
```

Felhasználóazonosítás

A „névtelen" böngészés mellett a wapos telefonok támogatják a HTTP Basic *(azaz nyílt jelszavas)* felhasználóazonosítást is. Vigyázzunk, mert ez a protokoll nem titkosított, azaz elkapott hálózati forgalomból könnyen visszafejthető a felhasználó neve és jelszava. A titkosított wapos kommunikációt a wap szabványcsalád további elemei *(pl. a WTLS)* definiálják.

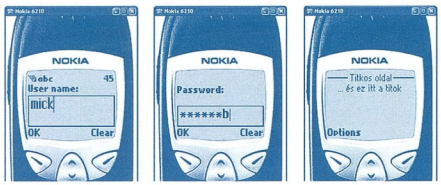


A névtelen wapos böngészés letiltható, a wapos készülékek pedig támogatják a nyíltjelszavas felhasználóazonosítást

A felhasználóazonosítás bekapcsolásának legegyszerűbb módja, ha magán a webkiszolgálón beállítjuk azt. A fenti ábrán egy fájl tulajdonságlapján kapcsoltuk ki a névtelen kérések kiszolgálását, de ugyanezt teljes könyvtárakon is megtehetjük. A másik módszer ASP-ből adódik: rögtön az oldal elején ellenőrizhetjük, hogy az AUTH_USER HTTP változó tartalmazza-e a felhasználónevet. Ha nem, akkor pedig az oldal helyett visszaküldhetjük a szokásos, bejelentkezést előíró hibaüzenetet, valahogy így:

```
If Request.ServerVariables("AUTH_USER") = "" Then
  Response.Status = "401 Unauthorized"
  Response.End
End If
```

Ha a telefon ilyen kéréssel találkozik, a felhasználónevet és jelszót bekéri a felhasználótól:



Bejelentkezés

Egy egyszerű példa: a Service Manager

A cikk végére egy rövid, de velős példát hagytunk: a Windows rendszerszolgáltatásainak távoli indítását és leállítását biztosító kis példaprogramot *(a kód az* **[1]** *címen, az /svc alkönyvtárban található)*. A rendszerszolgáltatásokhoz a WMI-nek köszönhetően férhetünk hozzá. Először is, – ha megfelelő jogosultságokkal rendelkezünk – lekérhetjük az összes létező – vagy épp egy adott nevű – rendszerszolgáltatást jelképező WMI objektumok listáját:

```
Set oServiceSet =
↳ GetObject("winmgmts:{impersonationLevel=
↳ impersonate}").ExecQuery("select * from
↳ Win32_Service Where Name = '" & sName & "'")
```

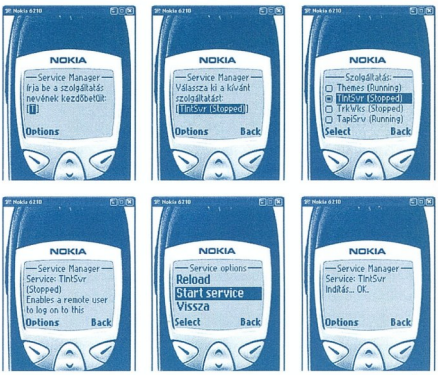
Az eredmény egy enumerálható kollekció, aminek az elemei WMI Win32_Service objektumok:

```
For Each oService In oServiceSet
  Response.Write oService.Name & "<br/>"
  Response.Write oService.State & "<br/>"
  Response.Write oService.Description & "<br/>"
Next
```

A Win32_Service objektum jellemzőiből kiolvasható a szolgáltatás neve, leírása, és pillanatnyi állapota is. Ezeket az adatokat mi is felhasználjuk – és még két fontos metódust:

```
nRet = oService.StopService()
nRet = oService.StartService()
```

A metódusok neve önmagáért beszél. A visszatérési érték 0, ha minden rendben ment, ellenkező esetben a művelet végrehajtása során valamilyen hiba történt.



Service Manager wapon – a szolgáltatások indítása vagy leállítása egyetlen „kattintás"

A példaprogramok további sorai ezeken kívül nem tartalmaznak semmi meglepő, új dolgot.

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://technet.netacademia.net/feladatok/asp/wml/
[2] http://developer.openwave.com
[3] http://www.rcp.co.uk/downloads_wbmp.htm
[4] http://www.forum.nokia.com/
[5] http://webcab.de/wapua.htm

.NET kalauz: tanuljunk .NET-et! De hogyan?

Aki nekilát .NET-et tanulni gyakran úgy érzi magát, mint egy dzsungelharcos: vágja maga előtt a bozótot, küzd nagyon, de valójában fogalma sincs merre tart. Ez a kis útmutató azoknak nyújt segítséget, akik szeretnék élvezni a .NET fejlesztések előnyeit, és minél gyorsabban használható tudásra szeretnének szert tenni.

Ebben a cikkben a Microsoft hivatalos tanfolyami anyagainak *(MOC)* tükrében tekintem át a leendő .NET fejlesztők előtt álló utat. A cég iszonyatos tankönyvválasztékkal lepte meg a fejlesztői társadalmat, mely talán azt sugallja, hogy a .NET megtanulása nem webkattintgatós feladat, hanem elmélyült felkészülést kíván. Ha valaki az önálló tanulás híve, látogasson el a http://msdn.microsoft.com címre. Ha a tanfolyami formát részesíti előnyben, már ma is több oktatóközpont kínálatában megtalálja a .NET-es tanfolyamokat.

A 6-7 kötetnyi, egyenként 4-500 oldalas könyvek láttán az emberben felmerül a csöndes visszavonulás stratégiája, ám ha mégis bátortalanul belekezd a .NET felfedezésébe, többé nem fog visszafordulni. Eldobja korábbi fejlesztési eszközeit és elveit, hogy valami csodálatos új dolog rabja lehessen.

A legfontosabb előzetes tudnivaló a leendő fejlesztőknek, hogy a .NET megtanulásában nem az új nyelvek elsajátítása lesz a legnagyobb falat, hanem a .NET keretrendszer megismerése. A legelemibb műveleteket *(fájlműveletek, bemenetek-kimenetek kezelése, hálózati kapcsolatok kezelése, adatbáziselérés, stb.)* is másképp kell a .NET-ben megvalósítani, hisz kapunk egy elválasztó réteget az operációs rendszer szolgáltatásai és a saját programjaink közé. Adnak egy objektumorientált programozói felületet *(.NET Class Library)*, és szinte mindent ezen keresztül fogunk elérni.

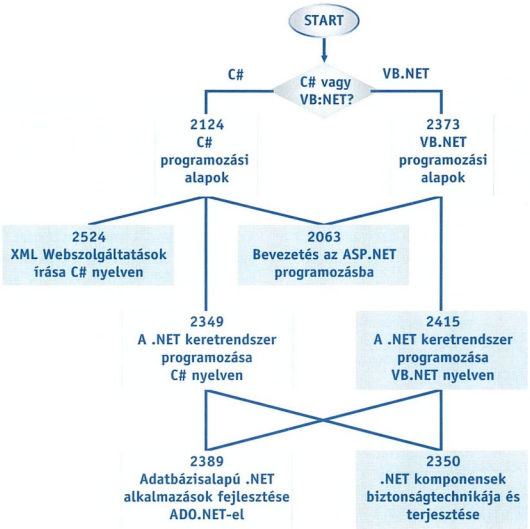
Emellett a .NET-es programoknak együtt kell működni a korábbi komponensekkel is *(DLL-ek és COM komponensek)*, ami további *(nem triviális)* programozási- és rendszerismereteket fog feltételezni. Ezek mellett az új nyelvek egyáltalán nem lesznek nehezek.

Az átlényegülés előrehaladottabb fázisában járó programozók azt mondják, hogy a .NET tanulásában 90%-nyi idő a keretrendszerre, további 10% pedig az új nyelvekre megy el.

A .NET alapjaiban objektumorientált elvekre épült, így nagyon hasznos az objektumorientált alapelvek megfelelő ismerete. A vaskos könyvek objektumorientált elmélettel is szolgálnak. Aki ezeket ismeri, máris 200 oldallal előrébb lapozhat.

Kezdjük az alapokkal: a .NET-es nyelvek

Elméletileg sok, gyakorlatilag kétféle nyelv érdekes a legtöbb .NET programozó számára: a C# *(ejtsd: szí sárp)* és a Visual Basic.NET. Aki szereti a C jellegű kapcsos zárójeles *({})* nyelveket, és szeretné a Microsoft által is favorizált nyelven programozni a .NET-et, annak a C#-ot ajánlom. Nem olyan bonyolult, mint a C++, nincsenek benne a sokak számára érthetetlen pointerek, inkább VB-re emlékeztet vele a programozás, csak C-s formátumban. Aki a JAVA-t ismeri, annak különösen könnyű lesz a nyelv elsajátítása.



C# programozási alapok

(2124 - Introduction to C# Programming for the Microsoft .NET Platform)

Aki szeretné mélységeiben megismerni a nyelvet, és felfrissíteni az objektumorientált programozási ismereteit, annak a 2124-es „Introduction to C# Programming for the Microsoft .NET Platform" tanfolyam ajánlható. Ez egy kőkemény alapozó tanfolyam, amely a „Helló világ" C# nyelvű megvalósításától indul el, a végére azonban nagyon mélyen belemegy a C# nyelv fejlett lehetőségeibe is. Eközben elkalauzolja a hallgatót az objektumorientált elvek C#-beli alkalmazásába, megtanítja néhány Design Pattern alkalmazását, s közben össze-

tett, de jól előkészített és a tanfolyamon tanult elmélettel szinkronban haladó laborgyakorlatokkal segít elsajátítani a bőséges elméleti ismereteket. A tanfolyam nagy előnye, hogy ahelyett, hogy szárazon bemutatná a nyelv elemeit, inkább sok példával és nagyon sok programozási stílust javító tanáccsal látja el a hallgatókat.

A .NET keretrendszer programozása C# nyelven

(2349 - Programming Microsoft .NET Framework with Microsoft Visual C#)

Aki már akár saját maga, akár a 2124-es tanfolyam segítségével túljutott a C#, mint programozási nyelv alapjain, megértette azokat az objektumos és komponensalapú fejlesztési elveket, amelyek a C# magját alkotják, akkor 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 az idejük java részében a keretrendszer szolgáltatásait veszik igénybe egy-egy funkció megvalósításakor. Ez a .NET lelke. Kihasználásához és eléréséhez jelentős segítséget és fejlesztési sebességet ad a Visual Studio.NET, de a keretrendszer ismerete nélkül csak kövérre hízott kódvarázslóként fogyasztja a merevlemezünket.

A kurzus áttekinti a menedzselt futtató környezet működését, és .NET komponensek fejlesztésének kérdéseit. A .NET egyik legfontosabb sarokköve a Common Type System, amely az összes .NET-es nyelv alaptípusait definiálja. Ezzel fontossága miatt több fejezet is foglalkozik. Kiderül a leggyakrabban használt alaptípus, a String sok fontos funkciója, és megismerjük a tömbök és kollekciók fajtáit és használatát.

Az egyik gyakran használt és nagyon sok programozási munkát megtakarító fogalom, a delegate *(függvénymutató)* használata, ehhez kapcsolódik a .NET natív eseménykezelésének megismerése.

Az egyik legtöbb félreértést és jövőbeli problémát okozó témakör, a memóriakezelés és a nem determinisztikus memóriafelszabadítás külön fejezetet kapott. A fájlkezelés és Internetes erőforrások kezelése külön-külön részben van kifejtve, hisz a leggyakrabban használt programozási alapfogásokról van szó. A Serialization, objektumok állapotának futásidejű elmentése és visszaállítása egy igen lényeges alaptechnológia a .NET-ben, emiatt egy komplett fejezetet dedikálunk neki. Ha .NET, akkor a WebSzervizek témája kikerülhetetlen *(nem mintha annyira égető lenne a téma Magyarországon, de előbb-utóbb ide is elér a szele)*, így az utolsó fejezet ezt taglalja. Megismerkedünk a technológia hátterével és programozási fogásaival. Ugyanez a fejezet foglakozik a DCOM utódjával, a .NET Remoting-al, ami az egyik legokosabban megvalósított, és nagyon sűrűn használt szolgáltatása lesz a keretrendszernek.

A programozók az idejük java részében a keretrendszer szolgáltatásait veszik igénybe egy-egy funkció megvalósításakor. Ez a .NET lelke.

Látható, hogy nagyon nagy anyagrészekkel ismertet meg ez a tanfolyam, amelyen elindulva már el lehet kezdeni a fejlesztési és tervezési munkákat. A .NET keretrendszer akkora falat, amit szinte lehetetlen teljes egészében feldolgozni akár tanfolyami keretek között, akár önállóan. Ez a tanfolyam viszont alkalmas olyan alap átadására, amelyen elindulva már képesek a hallgatók a további részletek és újabb témakörök önálló elsajátítására. Mondjuk úgy, hogy megtanít .NET-ül gondolkodni, szemléletet ad a további fejlődéshez.

VB.NET programozási alapok

(2373 - Programming with Microsoft Visual Basic .NET)

A VB.NET kezdő tanfolyam nem annyira a kezdetektől indul, mint C#-os párja, lévén hogy nem egy teljesen új nyelvről van szó. A jelenleg VB6-ban programozóknak segít megtanulni a nyelv új elemeit, illetve átadja a hozzátartozó új szemléletet. Pont a már meglevő alapok miatt a kurzusnak csak kb. a fele foglalkozik magával a VB.NET nyelvvel, a többi rész inkább megmutatja, hogy a VB-ben már ismert eszközöket és technológiákat hogyan kell használni VB.NET-ben illetve a Visual Studio.NET alatt.

A .NET keretrendszer és a Visual Studio.NET alapjainak megismerése után jönnek az új nyelvi elemek: adattípusok, tömbök, függvények és strukturált hibakezelés *(try-catch-finally, halál az On Error-ra :)*.

Az egyik legtöbb félreértést és jövőbeli problémát okozó témakör, a memóriakezelés és a nem determinisztikus memóriafelszabadítás külön fejezetet kapott.

Mivel az objektumorientált elvek megjelenése új a VB.NET-ben, ezért erre a tanfolyam külön hangsúlyt helyez. Az objektumorientált alapfogalmakkal *(egységbezárás, öröklődés, interfészek és polimorfizmus)* példákon keresztül ismerkedünk meg, és megnézzük egy konkrét tervezési példán keresztül, hogy a valós objektumokat hogyan lehet programozott objektumokkal modellezni elméletben, és a Visio segítségével számítógépen is. Ezt követően átültetjük az elméletet a gyakorlatba, és a VB.NET konstrukciói segítségével létrehozzuk az objektumok VB.NET-es implementációját.

A VB6 Form technológia nagyon könnyen használható keretrendszert adott ablakokra épülő alkalmazások írásához. Áttekintjük mit kínál ezen a téren a .NET Windows Forms technológia, amely hasonlóan egyszerűen használható, ám jóval kifinomultabb és átgondoltabb VB-beli öccsénél.

ASP.NET-es webalkalmazásokat bármely .NET-es nyelven lehet írni, ezért megnézzük, hogy hogyan épülnek fel a .NET-es webalkalmazások és Webszervizek. Erről a témáról bővebben a 2063 „Introduction to ASP.NET" értekezik.

Az új adatelérő komponenskészlet, az ADO.NET az egyik leggyakrabban alkalmazott járulékos .NET technológia, így - egy fejezet erejéig - megismerkedünk az alapokkal.

Az újrafelhasználható komponensek témaköre a .NET assemblykel, a saját felhasználói felület elemekkel *(Windows Forms Controls)* és ASP.NET webformokkal foglalkozik.

Legtöbb cégnél az egyik legnagyobb fejtörést okozó probléma az elkészült alkalmazások tömeges telepítése szokott lenni. A megoldáshoz sok segítséget nyújt a .NET alapjaiban átgondolt komponens *(assembly)* azonosítási és verziózási képessége.

Fontos kérdés a VB6->VB.NET upgrade. Mi változott, mi minek felel meg az új nyelvben? Látható, hogy ez a tanfolyam gyakorlatiasabb mint a C#-os társa *(2124)*, viszont nem építi fel olyan mélységig az alapokat.

A .NET keretrendszer programozása VB.NET nyelven

(2415 - Programming the Microsoft .NET Framework with Visual Basic .NET)

Ez a tanfolyam gyakorlatilag ugyanarról szól, mint a .NET keretrendszer programozása C# nyelven tanfolyam *(2349-es)*, csak VB.NET nyelvre kihegyezve. A tartalmát lásd a C#-os társánál.

.NET alapú programfejlesztés

A nyelvek és a keretrendszer ismeretében már megvan a .NET-es programozó írási-olvasási képessége. Azonban a gyakorlati fejlesztés során további problémákba fog botlani, amelyek fejlesztőnként és feladatonként más és más lesz.

Az egyik leggyakoribb és legégetőbb kérdés az adatbázisok használatának kérdése a .NET-es programokból. Akiknek ez a feladata *(a fejlesztők jelentős része)*, azoknak a 2389 - Developing Applications Using ADO.NET for Microsoft SQL Server 2000 *(3 napos)* tanfolyam nyújt segítséget.

Az elkészült alkalmazások telepítése és biztonságtechnikája a .NET egyik legösszetettebb kérdése. Ebben a 2350 - Securing and Deploying Microsoft .NET Assemblies tanfolyam nyújt útmutatást az érdeklődőknek. A vírusok és egyéb ártalmak világában ez a témakör kulcsfontosságú lesz a biztonságos programok írásához.

Miért fog kihalni az ADO? Miért, ki fog halni?

A webalkalmazás-programozók számára az ASP.NET új, igen hatékony eszközöket kínál, melyek kiaknázásához a 2063 - Introduction to Microsoft ASP.NET tanfolyam szolgál kiindulópontul.

Az első két kurzus feltételezi a .NET keretrendszer ismeretét, amelyet a választott nyelven a 2349 - Programming Microsoft .NET Framework with Microsoft Visual C# vagy a 2425 - Programming the Microsoft .NET Framework with Visual Basic .NET tanfolyamok tudnak nyújtani.

Az ASP.NET tanfolyamhoz nem szükséges a keretrendszer átfogó ismerete, viszont valamelyik .NET-es nyelv *(C# vagy VB.NET)* tudása nagyban hozzájárul a tanfolyamon tanultak elsajátításához *(2124 - Introduction to C# Programming for the Microsoft .NET Platform vagy 2373 - Programming with Microsoft Visual Basic .NET)*.

Adatbázisalapú .NET alkalmazások fejlesztése ADO.NET-tel

(2389 - Developing Applications Using ADO.NET for Microsoft SQL Server 2000)

Miért kellett már megint lecserélni az adatelérési komponenseket? Miért fog kihalni az ADO? Miért, ki fog halni? Milyen problémákkal nézett szembe az ADO, ami miatt ki kellett fejleszteni az ADO.NET-et?

Ezen kérdések megválaszolása után rátérünk az ADO.NET-et alkotó osztályok és névterek feltérképezéséhez. Megbeszéljük melyek azok a helyzetek, ahol folyamatos adatbáziskapcsolatra van szükségünk, és mikor használható a sokkal jobban skálázható lecsatolt *(disconnected)* adatkezelési modell.

Megnézzük, melyek a leghatékonyabb módszerek SQL Server adatbázisok, OLE-DB meghajtóval rendelkező adatbázisok és XML adatforrások adatainak elérésére. Áttekintjük a különböző jellegű adatbázislekérdezések leghatékonyabb végrehajtásának lehetőségeit. Megtanuljuk, hogy az ADO beégetett adatmódosító rendszer tárolt eljárásai helyett hogyan írhatunk saját, optimalizált adatmódosító eljárásokat. Kitérünk olyan finomságokra, mint az SQL Server zárolások szabályozása. Megnézzük hogyan kell szakszerűen lekezelni az adatbázismőveletek során fellépő hibákat.

Az ADO.NET egyik nagy újítása a típusos adathalmazok *(Typed Data Sets)* felépítésének lehetősége. Emellett bármikor láthatjuk a relációs adatokat XML-ként, illetve fordítva, XML forrásokkal módosíthatunk relációs adatbázisok tartalmát *(nem csak SQL Server)*. Ez a szoros integráció hatalmas lehetőségeket rejt magában. Megnézzük a DataSet, DataTable és DataView osztályok szerepét és működését. Felsorolásszerűen még néhány témakör a tananyagból: a DataSet mint ügyféloldali cache kihasználása lecsatolt architektúrákban; adatelérés a DataAdapter osztály segítségével; a Visual Studio .NET adatelérő varázslóinak használata tárolt eljárások és típusos adathalmazok generálására. Hogyan hajtsunk végre XLANG *(BizTalk)* ütemezett feladatokat ADO.NET-el?

Miként működnek a lecsatolt adathalmazokon végzett módosítások, és miként lehet a változtatásokat visszamenteni az adatbázisba? Fontos kérdés a módosítási konfliktusok kezelése is. Különösen nagyvállalati környezetben gyakori feladat adatbázison belüli, és adatbázisok közötti vagy teljesen heterogén rendszerek közötti tranzakciók létrehozása. Áttekintjük a lokális és az elosztott tranzakciók ADO.NET-féle végrehajtásának részleteit, és megnézzük, hogy a Message Queue technológia milyen segítséget nyújthat a fokozott és ingadozó teljesítményigények kiegyenlítésére.

Az utolsó részben megvitatjuk, hogy milyen feladatra melyik technológiát érdemes használni a Webszerviz, Webform és Windows alkalmazások közül? Írunk adatelérő Webszervizt, amely szolgáltatásait kihasználjuk webalkalmazásból és Windows Form alkalmazásból is.

.NET komponensek biztonságtechnikája és terjesztése

(2350 - Securing and Deploying Microsoft .NET Assemblies)

Attól, hogy a fejlesztő fogai közül kihúzzák a program „végső" verzióját, a felhasználók gépein még nem fog futni az alkalmazás. A szlogen: .NET-ben a telepítés nem más, mint fájlok másolása. Ez „szinte" így is van. És a konfiguráció, a verziókövetés, a komponens megbízhatóságának ellenőrzése és a rendszer védelme a káros programoktól? Hová irogathat egy .NET-es program *(pl. vírus)*? És a közös komponensek kérdése? A valóság egy „picit" összetettebb a másolásos telepítésnél, ezért ez a tanfolyam.

Megnézzük mi az a DLL pokol, mi idézi elő és mit tehettünk a COM világban a megelőzése érdekében. Áttekintjük, hogy a .NET assemblyk hogyan próbálják gyökerestül kiirtani a probléma okait. Körüljárjuk az egy illetve több fájlból álló assemblyk létrehozásának igényét és kérdéseit. Megnézzük mire való a Reflection nevű technológia, hogyan érhetőek el az assembly metaadatok a segítségével.

Megtekintjük a .NET-es biztonsági házirendek felépítését, hogyan kell egyedi és közös komponenseket telepíteni. Mélyen belemegyünk a verziókövetés kérdésébe, és megvilágítjuk a fokozatos letöltés *(incremental download)* technológia működését és használatát.

Megtanuljuk, hogyan ellenőrzi a .NET futtató egy program által tervezett lépéseket, hogyan lehet digitálisan aláírni az assembly-ket és milyen előnyünk származik ebből.

A .NET saját biztonsági réteget épített az operációs rendszerben meglévő védelem fölé. Ezt akár kódból is kihasználhatjuk *(Code Access Security)*. Az Isolated Storage nevű biztonságos lemezterület felhasználási technológiával megelőzhetők a programok egymásra hatásai, a különböző alkalmazások adatainak keveredése. Áttekintjük a használat módjait. A menedzsel .NET-es kódok és a nemmenedzselt COM vagy natív DLL kódok együttműködése igen érzékeny terület, amely a COM és a .NET együttélés miatt még sokáig aktuális téma lesz, megismerjük a használati lehetőségeket és buktatókat.

.NET-ben a telepítés nem más, mint fájlok másolása.

Bevezetés az ASP.NET programozásába

(2063 - Introduction to Microsoft ASP.NET)

A .NET leghamarabb bizonyított és már Béta állapotban is rengeteg website által is használt része. Tulajdonképpen ez az első, a .NET keretrendszer szolgáltatásaira épülő – Microsoft által fejlesztett – alkalmazás. Az ASP.NET a programozók számára is

hatalmas kényelmi lépést jelent, de emellett *(ami sokszor még fontosabb)* az újratervezett architektúra sokkal nagyobb megbízhatóságot kínál.

A tanfolyam elején megnézzük milyen okok miatt kellett átalakítani az egyébként nagyon sikeres ASP modellt. Megismerjük a két alapvető kiszolgálóoldali vezérlőt: a HTML elemek programozhatóvá tett változatait: az intrinstic vezérlőket; valamint a magasabb funkcionalitású webvezérlőket. Ezek mindegyike olyan paraméterezhető HTML generátor, amely segítségével a VB6-ban megismert eseménykezelőkhöz hasonló megközelítésben és jellemzőkön keresztül tudjuk programozni a webes űrlapokat. Az ASP.NET igyekszik elfedni a webes programozói világ és a formalapú alkalmazások gondolkodásmódjának különbségeit, így egy webprogramozásban nem jártas, de formalapú alkalmazásokban gyakorlott programozó a már számára ismert elvek jelentős részét képes alkalmazni munkájában.

Áttekintjük hogyan lehet deklaratív módon adatkapcsolatokat kijelölni vezérlők között *(nemcsak adatbázishoz, hanem egymás között is)*. A felpostázandó formadatok ellenőrzésére kihasználjuk az adatellenőrző vezérlőket *(validation controls)*.

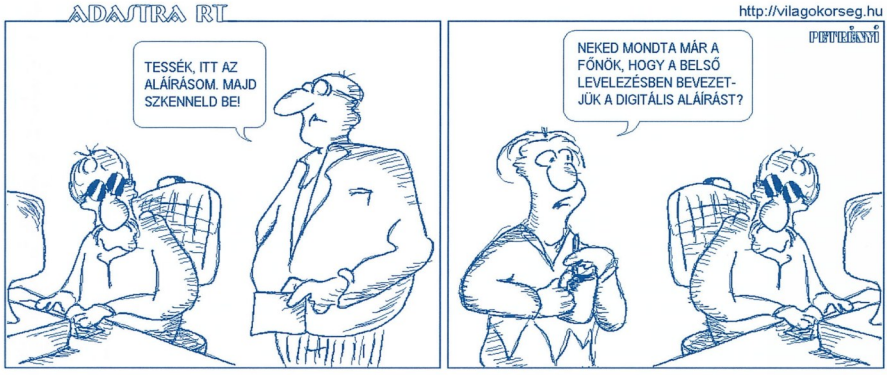
A dinamikus webalkalmazások háttéradatait leggyakrabban adatbázisok tárolják, így azok elérését közelebbről is megnézzük az ADO.NET segítségével. Alkalmazzuk azokat a vezérlőket, amelyek közvetlenül képesek adatbázisadatok *(sorok)* megjelenítésére.

Az ASP.NET lapok mögött nem interpretált, hanem lefordított kódok futnak, ami jelentős sebességnövekedést jelent az ASP-hez képest. Emellett a kódot teljes mértékben elválaszthatjuk a megjelenítést végző vezérlőktől, így könnyebben kézbentartható lapokat kapunk, és szétoszthatjuk a lapformátum kialakítási és a programozási munkáit. Ráadásul a forráskódot lefordíthatjuk bináris komponensekké, így a webszerveren nem kell forráskódot tárolni. A hibakeresés hírhedten problémás pontja volt az ASP alkalmazásoknak, a beépített nyomkövetési funkcionalitás nagy segítség lesz a programozók számára. Az internetes alkalmazások együttműködésének egyik bástyája a Webszerviz technológia lesz. Megtanuljuk a Webszervizek írásának és tesztelésének módszereit. Az alapozás után az utolsó fejezet felvázolja azokat a fejlett ASP.NET lehetőségeket, amelyeket eddig mindig meg kellett írni az ASP lapokban, de most készen kapjuk: cookie nélküli állapotkezelés, a global.asax új eseményei, kimeneti HTML gyorsítótár használata, out-of-process és SQL adatbázisalapú állapotkezelés használata, forms és Microsoft passport alapú felhasználói hitelesítés írása, webalkalmazások biztonsági konfigurálása.

Az ASP.NET lapok mögött nem interpretált, hanem lefordított kódok futnak

...és már túl is vagyunk a .NET alapjain. Jövő ilyenkor remélhetőleg sokan úgy tekintenek majd vissza erre a néhány oldalra, mint a megtett út térképére, és a kávéautomata mellett éppoly ügyesen dobálóznak majd a Reflection és Serialization fogalmakkal, mint manapság az Option Explicittel. Addigra egyesek szerint a C# nyelv is annyira mindennapivá válik, hogy megszűnik tanfolyami oktatása, és áttérhetünk a .NET komolyabb felhasználására: jöhetnek a platformfüggetlen alkalmazások!

Soczó Zsolt
zsolt.soczo@netacademia.net



.NET Akadémia
(II. rész) - .NET alaptípusok

Az előző részben áttekintettük a .NET alapjait madártávlatból, valamint láttuk hogyan kell mezőket, metódusokat és jellemzőket definiálni egy osztályban. Ebben a részben megnézzük az osztályok további jellemzőit, körbejárjuk, hogy miért nagyon fontos az érték és referencia típusok megkülönböztetése. Menet közben érintjük a .NET egyik igen fontos sajátosságát, a nemdeterminisztikus objektum-életciklust is.

Osztályok *(folytatás)*

Jellemzők (propertyk)

Láttuk, hogy a metódusok segítségével írhatunk ellenőrzött mező-beállító és -lekérdező megoldásokat, csak eleg kényelmetlen az obj.Metódus(érték) formátum egy mező értékének beállítására. Mennyivel természetesebb a tiltott gyümölcs, a közvetlen mező-elérés: obj.Mező = érték. Mivel egyszerűbb formátuma miatt a kígyó csábítja a programozókat a közvetlen mezőelérésre, ezért a .NET-be alapjaiban beépítették azt a lehetőséget, hogy a mezők értékét lekérdező és beállító metódusokat külön-külön megírhassuk, de kívülről úgy hivatkozzunk a metódus-párosra, mintha tagváltozók lennének. Ezt nevezzük jellemzőnek vagy angolul property-nek. Nézzük csak! Klasszikus esetben írunk egy SetXXX(), GetXXX() elérő metóduspárt, és ezt használjuk a mezők értékének manipulálására:

```
class JorgomBorger {
  int a = 4;

  //Klasszikus megoldás metódusokkal
  public int GetA() { return a; }
  public void SetA(int a) { this.a = a; }
}
```

Használatuk:

```
JorgomBorger jb = new JorgomBorger();
int x = jb.GetA();
jb.SetA(9);
```

Működik, de elég kényelmetlen.

Jellemzők definiálásához is meg kell írni a set/get párost egy meghatározott formátumban, azonban a hivatkozás sokkal egyszerűbb lesz.

```
class JorgomBorger {
  int a = 4;
  public int A
  {
    get { return a; }
    set
    {
      if (a > 0)
        a = value;
      else
        throw new ArgumentException(
        " A  csak pozitiv lehet!");
    }
  }
}

JorgomBorger jb = new JorgomBorger();
int x = jb.A;
jb.A = 9;
```

A set-ben használt value C# kulcsszó, a jellemzőnek átadott balértéket jelöli *(pl. 9)*. Bármelyik kulcsszó elhagyásával írhatunk csak olvasható vagy csak írható jellemzőt is.

A jellemző beállításakor általában értékhatár-ellenőrzést is végzünk. A példában csak pozitív számokat fogadunk el, más számoknál egy kizárást *(exception)* hajtunk végre, amelyet a hívó metódusnak le kell kezelni. Erre még visszatérünk egy későbbi részben. Amit nagyon fontos látnunk, hogy a jellemzők kényelmes lehetőséget biztosítanak a mezők ellenőrzött hozzáférésére, de a metódusok nehézkesebb hívási formátumát nélkülözve. Emellett a jellemzők mögött sokszor nem valódi tagváltozók állnak, hanem például valamilyen számított érték. Pont az a szerepük, mint a metódusoknak: elfedni a fizikai megvalósítást, az osztály belső szerkezetét.

Konstruktorok, destruktorok

Minden osztálynak van legalább egy speciális metódusa, amit konstruktornak hívunk. Ha mi magunk nem hozunk létre, akkor is keletkezik egy, amit a fordító hoz létre nekünk. A konstruktorok neve megegyezik az őket tartalmazó osztály nevével, nincs visszatérési értékük, de lehetnek paramétereik. Nézzünk bele a Kutyus osztályba:

```
Class Kutyus {
  Kutyus() { ... }
  Kutyus(string szin) { ... }
}
```

Neki mindjárt két konstruktora is van, az egyik nem vár paramétert, a másik igen, a kutya színét string típusban.

A konstruktor egyszer hívódik meg, akkor, amikor létrejön az osztályból egy példány. Feladata az objektum belső kiinduló állapotának, például tagváltozók értékének beállítása induló értékre. Több változat esetén melyik konstruktor hívódik meg? Az a létrehozás módjától függ. A

```
Kutyus k = new Kutyus();
```

hatalmas kényelmi lépést jelent, de emellett *(ami sokszor még fontosabb)* az újratervezett architektúra sokkal nagyobb megbízhatóságot kínál.

A tanfolyam elején megnézzük milyen okok miatt kellett átalakítani az egyébként nagyon sikeres ASP modellt. Megismerjük a két alapvető kiszolgálóoldali vezérlőt: a HTML elemek programozhatóvá tett változatait: az intrinstic vezérlőket; valamint a magasabb funkcionalitású webvezérlőket. Ezek mindegyike olyan paraméterezhető HTML generátor, amely segítségével a VB6-ban megismert eseménykezelőkhöz hasonló megközelítésben és jellemzőkön keresztül tudjuk programozni a webes űrlapokat. Az ASP.NET igyekszik elfedni a webes programozói világ és a formalapú alkalmazások gondolkodásmódjának különbségeit, így egy webprogramozásban nem jártas, de formalapú alkalmazásokban gyakorlott programozó a már számára ismert elvek jelentős részét képes alkalmazni munkájában.

Az ASP.NET lapok mögött nem interpretált, hanem lefordított kódok futnak

Áttekintjük hogyan lehet deklaratív módon adatkapcsolatokat kijelölni vezérlők között *(nemcsak adatbázishoz, hanem egymás között is)*. A felpostázandó formadatok ellenőrzésére kihasználjuk az adatellenőrző vezérlőket *(validation controls)*.

A dinamikus webalkalmazások háttéradatait leggyakrabban adatbázisok tárolják, így azok elérését közelebbről is megnézzük az ADO.NET segítségével. Alkalmazzuk azokat a vezérlőket, amelyek közvetlenül képesek adatbázisadatok *(sorok)* megjelenítésére.

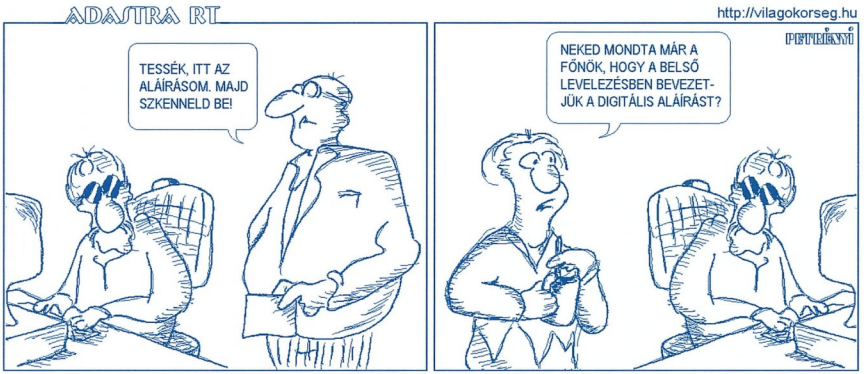
Az ASP.NET lapok mögött nem interpretált, hanem lefordított kódok futnak, ami jelentős sebességnövekedést jelent az ASP-hez képest. Emellett a kódot teljes mértékben elválaszthatjuk a megjelenítést végző vezérlőktől, így könnyebben kézbentartható lapokat kapunk, és szétoszthatjuk a lapformátum kialakítási és a programozási munkáit. Ráadásul a forráskódot lefordíthatjuk bináris komponensekké, így a webszerveren nem kell forráskódot tárolni.

A hibakeresés hírhedten problémás pontja volt az ASP alkalmazásoknak, a beépített nyomkövetési funkcionalitás nagy segítség lesz a programozók számára. Az internetes alkalmazások együttműködésének egyik bástyája a Webszerviz technológia lesz. Megtanuljuk a Webszervizek írásának és tesztelésének módszereit.

Az alapozás után az utolsó fejezet felvázolja azokat a fejlett ASP.NET lehetőségeket, amelyeket eddig mindig meg kellett írni az ASP lapokban, de most készen kapjuk: cookie nélküli állapotkezelés, a global.asax új eseményei, kimeneti HTML gyorsítótár használata, out-of-process és SQL adatbázisalapú állapotkezelés használata, form- és Microsoft passport alapú felhasználói hitelesítés írása, webalkalmazások biztonsági konfigurálása.

...és már túl is vagyunk a .NET alapjain. Jövő ilyenkor remélhetőleg sokan úgy tekintenek majd vissza erre a néhány oldalra, mint a megtett út térképére, és a kávéautomata mellett éppoly ügyesen dobálóznak majd a Reflection és Serialization fogalmakkal, mint manapság az Option Explicittel. Addigra egyesek szerint a C# nyelv is annyira mindennapivá válik, hogy megszűnik tanfolyami oktatása, és áttérhetünk a .NET komolyabb felhasználására: jöhetnek a platformfüggetlen alkalmazások!

Soczó Zsolt
zsolt.soczo@netacademia.net



.NET Akadémia
(II. rész) - .NET alaptípusok

Az előző részben áttekintettük a .NET alapjait madártávlatból, valamint láttuk hogyan kell mezőket, metódusokat és jellemzőket definiálni egy osztályban. Ebben a részben megnézzük az osztályok további jellemzőit, körbejárjuk, hogy miért nagyon fontos az érték és referencia típusok megkülönböztetése. Menet közben érintjük a .NET egyik igen fontos sajátosságát, a nemdeterminisztikus objektum-életciklust is.

Osztályok *(folytatás)*

Jellemzők (propertyk)

Láttuk, hogy a metódusok segítségével írhatunk ellenőrzött mezőbeállító és –lekérdező megoldásokat, csak elég kényelmetlen az obj.Metódus(érték) formátum egy mező értékének beállítására. Mennyivel természetesebb a tiltott gyümölcs, a közvetlen mezőelérés: obj.Mező = érték. Mivel egyszerűbb formátuma miatt a kígyó csábítja a programozókat a közvetlen mezőelérésre, ezért a .NET-be alapjaiban beépítették azt a lehetőséget, hogy a mezők értékét lekérdező és beállító metódusokat külön-külön megírhassuk, de kívülről úgy hivatkozzunk a metódus-párosra, mintha tagváltozók lennének. Ezt nevezzük jellemzőnek vagy angolul property-nek. Nézzük csak! Klasszikus esetben írunk egy SetXXX(), GetXXX() elérő metóduspárt, és ezt használjuk a mezők értékének manipulálására:

```
class JorgomBorger {
  int a = 4;

  //Klasszikus megoldás metódusokkal
  public int GetA() { return a; }
  public void SetA(int a) { this.a = a; }
}
```

Használatuk:

```
JorgomBorger jb = new JorgomBorger();
int x = jb.GetA();
jb.SetA(9);
```

Működik, de elég kényelmetlen.

Jellemzők definiálásához is meg kell írni a set/get párost egy meghatározott formátumban, azonban a hivatkozás sokkal egyszerűbb lesz.

```
class JorgomBorger {
  int a = 4;
  public int A
  {
    get { return a; }
    set
    {
      if (a > 0)
        a = value;
      else
        throw new ArgumentException(
        " A  csak pozitiv lehet!");
    }
  }
}

JorgomBorger jb = new JorgomBorger();
int x = jb.A;
jb.A = 9;
```

A set-ben használt value C# kulcsszó, a jellemzőnek átadott balértéket jelöli *(pl. 9)*. Bármelyik kulcsszó elhagyásával írhatunk csak olvasható vagy csak írható jellemzőt is.

A jellemző beállításakor általában értékhatár-ellenőrzést is végzünk. A példában csak pozitív számokat fogadunk el, más számoknál egy kizárást *(exception)* hajtunk végre, amelyet a hívó metódusnak le kell kezelni. Erre még visszatérünk egy későbbi részben. Amit nagyon fontos látnunk, hogy a jellemzők kényelmes lehetőséget biztosítanak a mezők ellenőrzött hozzáférésére, de a metódusok nehézkesebb hívási formátumát nélkülözve. Emellett a jellemzők mögött sokszor nem valódi tagváltozók állnak, hanem például valamilyen számított érték. Pont az a szerepük, mint a metódusoknak: elfedni a fizikai megvalósítást, az osztály belső szerkezetét.

Konstruktorok, destruktorok

Minden osztálynak van legalább egy speciális metódusa, amit konstruktornak hívunk. Ha mi magunk nem hozunk létre, akkor is keletkezik egy, amit a fordító hoz létre nekünk. A konstruktorok neve megegyezik az őket tartalmazó osztály nevével, nincs visszatérési értékük, de lehetnek paramétereik. Nézzünk bele a Kutyus osztályba:

```
Class Kutyus {
  Kutyus() { ... }
  Kutyus(string szin) { ... }
}
```

Neki mindjárt két konstruktora is van, az egyik nem vár paramétert, a másik igen, a kutya színét string típusban.

A konstruktor egyszer hívódik meg, akkor, amikor létrejön az osztályból egy példány. Feladata az objektum belső kiinduló állapotának, például tagváltozók értékének beállítása induló értékre.

Több változat esetén melyik konstruktor hívódik meg? Az a létrehozás módjától függ. A

```
Kutyus k = new Kutyus();
```

az első konstruktort hívja meg, mert nem adtunk át paramétert az osztálynév után a new-ban. A második verziót akkor tudjuk meghívni, ha átadunk egy string típusú paramétert az objektum létrehozásakor:

```
Kutyus k = new Kutyus("cirmos");
```

Ha vannak konstruktorok, amelyek az objektumok születésekor futnak le, akkor joggal várhatjuk, hogy lesznek destruktorok is, amelyek akkor hívódnak meg, mielőtt az objektum elpusztul. Mikor pusztul el egy objektum? Azokban a nyelvekben és programozási környezetekben, ahol nekünk kell kézzel létrehozni és felszabadítani az objektumokat, van a new operátornak párja, általában a delete operátor, amivel kijelölhetjük mikor szeretnénk felszabadítani egy példány által elfoglalt memóriát. Ebben az esetben egy destruktor végrehajtódásának ideje teljesen meghatározott – meghívom a delete Kiskutya-t, erre megsemmisül a kiskutya objektum, de előtte meghívódik a destruktor.
Ilyen például a klasszikus C++-os környezet. De nem ilyen a .NET! A .NET tervezőinek nehéz döntési kérdésben kellett elhatározni magukat: milyen legyen a memóriakezelés: a programozó által vezérelt, azaz a programozónak kell felszabadítania a lefoglalt objektumokat, vagy a futtatórendszer tegye ezt meg? Igazából az első gyorsan kiesett, mert - tanulva a rengeteg memóriaszivárogtató alkalmazásból - az MS úgy döntött, nem érdemes a fejlesztőkre bízni a memória kezelését.
A második módszer *(amikor a keretrendszer szabadítja fel a memóriát, ha már nem hivatkoznak egy objektumra)* még mindig kétféleképpen valósítható meg: vagy számoljuk, hogy hány hivatkozás mutat egy objektumra, és ha már egy sem, azonnal felszabadítjuk, ez például a COM és a VB6 megközelítése.
Ennek egy alternatívája, ha csak időnként, alkalmanként fut le egy szemétgyűjtő algoritmus, ami megnézi mely memóriabeli objektumokra nem mutat már semmilyen referencia, s azokat felszabadítja. Ebben a kérdésben volt csak igazán nagy meccs az MS-en belül! A két módszer nagyon hasonló, alapvető különbség abban van, hogy mikor semmisül meg egy már nem használt objektum: az első esetben azonnal, ha már nem hivatkozik rá senki *(ld. VB obj = Nothing)*, azaz a megsemmisülés ideje meghatározható azzal, hogy mikor dobjuk el explicit az objektumra mutató referenciát.
A .NET a második stratégiát követi. A nem használt objektumokat felszabadító szemétgyűjtő nem liheg a programunk nyakában, és lesi árgus szemekkel mikor van egy kis takarítanivaló, hanem időnként, leginkább külső kényszer *(fogy a szabad memória)* hatására kezd el működni. Ilyenkor kidobja a már senki által nem hivatkozott, így senki által fel nem használható objektumokat, majd visszavonul.
Mitől más a szemétgyűjtés a programozók szemszögéből mint a hivatkozásszámlálás? Nos, a nagy különbség az, hogy a megsemmisülés időpontja nem meghatározható, **nem determinisztikus**! Azaz **nem** építhetünk a szó klasszikus értelmében vett destruktorok működésére, mert lehet, hogy csak egy óra múlva semmisül meg egy nem használt objektum! A .NET programozás egyik fontos eleme ennek a ténynek a realizálása! Ha például egy objektumban megnyitunk egy állományt, és úgy írjuk meg az osztályunkat, hogy a destruktor fogja lezárni a fájlt, akkor lehet, hogy órákig nyitva marad *(nem töri magát a kukás)*, ami gondokat okozhat. Feleslegesen foglalunk rendszererőforrásokat, vagy éppen egymás működését akadályozhatják a különböző folyamatok. Ez a tény nagyon fontos, úgyhogy ki is emelem:

A .NET Runtime felszabadítja az általunk már nem használt objektumok memóriaterületeit, de nem tudja automatikusan lekezelni az egyéb nem menedzselt erőforrásokat *(ablakkezelők, fájlkezelők, adatbáziskapcsolatok, satöbbi)*. Az a mi feladatunk.

Mit lehet tenni? Ne építsünk a destruktor megszokott, determinisztikus működésére, így minden erőforrást zárjunk le azonnal, amint nincs rá szükségünk. **Ne** rakjunk erőforrás-felszabadító kódot a destruktorba!
A C# a C++-os destruktor formátumot használja:

```
Class Kutyus {
  ~Kutyus() { ... }
}
```

Azaz pont olyan, mint egy paraméter nélküli konstruktor, csak egy kiskigyó *(~)* van a neve előtt.
Az alábbi kódrészletben tanulmányozhatjuk egy Kutyus objektum élettartamát.

```
using System;
class KonstruktorokEsDestruktorok {
  public static void Main() {
    Kutyus k = new Kutyus();
    Console.ReadLine();
    k = null;
    Console.WriteLine("Kutyus kidobva.");
    Console.ReadLine();
  }
}

class Kutyus {
  //Konstruktor
  public Kutyus() {
    Console.WriteLine("Kutyus
↳           konstruktor lefutott");
  }
  //Destruktor
  ~Kutyus() {
    Console.WriteLine("Kutyus
↳            destruktor lefutott");
  }
}
```

A kimenet első része: „Kutyus konstruktor lefutott". Ekkor Entert kell ütni a program folytatásához. Ezután az objektumra mutató referenciát rögtön kinullázzuk, s referenciaszámlálót alkalmazó rendszerben azonnal lefutna a destruktor. De nem a .NET-ben! A nullázás hatására nem történik semmit. A destruktor csak akkor hívódik meg, ha a második Enter hatására a program futása befejeződik.
Vannak nyelvek, amelyek nem támogatják a destruktor közvetlen definiálását. Ez nem probléma, mert valójában a C# destruktor is csalás, az alábbi kódra fordul le:

```
protected override void Finalize() {
    try
    {
        // Ide kerül a destruktor kódja
    }
    finally
    {
        base.Finalize();
    }
}
```

Azaz a destruktor helyett készül egy Finalize() virtuális metódus, ami felüldefiniálja az ősosztályból örökölt metódust. Ennek törzsében lefut a destruktorban kijelölt kódunk, majd meghívódik az alaposztály Finalize() metódusa is, hogy neki is legyen alkalma felszabadítani az általa *(esetlegesen)* lefoglalt erőforrásokat.
Az olyan nyelvekben, amelyekben nincs destruktor, hasonló módon a Finalize() felüldefiniálásával hozhatunk létre takarítókódot, ügyelve az alaposztály Finalize() meghívására is. C#-ban viszont ez nem megengedett, ott a kiskigyó destruktor szintaxist kell használnunk.
A destruktorokat csak akkor használjuk, ha tényleg szükség van rájuk. A szemétgyűjtőnek külön kell még egyet fordulni a destruktorok meghívásához, azaz ezzel pluszmunkát adunk a szemétgyűjtőnek *(és persze a processzornak)*.
Determinisztikus erőforrásfelszabadításhoz általában az `IDisposable` interfész Dispose metódusát valósítjuk meg, majd explicit meghívjuk akkor, ha fel akarjuk szabadítani az erőforrásokat. Ezzel, valamint a szemétgyűjtő működésével és idomításával egy későbbi számban még foglalkozunk.

Érték és referencia szerinti típusok *(Value types és Reference types)*

Az eddig létrehozott osztályaink referenciatípusúak voltak, ami azt jelenti, hogy a new operátorral hoztuk létre őket a szemétgyűjtő által kezelt memóriában. Így például a korábbi példánkban k változó a Kutya osztály egy memóriabeli példányára mutató pointer *(referencia, menedzselt világban nem illik pointerekről beszélni)*, nem pedig a tényleges kutya objektum. Amikor az értékadásokban referenciatípusokat használunk, akkor mindig csak a referencia értéke másolódik át, a valódi objektumhoz nem nyúlunk hozzá.
Ezzel szemben az primitív alaptípusok, például az `Int32` másként működnek. Ők a vermen jönnek létre, így automatikusan felszabadulnak, ha az őket deklaráló metódus lefutott. Nincs szükségük a kukásra. Ami ennél is fontosabb, hogy amikor értékadásban használjuk őket, akkor a tényleges értékük másolódik át, nem csak egy rájuk mutató pointer.
Saját magunk is létrehozhatunk érték szerinti típusokat, egyszerűen a `class` kulcsszó helyett a `struct` kulcsszóval kell létrehozni őket. Automatikusan öröklődnek a `System.ValueType` osztályból, hasonlóan, ahogy a referencia típusú objektumok a `System.Object`-ből.
Miért kellett bevezetni ezt a megkülönböztetést? Gondoljunk csak egy byte-ot ábrázoló típusra! Mennyi a tényleges tárigénye egy ilyen típusnak? 1 byte. És mennyi kell hozzá, ha a memóriában hozzuk létre referencia típusként? Ha egy referencia 4 byte, akkor minimum 4+1 = 5 byte. Ez nem túl biztató. De az igazi probléma nem is annyira a hely, hanem a memórialefoglalás sebessége. Melyik a gyorsabb objektumfoglalási módszer? Letolni a veremmutatót x bájttal, vagy keresni helyet a menedzselt memóriában, adminisztrálni a helyfoglalást, és visszaadni egy mutatót? Nyilvánvalóan az első megoldás lényegesen gyorsabb, ráadásul a megsemmisítés automatikus lesz.
Az érték szerinti típusok hátrányai akkor kezdenek megmutatkozni, amikor nagyméretű típust akarunk létrehozni. Egy bizonyos határ fölött a nagyméretű, értéktípusú objektumok másolása több időt igényelhet, mint referenciatípusként a lefoglalása jelentett volna. Emiatt az érték szerinti típusok leginkább kisméretű objektumok tárolására alkalmasak *(durván 16 byte a határ)*. Viszont kis adatmennyiségek használatára sokkal gyorsabban tudnak lenni, mint a referencia típusok. Például tekintsünk meg egy komplex számokat tároló típust.

```
public struct Complex {
   double real, imaginary;
}
```

Vagy egy pixel koordinátáit leíró típus:

```
public struct Point {
   int x, y;
}
```

Az összes adatelrejtési elv, amit az osztályoknál elmondtam, érvényes az érték szerinti típusokra is, azaz használjunk jellemzőket és metódusokat a belső részletek elfedésére.
Hasonlóan az osztályokhoz, nekik is lehetnek konstruktoraik, de csak paraméterezettek:

```
public Point(int x, int y) {
   this.x = x; this.y = y;
}
```

Éppúgy létre kell belőlük hozni egy példányt, mint az osztályokból, a new operátorral:

```
Point p = new Point();
```

Ha létrehozáskor azonnal inicializálni akarjuk a struktúrát, akkor például használhatjuk az előbbi paraméterezett konstruktort:

```
Point p = new Point(1,4);
```

A legtöbb alaptípus használatakor elég kényelmetlen lenne a new használata, ezért élhetünk egy rövidített formátummal is:

```
Point r;  //Létrehozás
r.x = 10; r.y = 6;  //Használat
```

Vagy például egy 32 bites egész szám létrehozása egyenértékű megoldások:

```
int i = new int();
int j = new System.Int32();
int k;
```

A new nélküli esetekben a tagváltozók automatikusan kinullázódnak. Ami első ránézésre nagyon szokatlan, hogy az érték szerinti típusok is implementálhatnak interfészeket. Például a legtöbb alaptípus implementálja az `IComparable` interfészt, amire többek között a keretrendszer által nyújtott tömbrendezések automatikusan építenek, hogy össze tudjanak hasonlítani két példányt az adott típusból.

Ilyen a BOX!

Többször használtuk már a Console osztály WriteLine metódusát egész számok kiíratására is, például:

```
int j = 10;
Console.WriteLine("{0}", j);
```

Pedig a `WriteLine` ezen változata egy `System.Object` típusú referenciát vár második paraméterül, nem egy integert. Akkor miért nem jelez hibát a fordító? Csak nem lehet az, hogy egy érték szerinti típust, mint az int el lehet érni egy `Object` típusú referencián keresztül?

De bizony! De hogyan lehet egy veremben lefoglalt érték szerinti típust pointeren keresztül elérni? Csak nem a verebe mutat ilyenkor a referencia? Természetesen nem. Amikor egy érték szerinti típust kell referencián keresztül elérni, akkor a fordító olyan kódot generál, amiben a típusnak megfelelő memóriát lefoglalja a szemétgyűjtő által menedzselt memóriában, és átmásolja a típus tartalmát a veremből. Ezt hívjuk Boxing-nak, hisz bedobozolja a változó tartalmát egy másik memóriahelyre.
Emiatt olyan furcsa kódokat lehet írni, mint:

```
int j = 10;
object o = j; //box
```

Miért jó ez? Ha minden – és tényleg minden – típust elérhetünk `object` típusú referencián keresztül, akkor ez azt jelenti, hogy például egy kollekcióban bármilyen típust tárolhatunk, hisz elég csak az `object` referenciákra megírni a szükséges elérőmetódusokat, a nem object típusok automatikusan boxolódnak `object`-té.
Mi a helyzet a visszaúttal, hogyan lehet kicsomagolni egy bedobozolt érték szerinti típust? Visszafelé már nem teljesen sima az ügy, mert egy általános `object` referencia által hivatkozott memóriaterületről kell visszamásolni az értéket a veremre. Ha például beboxolunk egy 16 bites egészet a memóriába, majd azt vissza akarnánk nyerni mint egy 32 bites számot, akkor abból nagy baj lehetne, ha túlolvasna az értéket másoló eljárás az első két bájton. Ezért az egyértelmű szándék kijelölésére visszirányban egy kasztoló operátorral jeleznünk kell, hogy mivé kell visszaalakítani a referenciát:

```
//Hibás: Cannot implicitly convert
//type 'object' to 'int'
int k = o;
//Rendben van.
int m = (int)o;
```

A kényelmes átjárás miatt könnyedén lehet írni általános tömböket vagy metódusparamétereket, de nyilvánvalóan ára van az általánosításnak: a boxolás lassítja a programok futását. Például egy specializált kollekció, ami csak egy adott érték szerinti típust tud tárolni, de azt közvetlenül, típusos elérő metódusokon keresztül *(boxolás nélkül)*, jelentősen gyorsabb lehet az általános `object` referenciát használó társánál. A .NET keretrendszer tartalmaz néhány általános kollekciót, és pár specializáltat is. Sajnos azonban igazi, boxolás nélküli, primitív típusokat kezelő típusos kollekciója nincs. Az **[1]** címről letölthető egy erősen típusos kollekció megvalósítás *(coll.cs)*, amiben integereket tárolhatunk a keretrendszer `ArrayList` nevű kollekció osztályhoz hasonlóan, azaz egy változtatható hosszúságú tömbként. Az osztály egy sor átírásával könnyedén átalakítható más típusú érték szerinti típus tárolására, de sajnos ez azt jelenti, hogy minden típushoz le kell másolni és át kell írni a teljes forráskódot. Hej, ha lennének template-ek, mint a C++-ban...

A boxing teljesítménycsökkentő hatásának demonstrálására írtam egy tesztelő alkalmazást *(box.cs)*. Ebben paraméterezhető módon ki lehet próbálni az általános `ArrayList` és a saját típusos kollekció használatát nagyszámú 32 bites egész tárolására és visszaolvasására. A tesztelő programban megfigyelhető a nagyfelbontású *(<1 microsec)* rendszeróra használata, amihez a `QueryPerformanceCounter` és a `QueryPerformanceFrequency` API függvények köré írtam egy egyszerűen használható csomagoló osztályt.

Egy mérés nem mérés, ezért paraméterezhető számban megismételhetjük a tömböket terhelő hívásokat. Az így keletkező átlagok elég jól reprodukálható eredményeket adnak. Nyilván egy multitaszkos operációs rendszeren lehetetlen kizárni a többi program ráhatását, de azért néhány bemelegítő futtatás után kielégítően stabil eredményeket kapunk. Nagyon fontos, hogy legyen elég szabad RAM a gépben, hogy ne okozzunk lapozást a tömbök helyfoglalásakor.
Az alábbi táblázatban összefoglaltam néhány mérés eredményét:

Elemek száma	Mérések száma	Végrehajtási idő Array-List-tel [ms]	Végrehajtási idő specializált kollekcióval [ms]	Idő-nyereség szorzó
100.000	10	196,64	63,23	3,10
100.000	10	206,27	57,08	3,61
100.000	10	196,70	57,45	3,42
300.000	10	1142,08	215,17	5,30

Az első három sorból látszik, hogy a mérés jól reprodukálható, nincs 10%-nál nagyobb szórás a mért értékekben. Láthatóan a boxolást kiküszöbölő specializált kollekció többszörösen gyorsabb az általános típusnál. A forrásokat letöltve **[1]** otthon bárki megismételheti a méréseket a saját gépén, illetve ötleteket kaphat egy specializált kollekció megírására. Ha egy alkalmazásban nagyon fontos nagy elemszámú alaptípus hatékony tárolása, akkor érdemes elgondolkodni egy saját verzión.
Mindezek ellenére nem szeretném, ha azt éreznék, hogy a boxing egy ártalmas és teljesen felesleges technológia. A két típus megkülönböztetése nagyon jelentős teljesítménynyereséget eredményez, ha rövid típusok tárolására és paraméterek átadására érték szerinti típusokat használunk. Egy másik, J-vel kezdődő technológiában nincs ez a megkülönböztetés, csak referenciatípusok vannak, ami miatt egyes teljesítménycentrikus helyeken kénytelenek nagyobb számú primitív típust átadni paraméterül egy egyszerű struktúra helyett, a referencia típusok okozta teljesítményproblémák miatt...

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

A cikkben szereplő URL-ek:
[1]: Letölthető példakódok
http://technet.netacademia.net/download/net

XMLgessünk

Átjárás a relációs és az xml világ között

Az xml nagyszerű eszköz adatok átvitelére, heterogén rendszerek közötti információcserére, de csapnivaló adattárolásra. Redundáns és csak lassan lehet belőle kihámozni a tényleges információtartalmat. Hatékony adattárolásra már régóta léteznek megfelelő eszközök, az adatbázisok. A legtöbb adatbázis azonban relációs elvekre épült, téglalap alakú információhalmazokban gondolkodik, amelyek között kapcsolatok vannak definiálva. Hogyan valósítható meg az átjárás két világ között, a lehető leghatékonyabb módon? Erre keresem a választ a .NET eszközeivel.

A kihívás

Hogyan lehet relációs adatbázisban tárolt táblák adatait xml-lé transzformálni, és visszafelé, hogyan lehet xml-ben kapott információkat relációs adatbázis táblákban eltárolni? Leggyakrabban különböző rendszerek közötti kommunikációban kell megvalósítani a konverziót, amelyben például két cég interneten keresztül, xml formátumban kommunikál egymással, de minkét oldalon relációs adatbázisban tárolják a küldendő vagy fogadandó adatokat.
Másik lehetséges felhasználás az xml könnyű transzformálhatóságát szeretné kihasználni *(XSLT-vel)*, azonban ehhez elérhetővé kellene tenni a relációs adatokat xml formátumban. Nézzük meg, milyen lehetőségeink vannak!
Olyan adatbáziskezelő esetén, amely közvetlenül képes xml kimenetet generálni és xml formátumú adatokat feldolgozni, nincs különösebb gond az átjárással. Ilyen például az SQL Server 2000, erről már írtam az újság hasábjain tavaly ősszel. Ebben az esetben a konverziós eljárásokat megírták helyettünk, nekünk csak definiálni kell az adatbázistáblák és az xml dokumentum közötti leképezést. Erre a legkézenfekvőbb eszköz az XML séma, amely szinte minden szükséges konstrukciót tartalmaz az átjáráshoz. XML sémával definiálni tudjuk az xml oldal szerkezetét, az adatbázis szerkezete adott, azt az adatbázistáblák létrehozásakor konstruáltuk meg. Ami nem része az xml sémának, az maga a leképezés a táblák és az xml tagok között. Ezzel az információval viszont könnyedén kiegészíthető a séma, hiszen külön névtérben bármilyen kiegészítő xml tartalommal megtűzdelhetünk *(annotálhatunk)* egy sémát, attól az eredeti működése nem változik meg. Viszont egy olyan alkalmazás – például az SQL Server OLE-DB meghajtó – ami értelmezi a kiegészítéseket, képes forrás xml adatokat SQL parancsokká leképezni, valamint tud olyan SQL parancsot legyártani, amely a sémának megfelelő xml kimenetet generál. Többek között erről szól az SQL Server xml támogatása. Ha nem ilyen rózsás a helyzet, azaz a támogatandó adatbázisok nem rendelkeznek beépített xml támogatással, akkor szükség lesz egy olyan keretrendszerre, amely képes a híd szerepének betöltésére. Jöjjön az ADO.NET!

ADO.NET alapvetés

Mit keres egy adatbáziselérő technológia az XML rovatban? A .NET-ben megszűnt az éles határvonal az hierarchikus XML, és a sík relációs világ között, az ADO.NET adatbáziskezelő osztályai és az előző két részben látott XML osztályok jó barátságban élnek, és remekül együttműködnek egymással. A kapcsolat megértéséhez induljunk el a relációs világ irányából.
Az ADO.NET részletes bemutatatása nem célunk, ezzel majd a .NET Akadémia sorozatban egy külön számot fogunk eltölteni. Most csak a legszükségesebb ismereteket mutatjuk be. Az adatelérő osztályok a System.Data névtérben vannak. SQL Server elérésére a specializált Sql... kezdetű osztályokat használjuk, általános OLE-DB meghajtóval rendelkező adatokhoz az OleDb... kezdetű osztályok valók. A példákban az egyszerűség kedvéért az Sql osztályokat használom, de nem használom ki az SQL Server beépített xml lehetőségeit.
Adatbáziskapcsolat kiépítésére az SqlConnection osztály használatos, nagyon hasonlóan az ADO-beli Connection objektumhoz.

```
//Adatbáziskapcsolat kijelölése (itt még nem
//kapcsolódik az adatbázishoz)
SqlConnection c = new SqlConnection("Initial
↳ Catalog=Northwind;Data Source=(local);
↳ Integrated Security=SSPI;");
```

A lekérdezéseinket SqlDataAdapter objektumon keresztül hajtjuk végre:

```
//DataAdapter felépítése
SqlDataAdapter a = new SqlDataAdapter(
↳         "SELECT * FROM Products", c);
```

A lekérdezés eredményhalmazát DataSet objektumban tároljuk. A DataSet az univerzális adattároló eszközünk, amely leginkább memóriabeli adatbázisként fogható fel.

```
DataSet ds = new DataSet();
```

Az eddig még üres DataSet-ünket a DataAdapter segítségével töltjük fel:

```
//A parancs végrehajtása. Az eredményhalmaz
//a Termek táblában jelenik meg.
a.Fill(ds, "Termek");
```

A DataAdapter megnyitja az adatbáziskapcsolatot, végrehajtja a konstruktorban kijelölt lekérdezést, és lezárja a kapcsolatot. A

DataSet-ünkben létrejön egy Termek nevű tábla *(DataTable objektum)*, amely a lekérdezés eredményét tárolja. Az így nyert tábla illetve DataSet teljen független a forrásadatbázistól, attól el van választva.

Egy DataSet tetszőleges számú táblát tárolhat, a táblák között relációkat definiálhatunk pont úgy, mint egy valódi adatbázisban. Emellett kulcsokat, megszorításokat is létrehozhatunk a oszlopokon, azaz valóban úgy viselkedik, mint egy igazi adatbázis. A lekérdezés kimenetét tároló tábla szerkezetét a lekérdezés oszlopai szabják meg. Ezt könnyedén megtudhatjuk a WriteXmlSchema metódus meghívásával:

```
StreamWriter w = new StreamWriter("s1.xml");
ds.WriteXmlSchema(w);
```

Tekintsünk bele a kimenetbe *(s1.xml)*:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="NewDataSet"
xmlns:xs=http://www.w3.org/2001/XMLSchema
xmlns:msdata="urn:schemas-microsoft-com:
xml-msdata">
 <xs:element name="NewDataSet">
  <xs:complexType>
   <xs:choice max0ccurs="unbounded">
    <xs:element name="Termek">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ProductID"
        type="xs:int" min0ccurs="0" />
       <xs:element name="ProductName"
        type="xs:string" min0ccurs="0" />
       <xs:element name="Discontinued"
        type="xs:boolean" min0ccurs="0" />
       ...
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Ez egy közönséges XML séma *(XSD)* fájl. Olyan xml dokumentumokat ír le, aminek a gyökéreleme a NewDataSet nevű elem, ebben tetszőleges számú Termek elem helyezkedik el. Ezek az adatbázisból lekérdezett sorokat tárolják, láthatóan az oszlopok azonos nevű gyermekelemekre képződtek le.

Örülünk ennek a szép sémának, de nekünk a tényleges adatokat tartalmazó xml kimenet a fontos. Mi sem egyszerűbb:

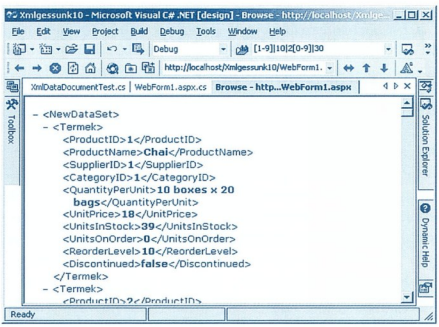
```
ds.WriteXml("termekek1.xml");
```

Kukkantsunk bele a generált dokumentumba:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Termek>
    <ProductID>1</ProductID>
    <ProductName>Chai</ProductName>
    ...
    <Discontinued>false</Discontinued>
  </Termek>
  <Termek>
    <ProductID>2</ProductID>
    ...
  </Termek>
</NewDataSet>
```

A WriteXml() természetesen nemcsak fájlba képes írni, hanem tetszőleges kimeneti adatfolyamba is. Például egy xml kimenetet generáló webform *(aspx, .NET-es ASP)* esetén közvetlenül írhatunk a kimeneti adatfolyamba:

```
Response.ContentType = "text/xml";
ds.WriteXml(Response.OutputStream);
```



DataSet xml kimenete böngészőben

Az xml kimenet testreszabása

Láttuk, hogy a DataSet feltöltése után kiolvashatjuk a betöltött – pontosabban a generálandó xml – adatok szerkezetét a WriteXmlSchema()-n keresztül. Nem lehetne ezt visszafelé is megtenni? Egy sémával definiálni az xml kimenet szerkezetét, rátölteni a relációs adatokat, majd kinyerni az általunk igényelt xml kimenetet? Természetesen lehet!

Tegyük fel, hogy a ProductID-t, a CategoryID-t és a SupplierID-t szeretnénk kiemelni a megfelelő Termek elemek attribútumaként. Ehhez így alakítjuk át a sémát:

```
...
<xs:element name="Termek">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ProductName" ... />
      ...
    </xs:sequence>
    <xs:attribute name="ProductID"
     type="xs:int" use="required"/>
    <xs:attribute name="CategoryID"
     type="xs:int" use="optional"/>
    <xs:attribute name="SupplierID"
     type="xs:int" use="optional"/>
  </xs:complexType>
</xs:element>
...
```

Mielőtt feltöltenénk a DataSet-et a DataAdapter-rel, adjuk meg neki az általunk kialakított sémát:

```
ds.ReadXmlSchema("s2.xml");
a.Fill(ds, "Termek");
ds.WriteXml("termekek2.xml");
```

Íme a hőn áhított xml kimenet, az ID-kkel az attribútumokban *(termekek2.xml)*:

```
<NewDataSet>
  <Termek ProductID="1" CategoryID="1"
↳ SupplierID="1">
    <ProductName>Chai</ProductName>
    ...
```

Típusos DataSet-ek

Ha a DataSet képes xml séma alapján saját formátumú xml kimenetet generálni, akkor miért nem lehet szorosabban összeházasítani a DataSet struktúráját meghatározó XSD sémát a DataSet-el? Azaz készíthetnénk egy specializált DataSet osztályt, ami már eleve tartalmazza az oszlopok saját, típusos definícióját. Ez nem csak az xml kimenet egyszerűbb generálásában segítene, de az oszlopokra erősen típusos módon tudnánk hivatkozni, így a relációs adatok közvetlen elérésekor is már a kívánt típusban érhetjük el az oszlopokat, így már a fordító ellenőrizni tudja az adatelérés helyes típusát.

Nézzünk erre egy példát! Az előbb létrehozott általános DataSet esetén a Termek táblát a DataSet Tables kollekcióján keresztül érhetjük el, majd végigmehetünk az összes sor minden oszlopán:

```
DataTable t = ds.Tables["Termek"];
int numCols = t.Columns.Count;
for(int row = 0; row < t.Rows.Count; row++) {
  Console.WriteLine("Sor: {0} ", row);
  DataRow r = t.Rows[row];
  for(int col = 0; col < numCols; col++) {
    Console.WriteLine("{0}, {1}: {2} ",
          t.Columns[col].ColumnName,
          t.Columns[col].DataType, r[col]);
  }
}
```

Kimenet:

```
Sor: 1
ProductID, System.Int32: 2
CategoryID, System.Int32: 1
SupplierID, System.Int32: 1
ProductName, System.String: Chang
...
Discontinued, System.Boolean: False
...
```

Látható, hogy az oszlopoknak van adattípusa, ami a megfelelő háttérsémából képződik futásidőben. A probléma oka pont ez. Ha például a Discontinued oszlop tartalmát típusosan meg akarjuk ragadni, akkor azt kasztolnunk kell bool típussá:

```
bool disc = (bool)r["Discontinued"];
```

Ha 'neadjisten rosszul tudjuk az adattípust, vagy például megváltozott az adatbázis sémája, akkor erre csak futás közben, a konverzió tényleges végrehajtásakor derül fény:

```
int disc = (int)r["Discontinued"];
```

```
Unhandled Exception:
System.InvalidCastException:
Specified cast is not valid.
```

Ha a DataSet oszlopainak típusa nem futásidőben, hanem már fordításkor ismertek lennének, akkor már a fordító észrevenné a helytelen konverziót. Irány a típusos DataSet!

A típusos DataSet egy általános DataSet osztályból származik örökléssel, így annak összes tulajdonságát magán viseli, ám az oszlopai adattípusát előre definiáljuk, és minden oszlophoz készítünk egy típusos adatelérő jellemzőt. Egy ilyen specializált DataSet osztály megírása igencsak mechanikus babramunka lenne, ezért jelentős segítséget kapunk hozzá.

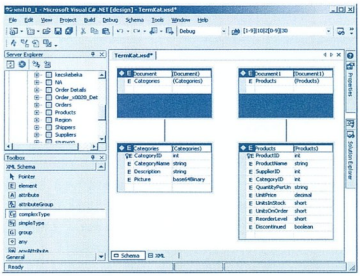
A DataSet szerkezetét egyszerűen leírjuk egy XSD sémával, majd az xsd.exe segítségével legeneráltathatunk egy specializált, típusos DataSet osztály forráskódot, amelynek pontosan olyan lesz a szerkezete, mint amilyet a séma definiál. Azaz ebben az esetben nem futásidőben töltjük be a sémát, hanem még szerkesztési időben „dolgozzuk bele" a DataSet-be.

Kiindulásként szükség van egy xml sémára. Ezt megírhatjuk kézzel is, lévén, hogy egy közönséges xml fájlról van szó. Azonban elérkezett a pont, hogy bevessük a Visual Studio.NET-et is!

Egy tetszőleges projektet létrehozva *(én Console alkalmazást használtam)* a Project/Add New Item menüpont mögötti párbeszédablakban keressük ki az XML Schema típust, és adjunk neki valami nevet. Ekkor létrejön egy üres xml séma fájl *(.xsd)*, amit tervezési nézetben láthatunk. A kívánt struktúrát kialakíthatjuk a Toolbox-on található alapösszetevőkből: elemek, attribútumok, csoportok, komplextípusok, satöbbi. Ez a módszer akkor jöhet jól, ha nem érhető el a forrásadatbázis, amely kimeneteihez írunk DataSet-et.

Mostani példánk egy összetettebb DataSet lesz, melynek sémája kétféle táblát fog tartalmazni: Categories és Products. Ők a Northwind adatbázis termékeket és termékkategóriákat tároló táblái. A két tábla között ráadásul szülő-gyermek kapcsolat is van, egy termékkategóriához több termék is tartozhat. A sémában ezt is definiálhatjuk, így majd a DataSet tábláinak feltöltésekor már működni fog az idegenkulcs megszorítás, így eleve nem generáltathatunk bele inkonzisztens adatokat.

A két tábla szerkezetét kézzel is összerakhatjuk, ha ismerjük az adatbázistáblák struktúráját. Ennél azonban jóval egyszerűbb, ha a Server Explorer ablakban felveszünk egy adatbáziskapcsolatot, és ráhúzzuk a táblákat a tervezőre.



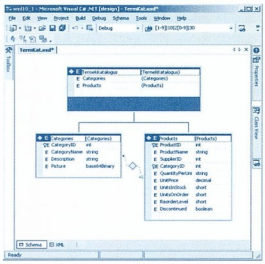
Már benn vannak a forrástáblák, csak még nem a megfelelő kapcsolatban

Alakul a dolog, csak éppen nincs szükségünk a két Document elemre, hanem azt szeretnénk, ha a Categories és a Products egy gyökérelem közvetlen gyermekelemei lennének. Bizonyára ez elérhető a tervezővel is, azonban hamarabb célhoz érünk, ha az xml forrás-

nézetben egyszerűen kézzel átmozgatjuk a Products elemet *(minden belső tartalmával együtt)* a Categories elem mögé. Rövidítve így fog kinézni a séma:

```
<xs:element name="Document">
<xs:complexType>
  <xs:choice max0ccurs="unbounded">
    <xs:element name="Categories">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CategoryID"
          msdata:ReadOnly="true"
          msdata:AutoIncrement="true"
          type="xs:int" />
          <xs:element name="CategoryName"
         type="xs:string" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Products">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ProductID"
         type="xs:int" />
          <xs:element name="ProductName"
         type="xs:string" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

A Document elemet átnevezhetjük Termék katalógusra. Sajnos a tervező önmagában inkább csak egy kiinduló váz elkészítésére alkalmas, a hatékony munkához mindenképpen elengedhetetlen az XSD ismerete, mert időnként bele kell nyúlni az xml forrásba. Már csak a két tábla közötti kapcsolat definiálása van hátra, ebben jó a tervező. Mintha csak egy adatbázistervező programot használnánk, a Products elem CategoryID nevű elemét húzzuk rá a Categories elem CategoryID elemére. Az előugró dialógusablakban ügyeljünk arra, hogy jól jelöljük ki melyik a szülőtábla *(Categories)* és melyik a gyermek *(Products)*.
Ezzel készen is van a sémánk!



Készen van a két táblát tartalmazó xml séma

A két tábla közötti kapcsolat leírására XSD key és keyref elemeket generál a szerkesztő, amelyek kijelölik a CategoryID-n keresztüli kapcsolódást. Ami számomra furcsa *(bizonyára én vagyok az ügyetlen, nem a Visual Studio)*, hogy a Products tábla CategoryID elemére létrehozott egy XSD keyt, mintha az egy elsődleges kulcs lenne. Erről persze szó sincs, ezért azt ki is töröltem. A két alaptábla ID elemeire generált kulcsokat átneveztem, és a kapcsolatot leíró keyrefet is ennek megfelelően módosítottam. Hiába na, a kézi munka gyönyöre. Előállt az alábbi kapcsolatot leíró sémarészlet:

```
<xs:unique name="CategoryKey"
    msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:Categories" />
  <xs:field xpath="mstns:CategoryID" />
</xs:unique>
<xs:unique name="ProductKey"
    msdata:PrimaryKey="true">
  <xs:selector xpath=".//mstns:Products" />
  <xs:field xpath="mstns:ProductID" />
</xs:unique>
<xs:keyref name="CategoriesProducts"
    refer="CategoryKey">
  <xs:selector xpath=".//mstns:Products" />
  <xs:field xpath="mstns:CategoryID" />
</xs:keyref>
```

Hogyan lesz az xml sémából típusos DataSet? Az xsd.exe segítségével a következő módon:

```
xsd.exe /d TermKat.xsd
```

Az xsd.exe generál egy TermKat.cs C# nyelvű forráskódot, amelyben egy hamisítatlan, erősen típusos DataSet-ből származó osztály kódja található. Ugyanezt megtehetjük a Visual Studio-ban is, a Schema menü Generate DataSet paranccsal *(a háttérben ott is az xsd.exe hívódik meg)*. A VS nagy előnye, hogy az xsd módosításakor a háttér DataSet osztály azonnal újragenerálódik.
A generált DataSet-et le kell fordítanunk a használathoz. Hogyan lehet feltölteni a két forrástáblával a DataSet-ünket? Egyszerűen létrehozunk két DataAdaptert a megfelelő lekérdezésekkel, és mindkettővel feltöltetjük ugyanazt a saját típusos DataSet-et *(TermekKatalogus)*.

```
//DataAdapter felépítése a termékekhez
SqlDataAdapter aProd = new SqlDataAdapter(
↳               "SELECT * FROM Products", c);
//DataAdapter felépítése a kategóriákhoz
SqlDataAdapter aCat = new SqlDataAdapter(
↳               "SELECT * FROM Categories", c);

TermekKatalogus ds = new TermekKatalogus();
//Fontos a feltöltési sorrend, már működik az
//idegenkulcs!
aCat.Fill(ds);
aProd.Fill(ds);
```

folytatjuk...

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:
[1]: A cikkben szereplő példakódok
http://technet.netacademia.net/download/xml

Zárolás, livelock, deadlock

Az SQL Server zárairól lapunk 2001 márciusi számában már olvashattak Soczó Zsolt tollából *(28. oldal, Transact SQL VI. rész)*. Végigvette a zárak típusait, a tranzakcióizolációs szinteket, majd a cikk végén könnyelműen megígérte, hogy legközelebb a deadlock következik. Még egy év sem telt el, s íme a folytatás.

Mivel azonban mégiscsak eltelt 11 hónap, bátorkodom saját szavaimmal ismét feleleveníteni a zárolást, mert anélkül nincs halálos ölelés!

I. rész, elmélet: zárak

A komoly, adatbázismotoros adatkezelő rendszerek mindegyike rendelkezik tranzakciókezelő, és a párhuzamos tranzakciók adatmódosításainak elkülönítésére szolgáló beépített rendszerrel. A megoldási módszerek tekintetében közel sem egységesek a piac versengő termékei - az SQL Server például zárolást használ a módosítás alatt álló rekordok védelmére.

Ha megpróbálunk áthatolni e védelmen, ellenállásba ütközünk: az SQL Server megvéd minket attól a szerencsétlenségtől, hogy nem végleges, félkész adatokat olvassunk. Ilyenkor várnunk kell a zár felszabadulásáig. Vannak esetek, amikor ez a várakozás reménytelen.

Ha megpróbáljuk józan paraszti ésszel megállapítani, hogy mi legyen a zárolás alapegysége, arra juthatunk, hogy elegendő csupán az éppen módosítás alatt álló rekord megváltozó mezőjét hozzáférhetetlenné tenni. Mert hiszen minek is lefoglalni egy teljes rekord összes mezőjét *(név, cím stb.)*, ha a felhasználó most csak a személyi számot és a születési évet módosítja?

Ez a gondolatmenet helyes ugyan, de iszonyatosan „drága". Az adatbáziskezelő minden egyes zárolásért memóriával *(meg kell jegyeznie, hogy mit fogunk módosítani)* és processzoridővel *(a zárolást nem hardveresen oldjuk meg ugyebár)* fizet. Természetesen minél több a lock, annál több erőforrást kell e rendszer fenntartására fordítani. A magas erőforrásigény miatt az ős SQL Serverben *(6.5-ig bezárólag)* a legkisebb zárolható egység a lap *(page)* volt. Az akkori 2 kb-os lapokra átlagosan 50-100 rekord fért, így egyetlen sor módosítása miatt kényszerűen még további 49-99 sort fogtunk meg. Volt is rengeteg olyan alkalmazás, mely csak úgy dobálta magából a deadlockokat, és ütemesen hajigálta vissza a tranzakciókat a megdöbbent felhasználónak. Az SQL Server felhasználói és fejlesztői új megoldást követeltek Redmondtól.

A 7.0-s változatban a lapméretet 8 kb-ra növelték, és ha a zárolási rendszert változatlanul hagyták volna, s mind a mai napig egész lapokat kellene lefogni, most bizony 200-400 rekordot zárolnánk feleslegesen egyetlen sor módosítása miatt. A hetes változat óta szerencsére a legkisebb zárolható egység a rekord - de ne higyjük, hogy ez minden alkalmazásfejlesztési és tervezési hibát megold! Deadlock mindig lesz!

A zárak típusai

„Szaporodjatok, és sokasodjatok!"

A zárak típusai hatókör, vagy „méret" szerint

Vegyünk egy adatmódosító tranzakciót:

```
UPDATE vevok SET cegnev='NetAcademia'
```

Ha ezt ráeresztjük a partneradatbázisunkra, meghökkentő eredményt kapunk: az összes vevőnk neve NetAcademia-ra változik. Kellett volna egy WHERE feltétel, de lemaradt. Most vonatkoztassunk el e módosítás adatbázisgyilkoló eredményétől, és koncentráljunk kizárólag arra, mi is történik a végrehajtás során. Végigfutunk a táblán, és módosításra kerül az összes vevő cégneve. Ha minden egyes rekordot külön-külön zárolna az SQL Server, akkor egy-egy 8k-s lapon többszáz egyedi fogd-meg-ereszd-el történne, amelynek végrehajtási ideje vetekedne az érdemi munka, a módosítás idejével. Az SQL Server azonban résen van: ha egy lapon túl sok egyedi rekordot kellene zárolni, akkor visszatér a jó öreg page lockhoz, és egy akkorddal lefogja a lap összes sorát, módosít, majd elengedi a lapot. Ez az eljárás többször gyorsabb lehet, mintha minden egyes soron külön-külön vacakolna a lakattal. A folyamatot lock escalationnak hívják, és némiképp paraméterezhető az sp_configure tárolt eljárás segítségével *(százszámokat és százalékokat lehet állítgatni, de ne babráljuk)*.

A fenti utasításból és gondolatmenetből kiolvasható, hogy az SQL Server automatikusan teszi fel és veszi le a megfelelő zárakat a tranzakciók futtatása során. Optimizer Hintekkel a működés befolyásolható ugyan, de általános, és sokat ismételt aranyszabályként jegyezzük meg: Optimizer Hintet alkalmazásokba bevarrni **TILOS**!

Ha egy tranzakció netántalán egy teljes táblát érint, akkor - az eszkaláció következő lépéseként - az SQL Server a sok-sok egyedi page lockot lecserél*(het)*i táblaszintű zárolásra, mert ezzel megintcsak értékes erőforrások szabadulnak fel. Ezernyi page lock helyett egyetlen table lock - erőforrásspórolás a köbön.

A zárak típusai a felhasználás módja szerint

Ha a zárolási rendszerről beszélünk, óhatatlanul felmerül a gondolat: mely műveletek során van szükség zárak használatára? Az UPDATE, INSERT és DELETE műveleteknél egyértelmű a válasz: adatmódosítás történik, tehát kell a lakat. Ilyenkor az SQL Server úgynevezett exclusive lockot, kizárólagos felhasználásra jogosító zárat tesz az érintett sorra/lapra/táblára *(az eszkaláció szerint)*, s azt a tranzakció végéig fenn is tartja - más még csak nem is olvashatja a megbűvölés alatt álló adatokat.

De SELECT esetén kell-e zárolni? Ilyenkor adatmódosítás nyilvánvalóan nem történik - minek ide zár? Ilyen esetben a zárolás célja, hogy lehetetlenné tegye azon adatok módosítását, melyeket éppen olvasunk. Az SQL Serverben alapértelmezésben minden olvasás konzisztens adatokat, lezárt tranzakciók utáni állapotot ad vissza. Erre azért képes, mert minden olvasott sorra/lapra/táblára *(az eszkaláció szerint)* úgynevezett shared lockot, megosztott zárolást helyez. Azért megosztott, mert egy így zárolt objektum-

ra tetszőleges számú további olvasó csimpaszkodhat, és olvashatja párhuzamosan a többiekkel.

A végeredmény: egy adatot egyszerre tetszőleges számú folyamat olvashat, de csak egyetlenegy írhatja.

Ha felütjük a Books Online megfelelő fejezetét, rábukkanhatunk a zárak kompatibilitási táblázatára. Ez megmutatja, hogy ha egy rekordon ilyen-olyan zár van, milyen további zárak pakolhatók rá akadálytalanul.

		Új zár		
		shared	update	exclusive
Jelenlegi zár	shared	*	*	
	update		*	*
	exclusive			

Csillag jelöli a táblázatban azokat a zárkombinációkat, ahol egy meglévő x zárra rá lehet tenni egy y zárat. Ebből kiolvasható, hogy a shared és az exclusive lock nem kompatibilis egymással, nem pakolható egyik a másikra. Egy módosító tranzakciónak ki kell várnia, hogy az összes olvasó eltakarodjék az útból. Azonban gátat kell tudnunk szabni az „olvasóközönségnek", mert egy éppen olvasott adatra a fenti táblázat szerint shared lockot akárki, akármikor ki tud adni, a módosításra váró tranzakció pedig csak várna, hogy tiszta legyen a terep, de hiába, mert az új olvasók csak jönnek és jönnek. Az update lock feladata az „álljon meg a menet!" jelzése az olvasó folyamatok felé. Ez ugyanis rátehető a shared lockokra, de őrá már sharedet pakolni nem lehet. Olyan, mint az UNO kártyában a kérőlap: mostantól csak adott kártyalapot lehet tenni, mégpedig exclusive lock feliratút. Akinek shared van a kezében, az mostantól nem teheti le a lapját. Az update lock nevével ellentétben még nem jelent adatmódosítást, csak a szándék jelzésére való. Amikor ugyanis „kijátsszuk" ezt a lapot, még bőven van a halomban alatta shared lock, melyek idővel eltakarodnak – új azonban már nem lesz! Így az update lock jelenléte egy soron/lapon/táblán bizony **várakozást** jelent: arra várunk, hogy végre mindenki eltűnjön, és átállhassunk exclusive zárolásra, s a módosítás valóban bekövetkezhessen.

A livelock és az új vasunk

A zárolási folyamat eddigi ismertetéséből kitűnik, hogy ha a szerencse, vagy a rossz tervezés úgy hozza, az adatmódosítások bizony jelentős késlekedést szenvedhetnek, amitől az alkalmazás maga is lelassul, és – Magyarországon vagyunk – a vezérigazgató egy idő múlva engedélyt ad új, nagyobb vas beszerzésére. *(Még véletlenül sem gyanakszik a fejlesztőre.)* Hisz nyilvánvaló, hogy a régi pécé nem bírja a terhelést! A sors különös fintorának tűnik ezek után, hogy az új vas sem bírja. Egyértelmű: rossz az SQL Server, kapitalista-globalista-monopolista a Microsoft és hülye a Bill Gates.

Meg tudom érteni a gondolatmenetüket: x raktáralkalmazás fél éve úgy futott, mint a kisangyal, de azóta belevittünk még ezer *(!)* tételt, felvettünk négy új alkalmazottat, és jól látható módon ez kifektette a régi, Pentium III 450-es kiszolgálót. Bárhogyan is logikusnak tűnik az okfejtés, én az ilyen okoskodás hallatán sírógörcsöt kapok: ezer rekord és négy júzer kiüt egy SQL Servert? Tessenek belegondolni: amikor 486 DX4-100 volt a csúcspécé, SQL Server 6.0-val többezer felhasználós, gigabájtos adatbázisrendszerek készültek!

Axióma

Magyarországon nincs az az adatbázisméret, ami megfektetne egy SQL Servert, és nincs az a kereskedelmi forgalomban kapható PC, ami ne lenne elég erős bármilyen feladatra!

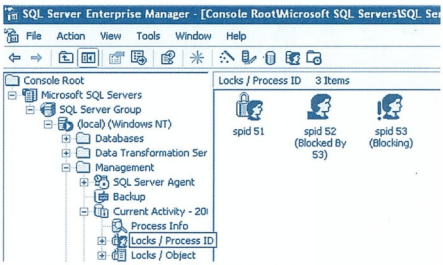
Ez talán egy kicsit sarkított véleménynek tűnik, de van ennél durvább is:

128 MB RAM, 10 GB HDD, PIII 800 mindenre elég

(Bill Gates után szabadon :-)

Gigabájtok? Ugyan már! Kétszáz felhasználó? Hol, melyik vállalatnál van akár csak feleennyi SQL user?

Az SQL Server teljesítményproblémák 175%-ban hibás tervezésre vezethetők vissza.



Az Enterprise Managerben jól látható, hogy melyik folyamat vár, és melyik okozza a várakozást (felkiáltójellel jelölve)

A legváltozatosabb fejlesztői baklövések miatt lassulnak be az alkalmazások. Vegyünk néhány, életből ellesett példát:

- Az ügyféloldali alkalmazás fejlesztője hadilábon áll az SQL nyelvvel, ezért mindenhova a legegyszerűbb lekérdezést írja az ADO kommandokba: SELECT * FROM TABLA. El se hinnék, hogy ez milyen gyakori eset. Hány sor, és hány oszlop megy át a munkaállomásokra? Ugye mind? Milyen index gyorsítja ezt a lekérdezést? Semmilyen. Kár indexelni. Az SQL Server a teljes tábla lekérdezése miatt table lockra eszkalál, nem fog pepecselni egyedi sorszintű zárolással. Ahogy növekszik a tábla, úgy válik egyre hosszabbá az az idő, amíg ezen a táblán táblaszintű shared lock van. A módosításra váró tranzakciók holtideje egyenletesen növekszik, merthogy indexhasználat híján table Scan „stratégiával" csoszog végig az SQL Server a táblán. Kétszerannyi rekord? Kétszerakkora idő! Háromszorannyi adat? Háromszorannyi idő!
- Eseményvezérelt ügyfélalkalmazásunkban ráakaszkodunk az egyik tetszetős eventre *(mondjuk: OnMouseMove)*, és itt elhelyezünk egy csinos kis SELECT-et. Hányszor fog lefutni? X-nél többször, az már egyszer biztos! shared lock, shared lock, shared lock.
- Tranzakció kellős közepén megkérdezzük a felhasználót egy párbeszédpanelben, hogy komolyan gondolta-e. Kattintson az igenre, ha igen. A felhasználó azonban a kávéautomatát támasztja: exclusive lock a végtelenségig. Eközben nem lehet eladni, mert a mocskos SQL Server nem mutatja meg a pultnál, hogy hány darab van a vevő által kért fittingből.
- További furfangos lassító módszerek léteznek az indexek megspórolásától kezdve a tempdb_in_ram őrült opció beállításáig.

A zárak típusai a felhasználás szándéka szerint

Az alcím az eredeti angol kifejezés *(intent lock)* fordítása, de némiképp félrevezető. Az Intent lockok ugyanis segédfunkciót töltenek be az SQL Serverben. Vegyünk egy példát. Néhány tranzakció éppen módosít egy-egy sort, emiatt sorszintű exclusive lockokat használnak. Egy másik alkalmazás egy SELECT * utasítással lekérdezi a teljes táblát, tehát táblaszintű shared lockot

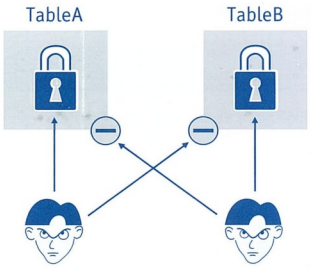
használna. Hogyan dönthető el minimális idő alatt, hogy az egész táblára rátehető-e a shared lock? A helyzet a következő: a fiúk a bányában *(sorszinten)* dolgoznak, de nekünk a bánya bejáratánál állva el kell tudnunk dönteni, hogy van-e valaki odalenn, mert el szeretnénk árasztani vízzel. Ha óvatlanok vagyunk, megfulladhatnak a vájárok. Ha lemegyünk körülnézni, végig kell járnunk az összes járatot, hogy meggyőződhessünk róla: senki sem dolgozik. Idő, értékes idő! A bejárós stratégia további hátránya, hogy miközben odalenn mászkálunk, és hahózunk, újabb vájárok szállhatnak be a liftbe, és jöhetnek le. Így sohasem végzünk! Ehelyett inkább hozzunk olyan szabályt, hogy aki lemegy, az egy zászlócskát tűz a bejárathoz, amit kifelé jövet eltávolít onnan. Így egyetlen pillantással eldönthető, hogy mehet-e a víz. A zászlócska neve: intent lock. Különböző típusai vannak annak megfelelően, hogy a leszálló személy csak nézelődni megy *(munkavezető, shared lock)*, vagy dolgozni is fog *(vájár, exclusive lock)*. A bányászos példánál annyival bonyolultabb a valóság, hogy jelzőzászlót nemcsak a bejárathoz *(táblaszinten)* kell tenni, hanem lapszinten is, hogy a 8 kilobájtos lapok zárolásáról is dönteni lehessen az egyes sorok átvizsgálása nélkül.

Tranzakciók és izolációs szintek

Az izolációs szintek arról szólnak, hogy egy alkalmazás mennyire szeretné, hogy az általa használt adatok a tranzakciók végéig változatlanok maradjanak. Minél „változatlanabb" adatra van szükség, annál erősebben kell markolni mindent, amit használunk. Alapértelmezésben *(Read committed)* a tranzakciók során kiadott exclusive lockok megmaradnak, de az olvasott rekordok shared lockjai menet közben elpárolognak, hogy amit mi csak olvasunk, azt más szabadon módosíthassa. Repeatable read esetén már a shared lockok is kitartanak a tranzakció végéig: bármi, amit olvasunk, az garantáltan változatlan marad tranzakciónk végéig. Serializable izolációs szinten még az indexfa elemei is zárolásra kerülnek, hogy nehogy hasonló rekordok bukkanjanak fel, mialatt mi dolgozunk.
Ez ügyben mégiscsak a tavaly márciusi szám elolvasását javaslom, a 31. oldaltól két teljes oldalon át az izolációs szintek elemzését találjuk. Térjünk rá a halálra.

II. rész, gyakorlat: a halálos ölelés

Amikor két *(vagy több)* tranzakció úgy kerül várakozási állapotba, hogy kölcsönösen egymás erőforrásaira várnak, a konfliktus feloldhatatlanná válik: egyik sem engedi, ami az övé, ezzel nem hagyja a másikat sem továbbfutni. Ezt a zárolási esetet az SQL Server szerencsére éppúgy észreveszi, mint a fent mutatott live lockot, és pontot tesz az ügy végére: az egyik tranzakciót visszagörgeti *(rollback)*.
Ha kettőnél több folyamat csimpaszkodik egymást akadályozó módon egymásba, csodálatos deadlock fürtök alakulhatnak ki. Ilyen esetben rendkívül nehéz utólag megállapítani, hogy hová implementáltuk azt a tervezési hibát, ami miatt megáll egy tucat tranzakció. Célszerű már előre olyan adatbázistevet és alkalmazást készíteni, mely minimálisra csökkenti a deadlockok kialakulásának esélyét. Van egy-két ökölszabály, melyekre érdemes figyelni, ezek közül a legfontosabb, hogy tranzakcióink mindig ugyanabban a sorrendben nyúljanak hozzá a táblákhoz, így eleve nem alakulhat ki olyan helyzet, mint az alábbi ábrán, ahol az egyik folyamat az A táblát fogja, és B elérésére vár, míg a másik folyamat B-t zárolja, és A-ra vár.



Két tranzakció egymás zárolt erőforrására vár. Az egyiknek el kell pusztulnia, hogy a másik tovább élhessen!

Hamarosan látni fogjuk, hogy a fenti ökölszabály nem igazán jelent megoldást, mert bizony egyetlen kétsoros táblán is létrejöhet deadlock – csak „megfelelő" eljárás kell hozzá!

Melyiket vonjuk vissza?

Fontos kérdés, hogy - ha már egyszer megtörtént a baj – vajon az összeakaszkodott tranzakciók közül melyikre mondjunk halálos ítéletet. Ezt a folyamatok fontossága alapján maga az SQL Server eldönteni nem tudja, hisz nem érti, mit veszítünk egy-egy adatmódosítás visszavonásával. Alapértelmezésben az SQL Server-féle költség dönt: amelyik „olcsóbban" visszavonható, az fog meghalni. Még pontosabban: amelyik kevesebb tranzakciónapló bejegyzést tett, az lesz az áldozat.
Ezt a működést némiképp szabályozhatjuk a

```
SET DEADLOCK_PRIORITY { LOW | NORMAL}
```

utasítás segítségével, bár a parancs szintaxisából kiderül, hogy pont az hiányzik, ami kellene: a magas túlélési prioritás beállítása!

Felkészülés a halálra

Most néhány sorban készítünk egy tesztadatbázist, melyben szabadon lehet majd összeakasztani felhasználókat egymással. Jelentkezzünk be az SQL Serverre egy rendszeradmin erősségű felhasználóval, majd:

```
create database semmi
```

Ez a parancs egy „semmi" nevű adatbázist készít. *(Default méret, fájl stb. stb.)*. Most átállunk erre az újdonsült adatbázisra:

```
use semmi
```

Kell egy tesztfelhasználó:

```
sp_addlogin 'senki'
```

... akit beengedünk a „semmi" adatbázisba:

```
sp_adduser 'senki'
```

Ezután létrehozunk egy nagyon egyszerű, két mezőből álló táblát:

```
create table vevok (
  id int,
  name varchar(22)
)
```

...melybe beszúrunk két sornyi adatot:

```
insert vevok values(1,'NetAcademia')
insert vevok values(2,'iNetCom')
```

Jogot adunk „senki"-nek a vevok tábla használatához:

```
grant select, update on vevok to senki
```

Már majdnem készen vagyunk. Indexeljük le a „name" mezőt, mert azt gyakran használjuk keresésekben:

```
create index nameindex on vevok(name)
```

Mostantól el kell szakadnunk a kéthasábos formától, mert két párhuzamosan futó tranzakció lépéseit fogom ábrázolni egymás mellett. A két folyamat betűről betűre ugyanaz: ugyanannak az ügyfélprogramnak két példánya akad össze egymással. Kérem, senki ne akadjon fenn a kód ostobaságán. Az élet szülte!

	1. tranzakció lépése	2. tranzakció lépése	Megjegyzés
idő ->	begin tran	begin tran	Mindkét folyamat elindul
	update vevok set name='Eniac Hungary Bt.' where id=1		Az első folyamat exclusive lockot helyez az első sorra. Simán lefut. A lock ottmarad a tranzakció végéig!
		update vevok set name='Eniac Hungary Bt.' where id=2	A második folyamat exclusive lockot helyez a második sorra. Akadálytalanul lefut, lock marad!
	select * from vevok		Az első folyamat kíváncsi a végeredményre, ezért leválogatja a teljes táblát. Leválogat-ná. Ha nem lenne zárolva a tábal második sora. Livelock, földgömb forog.
		select * from vevok	Nesze neked, mégegy select. Természetesen ez is elakad, és reménytelenül vár az első folyamat végére, mert az viszont ránk vár. Deadlock.
	Server: Msg 1205, Level 13, State 50, Line 1 Transaction (Process ID 52) was deadlocked on {lock} resources with another process and has been chosen as the deadlock victim. Rerun the transaction.		

A két folyamat közül az egyik visszavonódik *(rollback)*. A fenti esetben mindkettő ugyanannyit dolgozott, így véletlenszerűen dől el, melyik haljon meg. Az egyik mindenesetre áldozatául *(victim)* esik az SQL Server kemény akaratának. Alkalmazásainkban mindig figyeljünk a 1205-ös hibakódra. Ha ez felbukkan, törődjünk vele, mert bizony a tranzakciónk nem futott le!

A deadlock okainak felderítése

Talán az egyik legnehezebb adatbázistuningolási feladat a halálos ölelések kivédése, mert sok esetben meghatározhatatlan időközönként jön egy-egy. Ilyenkor természetesen senki nem áll készenlétben, így elkapni sem lehet. Olyan megoldásra van tehát szükségünk, amely hosszú időn át figyeli a deadlockokat, és naplót ír, ha megint beesett egy. Ezt a feladatot az SQL Server maga végzi. Ha az SQLSERVR.EXE-t nem szolgáltatásként *(service)*, hanem parancssorból indítjuk, egyrész látni fogjuk egy „DOS" ablakban, hogy mit művel, másrészt különböző parancssori kapcsolókkal módosíthatjuk a futását. A -T kapcsolóval nyomkövetési *(trace)* üzemmódba kapcsolhatjuk. Indítsuk így:
SQLSERVR -T 1204
Ekkor az SQL Server minden deadlock információt kiír a képernyőre, beleértve az összeakaszkodó folyamatok azonosítóit és a vetekedő SQL parancsokat is!
Jó nyomozást!

Fóti Marcell
marcellf@netacademia.net
MCDBA

MsDownload – itt a végleges .NET Framework!

2001 október közepétől a Netacademia Kft üzemelteti Magyarország egyik legnagyobb letöltőközpontját. A Microsoft Magyarország megbízásából elkészült oldalak az eddig megszokott tartalommal, de eltérő formában jelentkeznek. Az innen letölthető programok közül csemegézünk ebben a rovatban.

.NET Framework Software Development Kit

Sokan vagyunk, akik már hallottunk a .NET térhódításáról. Próbálgattuk az új technológiát, ismerkedtünk előnyeivel és hátrányaival. Nyomonkövethettük, hogy válnak egyre kidolgozottabbá, teljesebbé a névterek, az elérhető objektumok.

Január közepén végre annak örülhettünk, hogy az újonnan kiadott verzió már nem egy újabb béta, hanem az első teljes verziót tarthattuk kezünkben.

Ebben a cikkben a végleges verzióval foglalkozunk. Megtudhatjuk milyen rendszerkövetelményei vannak a .Net Frameworknek, milyen eszközök érhetők el a telepítés után, és milyen változásokon esett át a Beta2 óta.

A .NET Framework SDK a következő platformokra telepíthető:

- Windows 2000 a legutolsó javítócsomaggal
- Windows XP
- Windows NT 4.0. Ha ez a platform áll a rendelkezésére, akkor rendelkezni kell a Windows NT 4.0 SP6a javítócsomaggal is.

Szoftverek közül pedig az alábbiakra van szükség:

- Internet Explorer 5.01 vagy ennél frissebb verzió
- Internet Information Server
- a Microsoft Data Access Component 2.6 követelmény, de ajánlott az MDAC 2.7.

A .NET Framework részét képezi az ASP.NET platform, mely olyan webalkalmazások készítését teszi lehetővé, amelyek a Common Language Runtime *(CLR)* minden előnyét, például a biztonságos típushasználatot, az öröklődést, a verziózást kihasználják.

Ebben a rendszerben már elkülöníthető a design és a programozás. Külön fájlokban tárolhatók az oldalt kiszolgáló programkódok, melyek megírásához többféle nyelvet is választhatunk *(Visual Basic .NET, C#, Jscript .NET)*.

Az alkalmazások fejlesztéséhez használhatunk különböző HTML szerkesztőket, vagy akár Notepad-ot is, de a Microsoft Visual Studio .NET programcsomagot is az összes eszközével.

Természetesen ez utóbbi nem ingyenes, viszont tartalmazza .NET Framework-ot, valamint minden olyan eszközt, amire a fejlesztéshez szükségünk lehet.

Ám azok se keseredjenek el, akik az ingyenesen letölthető .NET Framework SDK-t választják és nem rendelkeznek Microsoft Visual Studio .NET-tel. Amire szükségük lehet, így is megtalálják. Lehet, hogy így kicsit tovább tart a fejlesztés, viszont beleláthatnak a .NET lelkivilágába, és olyan részletek is feltárulhatnak előttünk, amelyeket a Visual Studio varázslói egyébként elrejtenének.

Jó tudni, ha a .NET Framework SDK telepítése előtt nincs telepítve rendszerünkre IIS, akkor az ASP.NET alkalmazások – nyilván - nem futnak. Ha utólag történik meg az IIS telepítése, akkor kézzel kell regisztrálnunk az Aspnet_isapi.dll fájlt a Regsvr32.exe segítségével. Valamint minden egyes rendszerfrissítés után a .NET Framework-öt javítanunk kell. Ezt megtehetjük a Control Panel / Add or Remove Programs / Microsoft .NET Framework SDK –nál a Click here for support information linkre kattintva. Ez a hivatkozás egy repair.htm fájlra mutat a telepített .NET Framework megfelő verziókönyvtárában. Ha ez nem található, akkor a n:\\setup.exe /t:%temp% /c:"msiexec.exe /fvecms %temp%\netfxsdk.msi" utasítást kell beírnunk egy parancssorba.

Ez után a kis kitérő után nézzünk néhány olyan eszközt, amely a rendelkezésünkre áll a .NET Framework telepítése után.

Assembly Cache Viewer *(Shfusion.dll)*: A global assembly cache-t lehet vele nézegetni, illetve szerkeszteni.

Assembly Registration Tool *(Regasm.exe)*: egy assembly-n belüli metaadatokat olvassa ki és elvégzi a szükséges bejegyzéseket a registry-ben, amely lehetővé teszi, hogy COM kliensek .Net Framework osztályokat használjanak.

Assembly Binding Log Viewer *(Fuslogvw.exe)*: a hibás assembly-kötések okát mutatja meg. E segítségével megtudhatjuk, miért nem talál meg egy assemblyt futás közben a .NET Framework.

.NET Framework Configuration Tool *(Mscorcfg.msc)*: egy Microsoft Management Console-t nyit, melyen keresztül menedzselhetjük a Global Assambly Cache-ben lévő assemblyket, a kódhozzáférési védelmi beállításokat, a remote service-ket.

Microsoft CLR Debugger *(DbgCLR.exe)*: grafikus felülettel rendelkező *(vissza)*fordító, amely segít a fejlesztéskor fellépő hibák megtalálásában és javításában.

Runtime Debugger *(Cordbg.exe)*: parancssori hibakereső, amely segít a hibák azonosításában és javításában.

(A fenti két fordító valamelyikét vagy a Systems.Diagnostics osztály függvényeit kell használnunk akkor, ha nem rendelkezünk a Visual Studio .Net-tel.)

A Code Access Security Policy Tool *(Caspol.exe)*: segítségével meghatározhatjuk és módosíthatjuk a kódhozzáférések biztonsági beállításait a gép, felhasználó illetve vállalat szintjén.

Ezeken kívül persze nagyon sok hasznos dolgot találhatunk még. Végül az egyik újdonság a .NET Framework-ben a Beta2-höz képest: míg az előző verzióban az ASP.NET System-ként futott, addig a végleges verzióban már saját account-tal *(ASPNET)* rendelkezik. Így szabályozhatjuk, hogy milyen jogosultságokkal rendelkezzen .Net webünk.

Borsi Katalin
bobo@netacademia.net

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

Dupla KV

RAID, Windows XP

K1: *A merevlemezek elérési sebessége jelentősen lassítja a számítógépek összteljesítményét. A teljesítmény növelhető a Windowsba épített RAID eszközökkel. Mi lehet az oka, hogy néha mégsem érjük el a kívánt gyorsulást?*

K2: *2 db 40 GB-os merevlemezt szerettem volna össze stripe-olni (magyarul csíkkészletet létrehozni) és érdekes dolog történt. Mindkettő FAT32-re volt formázva, egyenként jól működtek, de a szoftveres stripe nem volt sikeres, a hibaüzenet 'Volume size too big' volt. Kikísérleteztem, hogy FAT32 fájlrendszerrel a maximális méret partíciónként 16384 MB lehet. Ennél nagyobb partíciókon „természetesen" NTFS fájlrendszernek kell lennie. Ennek nem igazán látom okát, mivel a FAT32 fájlrendszer Windows 9x alatt ennél lényegesen nagyobb területet tud egy darabban kezelni. Nem lenne egyébként semmi bajom az NTFS-sel, csupán annyi, hogy nagyon lassú, a két merevlemezt meg éppen azért akartam össze stripe-olni, hogy jó gyors elérést kapjak. Így, hogy NTFS van rajtuk, még együtt sincsenek olyan fürgék, mint egyenként FAT32-vel.*

V1: a FAT32 és a méret problémája. A válasz a Windows 2000-ben keresendő. A Windows 2000 maximálisan 32 GB méretre tud formázni FAT32 partíciót *(az előre formázott nagyobb méretőt kezeli)*, így volt lehetséges, hogy az összekötött *(stripe-ba, más néven RAID 0-ba szervezett)* merevlemezek egyenként 16 GB-ig, azaz összesen 32 GB-ig működtek. Ennek csupán az az oka, hogy a Microsoft szeretné leszoktatni a Windows 2000 felhasználóit a FAT, FAT32 használatáról.

V2: az NTFS és a stripe sebességéből eredő probléma. Nézzük meg mit takar a stripe fogalma! A stripe vagy RAID 0 a merevlemezek paramétereinek javítása érdekében kidolgozott műszaki megoldás, a RAID legalacsonyabb szintje. A RAID *(Redundant Array of Independent Disks)* alkalmazásával az adatelérési sebesség, az adattárolók összefüggő mérete, valamint a magasabb adatbiztonsági szint egyaránt növelhető, sőt, hibatűrő rendszer alakítható ki. A RAID ezeket a tulajdonságokat több merevlemez együttes alkalmazásával valósítja meg. Attól függően, hogy a kezelését hardver vagy szoftver végzi, hardveres vagy szoftveres RAID-ről beszélünk. A hardveres RAID speciális eszközöket igényel, alkalmazásuk általában nem függ az operációs rendszerektől. A szoftveres RAID csak olyan operációs rendszereknél alkalmazható, melyek tartalmazzák ennek támogatását. A Windows NT és a Windows 2000 tartalmazza a RAID 0,1,5 szintek támogatását.
A szoftveres RAID a processzor és az I/O rendszer számára többletterhelést jelent a RAID folyamatos számítási feladatainak következtében. Ez a mai GHz közeli processzorok esetében nem okoz túlterhelést, de kisebb processzoroknál vagy kiélezett esetekben problémát okozhat.
A merevlemezek elhelyezése a csatolókon szintén teljesítményprobléma forrása lehet. Minél több csatolón *(SCSI, IDE)* helyezzük el a stripe egységeit, annál nagyobb számítási és buszteljesítményt veszünk igénybe. Célszerű tehát a stripe egységeit egy vezérlőn elhelyezni.

Az alkalmazott merevlemezek száma újabb problémát vet fel. A merevlemezek számának növelése a rendszer összteljesítményének jelentős részét veheti igénybe. Másrészt minél több a merevlemez, annál nagyobb a hibalehetőség.
Ezekre a problémákra a hardveres RAID vezérlők jó megoldást nyújtanak, bár intenzív mentés nélkül a stripe alkalmazása veszélyes a hibatűrés hiánya miatt.

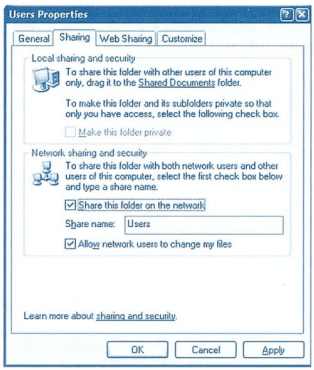
Az alábbi lehetőség már az NTFS gyorsítására vonatkozik:
Ha a naplózási és biztonsági információkra nem tartunk igényt kikapcsolhatjuk az elérési idők frissítését a Windows Explorerben. Ez olyan esetekben segíthet, ahol nagy a mappák száma egy partíción. A REGEDT32 elindítása után a `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\FileSystem` kulcsot kell szerkeszteni. Szerkesztés, érték hozzáadása:

Név: NtfsDisableLastAccessUpdate
Adattípus: REG_DWORD
Érték: 1

K: *Feltelepítettem a Windows XP-t, de valamiért eltűnt a mappák tulajdonságlapjáról a „Security" oldal, pedig a fájlrendszer ott NTFS... és a megosztás is valami nagyon furcsán működik! Mi történt? És egyáltalán, mi történik, amikor a fájlmegosztó varázsló varázsol? Kérem vissza a régi jól megszokott, átlátható felületet!*

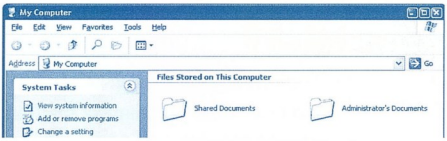
V: A Windows XP alapértelmezésben az úgynevezett „Simple File Sharing" *(egyszerűsített fájlmegosztás)* üzemmódban működik. Ilyenkor a felhasználó elől elrejti a fájlok/mappák tulajdonságlapjának „Security" oldalát, a „Sharing" pedig bizony átalakul varázslók gyülekezetévé:

Az ábrán látható oldal a Butított (na jó, Egyszerűsített)



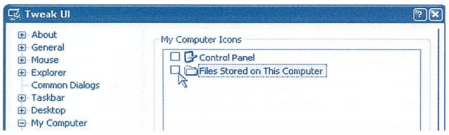
Fájlmegosztás eredménye. A Security fül – jól látható, hogy – nem látható

Mielőtt visszaváltanánk hagyományos üzemmódba, azért nézzük meg, ezen az oldalon mi mit jelent! A Local sharing and security (helyi megosztás és biztonság) mezőben a mappa/fájl helyi, számítógépen belüli megosztásának lehetőségét találjuk. A felirat tanúsága szerint a fájlt vagy mappát a helyi felhasználók között úgy oszthatjuk meg, hogy azt *(egészen pontosan annak egy másolatát)* bevonszoljuk a Shared Documents mappa alá.



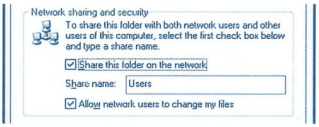
A Shared Documents mappa tartalmát minden helyi felhasználó eléri

A Shared Documents mappa – amit egyébiránt a My Computer ablakon keresztül érhetünk el – nem más, mint az „All Users" profil My Documents mappája – azaz egy mindenki számára elérhető közös tárolóhely, a <felhasználónév>'s Documents mappa pedig az aktuálisan bejelentkezett felhasználó My Documents könyvtára. Ha valakit zavar, hogy ezek a mappák *(ha az aktuálisan bejelentkezett felhasználó rendszergazdai jogokkal bír, akkor ráadásul minden helyi felhasználó mappája is)* megjelennek a My Computer ablakban, az **[1]** címről letölthető Windows XP PowerToys részeként telepíthető Tweak UI programocska segítségével kikapcsolhatja:



A Tweak UI segítségével kikapcsolhatjuk a speciális mappák megjelenítését (is)

Az egyszerűsített fájlmegosztás oldal alsó fele a hálózati megosztásra koncentrál *(emlékeztetőül ismét a két lehetséges választási lehetőség)*:



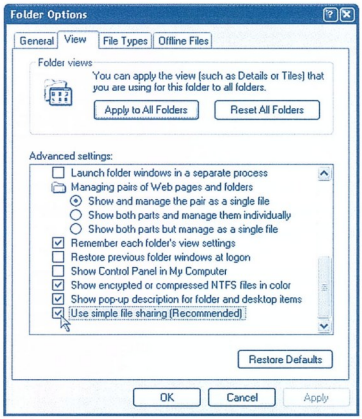
Egyszerűsített fájlmegosztás a hálózaton

A felső opció önmagáért beszél; bekattintásával megosztjuk a mappát a többi felhasználó részére. Ilyenkor a varázsló felveszi az Everyone csoportot mind a mappa NTFS jogai közé, mind a megosztás jogosultságlistájába. Hogy milyen jogosultságokkal, azt az alsó pipával határozhatjuk meg *(„Allow network users to change my files")*:

Everyone jog	Allow network users	Allow network users
Lokális NTFS	Modify	Read & Execute
Megosztás	Full Control	Read

A varázsló által különféle üzemmódokban a megosztás illetve a mappa NTFS jogosultságlistájához hozzáadott Everyone jogosultságok táblázata

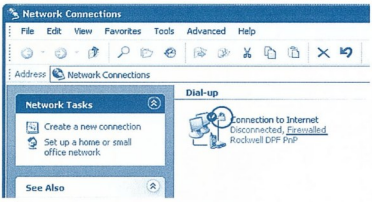
Ha valakinek ez nem megfelelő, az egyszerűsített fáljmegosztást *(a Windows XP Professional-ban)* egyetlen jól irányzott kattintással kikapcsolhatja a mappák tulajdonságlapjának View oldalán *(jól tesszük, ha a beállítást a rendszer összes mappájára érvényesítjük)*. Ehhez a listában töröljük a „Use simple file sharing *(Recommended)*" opció előtt álló pipát!



Az egyszerűsített fájlmegosztás letiltása egyetlen kattintás – ha tudjuk, mit keressünk!

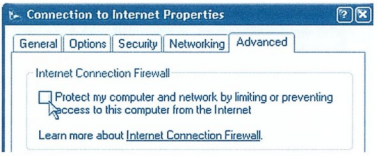
K: *Eddig más operációs rendszerem volt, és nagyszerűen működött minden. Amióta feltelepítettem a Windows XP-t, bizonyos dolgokat nem érek el az Interneten. Érdekes, hogy teljesen zökkenőmentesen működik a web-elérés, levelet is tudok küldeni-fogadni, de például az IRC már nem hajlandó működni. Vajon miért?*

V: Előfordulhat, hogy a Windows XP beépített tűzfala, az Internet Connection Firewall *(ICF)* van a háttérben. Az Internetcsatlakozás varázsló ugyanis a kapcsolat ikonjának létrehozásakor alapértelmezésben felajánlja az ICF bekapcsolását. Ha „figyelmetlenek" vagyunk, ez bekapcsolva maradhat. Ha az ICF aktív, minden kifelé induló kapcsolatot engedélyez *(ezért működhet zavartalanul a webezés, levelezés – mind-mind kifelé indított kapcsolat)*. Vannak azonban olyan szolgáltatások – és ilyen például az IRC is – amit nem vehetünk igénybe, ha a számítógépünk nem válaszol bizonyos külső kérdésekre. Nincs más teendőnk, mint kiütni a tűzfalból néhány téglát... csak az a kérdés, milyen téglákat, és hogyan?
Mindenekelőtt, bizonyosodjunk meg róla, hogy a problémánkat biztosan a tűzfal okozza-e. Az Internetcsatlakozás ikonján a tűzfal jelenlétét kis lakat jelzi.



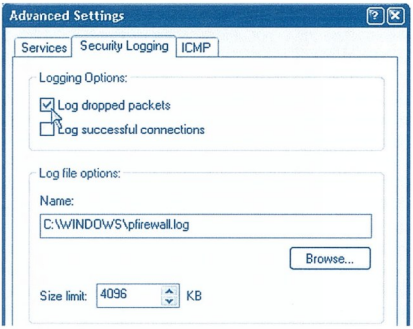
Ha az ikon jobb felső sarkában látható a kis lakat, a hálózati kapcsolatot az ICF vigyázza

Első lépésként a kapcsolat tulajdonságlapján kapcsoljuk ki a személyes tűzfalat, majd csatlakozzunk újra az Internetre.

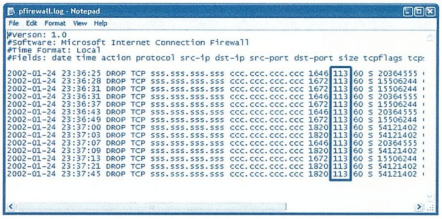


A tűzfalat az Internetkapcsolat tulajdonságlapjának Advanced oldalán kapcsolhatjuk ki-be

Ha kikapcsolt tűzfallal működik a kívánt dolog, bekapcsolt tűzfallal pedig nem, akkor érdemes meglesni, hogy pontosan mit szűr ki az ICF. Ehhez kapcsoljuk be újra a tűzfalat, majd a „Settings" gombra kattintva nyissuk meg az ICF beállításait tartalmazó dialógusablakot.

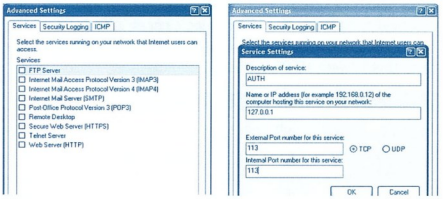


Itt válasszuk ki a „Log dropped packets" opciót, állítsuk be a logfájl kívánt nevét, majd csatlakozzunk újra az Internetre, és próbálkozzunk. Néhány sikertelen próbálkozás után keressük meg a naplófájlt és nézzük meg, milyen csomagok érkeztek esetleg, amit a tűzfalunk visszautasított:



A tűzfalnaplóban látszik, hogy az ICF milyen csomagokat utasított vissza

A naplófájl oszlopaiból kiderül, hogy honnan *(sss.sss.sss.sss)*, és milyen címre *(ccc.ccc.ccc.ccc)* tartott az elutasított csomag. Ha a ccc.ccc.ccc.ccc a saját IP címünk, akkor a csomag bejövő típusú volt. Ellenőrizzük azért a túloldal IP címét is, nehogy véletlenül rossz kapukat nyissunk ki. A bejövő csomagoknál a beérkező *(cél, azaz destination)* port címe a lényeges, ezt a kaput kell kinyitnunk, hogy a csomagok meg is érkezzenek hozzánk. Látható, hogy az esetünkben elfogott csomagok a 113-as portra érkeztek *(ez az úgynevezett AUTH, más néven IdentD protokoll)* – az IRC-hez ez tényleg kell.



A beengedett-publikált szolgáltatások listája

Az ICF lehetővé teszi, hogy a tűzfal mögött, a saját hálózatunkon – vagy akár a tűzfalat futtató gépen – található szolgáltatásokat tegyünk közzé az Internet felé. Ha be akarunk engedni valamilyen hálózati forgalmat, nincs más dolgunk tehát, mint publikálni az adott szolgáltatást – esetünkben nyissuk meg az ICF tulajdonságlapját, majd vegyünk fel *(és engedélyezzünk)* egy új bejövő protokollt a naplóból kilesett porton. A belső szolgáltatás IP címét pedig állítsuk a saját gépünkre – a 127.0.0.1 IP cím tökéletesen megfelel a célra. A kapcsolat újraindítása után az ICF be fogja engedni az adott portra érkező hálózati forgalmat.

A cikkben szereplő URL-ek:
[1] http://msdownload.netacademia.net/info.asp?prid=187

Nomádélet a XXI. században: mobil IP

Az emberi szabadságvágy egyik ősidők óta jelenlevő megnyilvánulási formája a helyváltoztatás, illetve a cselekvés minőségét *(kényelem, sebesség stb.)* is figyelembevevő mobilitás. Amit a honfoglaló őseinknek a lovak és szekerek jelentettek, azt a XX. század embere számára az autó, a vonat, a repülő testesítette meg. Az utazás ma már az egyszerű halandók mindennapi életének is részévé vált.

A kommunikáció fejlődése mindezek mellett egyre meghatározóbban vesz részt a mobilitás fogalmának alakulásában. A mobilitásnak ez az aspektusa azt kívánja, hogy számítástechnikai és kommunikációs szempontból mindenütt „otthon" érezzük magunkat. Én például minden macera nélkül szeretnék elektronikus képeslapot küldeni ismerőseimnek, amint szabadságomat töltöm a Rivierán *(főleg ezt szeretném :-)*, vagy egy külföldi vállalatnál tett látogatás során is jó lenne a saját vállalatom csoportmunka-rendszerét, belső állományait használni, pontosan úgy, mintha a helyemen ülnék. Az Internet hálózati protokollja ebben komoly gátat jelent, hiszen odaláncol bennünket, ahol kapjuk az IP címet. Folyton azt halljuk, hogy az internetes és a mobil világ konvergál. Az alkotók dicséretére váljék, a mobilosok az IP mellett döntöttek. Ha a víziók valóra válnak, akkor a közeljövőben a számítógépek és a mobiltelefonok párosításából született kütyük millióit áraszták el a világot és a világhálót. Ahány gép, annyi IP cím, így az IPv4-től azonnal el is búcsúzhatunk *(hiába hirdeti a 2002-es RFC, hogy támogatja a mobilitást)*, jöhet az IPv6.

Önmagában ezzel sem válik az IP cím hordozhatóvá, de a tervezők tanultak az elődök hibáiból, és flexibilisebb protokollt terveztek, amely igény szerint bővíthető. Ennek megfelelően léteznek mobil IP kiterjesztések is *(lásd* **[1]***)*. Az alapkoncepció szerint egy Home Agent-nek *(HA)* nevezett kiszolgáló felelős hollétünk nyilvántartásáért. Idegenben kapunk egy ideiglenes címet *(a DHCP már része lett az IPv6-nak)*, majd jelentjük a HA-nak, hogy mostantól ide *(Care-of-Address – CoA)* küldje a csomagjainkat. Sőt, a HA olyan rendes, hogy szól a velem *(Mobil Node – MN)* éppen kapcsolatban lévő gépnek *(Correspondent Node – CN)*, hogy egyből az új helyemre küldözgesse amit akar, ne kelljen szegény IP csomagoknak feleslegesen ekkora utat megtennie *(triangular routing)*, aztán meg azon szégyenkeznie, hogy zabálja a sávszélességet ott, ahol egyébként lennie sem kellene. Mindezt egyszerűen csak Mobil IP-nek, vagy még rövidebben MIP-nek hívják.

Semmi és senki sem lehet tökéletes, miért éppen a MIP és az IETF lenne az? Vegyünk egy olyan forgalmas helyet, ahol a cellák kicsik, és így a határaik közel esnek egymáshoz. Ez annyit jelent, hogy kütyünk mozgás közben gyakran vált IP címet. Most már tudjuk, hogy ilyen esetben szükséges egy üzenetváltás a HA-val, ami ily módon értesül arról, hogy nem bírunk egy helyben maradni. Ekkor pont a kisebb kapacitású vonalakat fogjuk terhelni, és ez cseppet sem szerencsés. Egy újabb szereplő *(Mobility Anchor Point – MAP)* felbukkanása azonban véget vet a pazarlásnak. A MAP több hálózatért felelős, és a HA válláról mindaddig leveszi a terhet, amíg a felügyelete alatt lévő területet nem hagyjuk el. Addig csak neki tartozunk bejelentési kötelezettséggel. A CoA ekkor önmagában már kevés, hiszen kell egy olyan cím, ami a CN és a HA számára változatlan, míg én a MAP tartományában többször is címet változtathatok. Előbbi a MAP tartományára jellemző *(Regional Care-of-Address – RCoA)*, míg az utóbbi annak a hálózatnak egy címe amelyikben éppen tartózkodom *(on-Link Care-of-Address – LCoA)*. Mivel ez már jóval több a MIP-nél, hozzátettek egy H betűt *(Hierarchical Mobile IP – HMIP)*, ami a rendszer hierarchikus felépítését hivatott jelezni.

Először kissé csalódottan vettem tudomásul, hogy míg a nagy telekommunikációs cégek esetenként már az IETF draft megjelenése előtt elkészülnek az ahhoz tartozó fejlesztéssel, addig a Microsoftnak *(publikus)* HMIP implementációja egyáltalán nincs, és a MIP-ből is csak egy viszonylag régi *(majdnem 1 éves, két verzióval ezelőtti)* draftnak megfelelő programjuk van. De lássuk be, a Microsoft nem telekommunikációs cég, így ne várjuk el egy tőlük még kissé távolabb eső területen is az élen járást. Mivel a *(H)*MIP az elkövetkező néhány évben még biztosan nem jelent majd komoly üzletet, a fejlesztést a Lancaster University lelkes ifjúságára bízták. Néhány egyetemista fejleszti a kódot, a Microsoft és az egyetem közötti együttműködést fedő LandMARC *(Lancaster and Microsoft Active Research Collaboration)* projekt keretében. A program és a dokumentáció letölthető a **[2]** címről. *(Három gép kell hozzá: MN, HA, CN!)*

Talán nem tartozik szorosan a tárgyhoz, de azért is felmentést adok az MS-nek a fejlesztési hátrányért, mert a szabványok, helyesebben ajánlások tervezetei *(IETF draft)* minősíthetetlen állapotban vannak. A MIP több éves csiszolgatás után még mindig csak *(helyesírási és hivatkozási hibáktól sem mentes)* draft, abból viszont már a 15., és áprilisban várható a még mindig nem végleges 16. változat. Konszenzus nincs, és élénk vita dúl a biztonsági kérdések körül. A HMIP draft 4. változatába olyan hiba is becsúszott, amitől a protokoll nem is működhetett volna. Elgondolkodtató, hogy az erősen bürokrata vonásokat mutató IETF alkalmas-e még egyáltalán arra, hogy az Internet számára szabványokat dolgozzon ki. Egyes cégek, talán ráunva a szektorban nehezen tolerálható tempóra, saját protokollok kifejlesztésébe kezdtek. Az iparág helyzete tehát – tömören szólva – divergensen konvergens.

A számítástechnika és a mobil világ találkozása alaposan összekuszálta a már-már kristálytisztán kirajzolódó trendeket. Fogjuk fel azonban úgy, hogy egy olyan korban élünk, amely egyszerre tárja fel előttünk szakmánk szép és kevésbé szép oldalát. Egy biztos: IP goes mobile!

Zacco

A cikkben szereplő URL-ek:

[1] http://www.ietf.org/html.charters/mobileip-charter.html

[2] http://research.microsoft.com/programs/europe/projects/MIPv6.asp

Rendezvénynaptár

2002 tavaszán is folytatódik a TechNet Szemináriumok sorozata. A Microsoft Magyarország rendszermérnökei és vendégeik az előadások során áttekintik a kereskedelmi webhelyek biztonságát növelő technológiákat; kitérnek arra, hogy a Windows platform milyen eszközökkel segíti a vállalati webes alkalmazások fejlesztését; betekintést adnak az SQL adatbázisok adminisztrációjába, valamint ismertetik a hamarosan megjelenő Windows .NET Server és Project 2002 újdonságait. Mindezt természetesen a jól ismert „100% technológia, 0% marketing" szabály betartásával teszik.

A következő félév rendezvényei:

2002. 03. 06.: Windows alapú webes alkalmazások

A webalapú technológiák terjedése a vállalati alkalmazások körében is tapasztalható: egyre több ilyen alkalmazás jelenik meg mind belső, mind külső célokra kifejlesztve.

Az előadás áttekinti a Windows 2000 és a Windows XP platform azon elemeit, melyek segítségével egyszerűen és olcsón készíthetünk ilyen alkalmazásokat. Kitér a Windows-platformon elérhető ingyenes alkalmazásfejlesztő eszközökre *(.NET Framework, parancsnyelvi eszközök)*, adatbáziskezelőkre *(Microsoft Destop Engine)*, valamint néhány biztonsági és licencelési kérdésre.

2002. 03. 20.: Biztonságos Windows

A Windows 2000 Server operációs rendszert széleskörűen használják az Interneten. Méginkább igaz ez a kereskedelmi webhelyekre. A biztonság kérdése már a tervezés során is rendkívül fontos szerepet játszik. Az előadás során ismertetjük a Microsoft által megalkotott Solution Offering termékcsoport részét képező, az Interneten üzemelő adatközpontok *(Internet Datacenter - IDC)* számára eltervezett hálózati architektúrát, és áttekintjük az üzemeltetés során felmerülő biztonsági kérdéseket. Szó lesz a 2001 őszén meghirdetett stratégiai technológia védelmi programról *(STPP)* is, amely a Windows 2000 alapú rendszerek biztonságossá tételét és szinten tartását célozza meg. Megmutatjuk a program részeként elérhető IIS Lockdown Tool-t és a Microsoft Security Tool Kit CD-t.

2002. 04. 03.: Vállalati szintű projektmenedzsment

Az összetett, sokszereplős projektek informatikai támogatása komplex feladat, érinti a hálózati és kommunikációs infrastruktúrát, a kiszolgáló- és ügyféloldali alkalmazásokat és a rendszerfelügyeletet is. Az előadásban a hamarosan megjelenő Project 2002 termékcsalád felhasználásával bemutatjuk, hogyan alakítható ki és tartható kézben a projektek tervezéséhez, a kapcsolódó anyagok tárolásához, a résztvevők kommunikációjához, illetve a követéshez és elemzéshez szükséges környezet. Ismertetjük a fájl- és intranetkiszolgálókkal, a levelezőrendszerrel, az adatkezelő és -elemző rendszerekkel való kapcsolódást. Külön figyelmet szentelünk a nagyvállalatok projektkezelési igényeinek, így például a szervezeti szintű erőforrásmenedzsment problémakörének, és bemutatjuk az ezekre választ adó új Project Server terméket.

2002. 04. 17.: Üzemeltetői Konferencia

A hagyományokhoz híven a 2002 tavaszi üzemeltető konferencián is a Microsoft rendszerek üzemeltetéséhez szeretnénk segítséget nyújtani. A tervezett előadások témái között megtalálható a Windows 2000 és a Windows XP egymás mellett élésének zökkenőmentessé tétele, Az SQL és Exchange szerverek üzemeltetésének fogásai és természetesen az elmaradhatatlan internetes biztonsággal foglalkozó előadás is. Sort kerítünk a virtuális magánhálózatok kialakításának üzemeltetőket érintő kérdéseire és a Windows alapú rendszereket összekötő hálózatokkal kapcsolatos napi feladatokra. Ezúttal is fogunk érdekes és néha meglepő dolgokról beszélni és megpróbáljuk az őszi konferencián a résztvevők által kért témákat is az előadások sorába iktatni.

2002. 05. 08.: A Windows.NET Server technikai újdonságai

A Windows 2000 Server-t követő új kiszolgálócsalád, a Windows .NET Server szolgáltatásait ezúttal nem tárgyaljuk témakörönként külön előadásokon, ez az áttekintő előadás elsősorban a Windows 2000-hez képest történt fejlesztésekre és újdonságokra koncentrál. Ismertetjük az Active Directory címtárszolgáltatás új funkcióit, a vegyes Windows 2000/Windows .NET címtár környezetek használatát, az új csoportos házirendeket, a rendszerfelügyeleti újdonságokat. Terítékre kerül az Internet Information Services 6.0 teljesen átalakított architektúrája, a kibővült funkcionalitású terminálszolgáltatás. Tárgyaljuk a biztonsággal kapcsolatos fejlesztéseket, kitérve a PKI infrastruktúrával kapcsolatos újdonságokra, a hálózatkezelés terén pedig a Windows XP munkaállomásban nem szereplő kiszolgálóoldali kibővült szolgáltatásokat érintjük.

A TechNet-előadások a már megszokott helyszínen, a Lurdy Házban található Hollywood Multiplex moziban „játszódnak", míg az Üzemeltetői Konferencia helyszínét később tesszük közzé.

Az egyes előadásokkal kapcsolatos részletes információkat - tervezett napirend, előadók, stb.- a rendezvényeket megelőző hetekben tesszük közzé rendezvénynaptárunkban. Ugyanitt elérhetők a **jelentkezési lapok**, illetve - a rendezvényeket követően - a bemutatók PowerPoint diái is.

`http://www.microsoft.com/hun/events/technet.asp`

Ugróiskola.

Felkészülés valami komolyabbra!

A tavaszi pezsgést a magunk részéről mi is szeretnénk élénkebbé tenni. Beindítjuk Ugróiskola előadássorozatunkat, ahol azonnal hasznosítható ismeretanyag átadására törekszünk. Aki beugrik hozzánk, az úgy jut tovább élete következő mezőjére, hogy valami hasznos szerszám-eljárás birtokába jutott.

Az ugróiskola főbb célkitűzései:

1. Ne raboljuk egymás idejét!

Egy-egy alkalom maximum 2 óra hosszúságú. Ez azt jelenti, hogy:

- **nem veszünk el egy fél munkanapot sem az életéből**
- **ami nem érdekli a sorozatból, azon nem vesz részt**
- **nem kell szólnia senkinek, engedélyt kérnie senkitől, hogy tanulni megy** *(anyagi támogatást sem kell kérnie főnökétől, lásd alább)*
- **nem kell újratölteni a parkolóórát**

2. Tapasztalat, tapasztalat, tapasztalat!

Amit kézzel nem tudunk megfogni, az nem is létezik. Ha ez tanulnivaló, akkor gyakorlás híján elillan. Épp ezért:

- **minden témakörhöz számítógépes gyakorlat társul. Minden ki lehet próbálni tantermünkben**
- **az egyes témakörökhöz nyomtatott háttéranyag áll rendelkezésre**
- **felkészült oktatóink segítenek túljutni a buktatókon**

Windows 2000 Scripting ugróiskola – felkészülés valami komolyabbra

Témakör	Gyakorlat	Időpont
A Windows Scripting Host és a scriptnyelvek *(VBScript, JavaScript)*	Scriptek: az egyszerűtől a bonyolultig. Hello worldtól a scriptvírusokig.	Áplris 5, péntek
Címtárkezelés. Active Directory Services Interface *(ADSI)*	Felhasználók, csoportok tömeges létrehozása, módosítása, mozgatása az AD hierarchiában. Jelszóállítás.	Április 12, péntek
Levelezés. Collaborative Database Objects *(CDO)*	Automatikus levélküldés bizonyos rendszeresemények *(rendszerindítás, service leállása stb.)* hatására, levél fogadása.	Április 19, péntek
Rendszerfelügyelet. Windows Management Instrumentation *(WMI)*	A rendszer állapotának lekérdezése, megjelenítése weblapon, *(táv)*felügyeleti parancsfájlok készítése.	Április 26, péntek
Adatbáziskezelés. ActiveX Database Objects	Adatbázis tartalmának lekérdezése, megjelenítése. Adatmódosítás. Tárolt eljárások hívása	Április 30, kedd

Helyszín: CEU Konferenciaközpont, Kerepesi út 87.

Kezdési időpont: reggel 9 óra.

Jelentkezés: www.netacademia.net/ugrosuli

A rendezvényeket számítógépekkel felszerelt tantermekben tartjuk, így minden ismeret kipróbálható, begyakorolható.

NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

Következő számunk tartalmából

Következő számunk tartalmából

RIS
Az utolsó részben megnézzük hogyan lehet több telephely esetén a RIS kiszolgálókat szinkronban tartani, valamint az Ügyféltelepítő Varázsló *(CIW)* képernyőket fogjuk átszabni.

W2k
Járjuk be együtt a Windows 2000 rétegzett hálózati architektúráját! Hogyan kapcsolódnak egymáshoz a komponensek? Hány szolgáltatás áll, ha „at least one service failed to start"?

W2k
Dinamikus DNS
A Windows 2000 Active Directory lelke a DNS - és az sem árt, ha az a DNS kiszolgáló bizony képes a dinamikusan érkező bejegyzéskérelmek fogadására és feldolgozására. Cikkünkben a DDNS protokollt, valamint a Windows 2000 DHCP kiszolgáló, a Windows 2000 kliensek, valamint a DNS kiszolgáló együttműködését mutatjuk be.

Farkasokkal táncoló
Ezúttal is alapozó munkát végzünk, folytatjuk az ismerkedést az erőforrástípusokkal. Nem ígérhetek könnyű olvasmányt: minden típusnak megvan a maga bogara, és nem árt ezekkel tisztában lenni, mielőtt használni akarjuk őket.

Jog
Az elektronikus kereskedelem körén túl – amint azt a már elemzett jogalkotói szándék is mutatja – az Internet számos egyéb kérdésére próbál meg jogalkotói választ adni a törvény. Így az amerikai és európai szabályozással összhangban rendezi a szolgáltatók egyes tevékenysége esetén felmerülő felelősség kérdését, továbbá a szerzői jogi jogsérelem esetére bevezeti a notice and take down eljárást.

.NET
Az alaptípusokat tovább boncolgatjuk, megnézzük mik azok az interfészek, miért fontosak számunkra, mire valók a delegate-ek, és ehhez kapcsolódóan hogyan működik a .NET natív eseménykezelése?

XML
Folytatjuk a relációs adatbázisok és az xml világ közötti átjárást. Tovább boncolgatjuk a DataSet osztály szolgáltatásait, és megszemlélünk egy valódi virtuózt, XmlDataDocument objektumot.

ASP Suli
Exchange postaláda elérése mobiltelefonon? Nem gond! ASP Sulinkban még egy cikk erejéig maradunk a mobil eszközök programozásánál. A következő számban a CDO for Exchange 2000 és az ADO segítségével megvalósítjuk az Outlook Wap Access szuperkompakt változatát.

Exchange – új sorozat
Méltatlanul elfeledett termékre térünk vissza: az Exchange Serverre. Nem a felhasználók feledték el, hanem mi nem törődtünk vele ezidáig megfelelően. Most útjára indítjuk Exchange Server sorozatunkat, mely a telepítéstől kezdve végigvezet a termék használatának rejtelmein.

RAS
Az útválasztási protokollok sorában a RIP-nál tartottunk. Kis hálózatokon jó lehet a RIP, de korábban ismertetett korlátai (különösen a lassú konvergencia, valamint a hurokgyanú) miatt nem mindenütt használható. Ezzel szemben az OSPF szinte mindent tud már. Nem útvonaltáblákat, hanem egy úgynevezett Link State Table állapottábla sorait küldi szét partnereinek, amelyek tetszőlegesen bonyolult hálózatban megtalálják a pillanatnyilag legolcsóbb utat.

Előfizetési szelvény

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

FAX

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

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

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

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

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

Az előfizetés kezdete:...../.....

Előfizető neve:. . . . . .

Cég neve: . . . . . .

Cím: ☐☐☐☐ . . . . . .

E-mail cím: . . . . . .

Telefon: . . . . . .

Fax:. . . . . .

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

Kelt:...../...../..... Aláírás:.....

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

Számlázási cím: **Szállítási cím:**

....

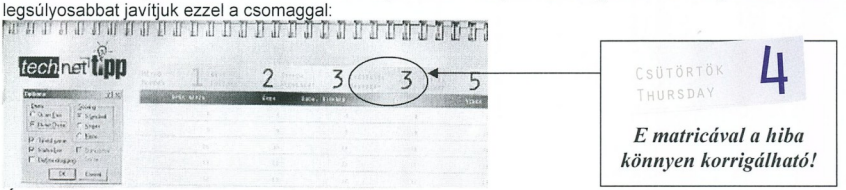
....

tech.net
working with windows

NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

Bugos naptár SP1

Mint azt bizonyára többen észrevették, ezévi naptárunk - a szoftveriparban talán nem váratlanul – hibásan került az Önök kezébe. Felelősségünket vállalva ezúton kibocsátjuk hozzá első javítócsomagunkat. Az eddig észlelt hibák közül az egyik legsúlyosabbat javítjuk ezzel a csomaggal:



Április hónapban nincs negyedike?!

A bugvadász verseny természetesen tovább folytatódik, a naptár korántsem vált hibátlanná. Az összes hiba felfedezéséért értékes jutalmat adunk! Megfejtését erre a címre várjuk: info@netacademia.net

infoBYTE

INFOKOMMUNIKÁCIÓ
A távközlési piac liberalizálása és a mobiltelefóniában várható generációváltás érdekegyeztetési törekvéseiről tudósít ez a rovat.

CÍMLAPSZTORI
A hónap vezető cikke új technológiákról, megoldásokról.

NEMZETKÖZI SZEMLE
Külfödi hírek, kitekintés.

E-KORMÁNYZAT
Az Informatikai Kormánybiztosság 2001– 2002-ben összesen 36 különféle programot koordinál. Az információs társadalom kiépítésének lépcsőfokai ezek.

INFORUM
Az Inforum célja, hogy párbeszédet folytasson a szakma és a kormányzat között. Aktív szerepet vállalt a szerzői jogi, az egységes hírközlési és az elektronikus kereskedelmi törvény megalkotásában.

EU-INFORMATIKA
Ebben a rovatban nem elsősorban a technológiára, hanem a megvalósult projektekre, az EU-kompatibilitás kérdéskörére, pályázatokra koncentrálunk.

CIO.INFOBYTE.HU
Az infopen.hu webmagazin és az infoBYTE közös rovata, kifejezetten IT vezetők számára. A rovat egy aktív CIO-val készült átfogó interjúval indul, amelyet stratégiai jellegű technológiai áttekintések, szakcikkek követnek. A rovat hangsúlyos részei a vállalati IT megoldásokat bemutató esettanulmányok, de interjúk, konferenciatudósítások formájában a piac meghatározó megoldásszállító cégeinek üzleti és termékstratégiájának bemutatása is helyet kap.

KARRIER
Tapasztalatok szerint három-négy évente változtatunk mi, informatikusok állást, szakmát vagy szakirányt, és ez a szám a jövő évtizedekben valószínűleg csökkenni fog.

KONZOL ELŐTT
E rovatunkban olvasóink nevében és helyett Novell szakértőket kérdez a szerző.

DR. WATSON
A NetAcademia-féle mélyvíz-tanfolyamokra iratkozhatnak be azon olvasóink, akik Dr. Watson nyomában járnak.

MÉRLEG
Hardver- és szoftvertesztek röviden, velősen.

PROCESSZOR
Sokakat érdeklő CPU-újdonságok mélységei a fejlesztéshez közel álló szakértők tollából.

Kérjen mintapéldányt: minta@infobyte.hu

Kihelyezve bejön!

SZÁMALK Hivatalos Microsoft tanfolyamok

Budapesten és kihelyezett formában már az ország több nagyvárosában is!

Kód	Megnezezés	Időt.	Ár	Kezdési időpontok						
1560	**Updating Support Skills to Windows 2000**	5 nap	139 000	okt. 15. nov. 19.	okt. 8. nov. 12.	okt. 1. nov. 5.	okt. 8. nov. 12.	okt. 1. okt. 15.	okt. 8. nov. 12.	okt. 1. nov. 5.
2152	**Implementing Windows 2000 Server and Professional**	5 nap	125 000	okt. 15. nov. 19.	okt. 8. nov. 12.	okt. 1. nov. 5.	okt. 8. nov. 12.	okt. 1. okt. 15.	okt. 8. nov. 12.	okt. 1. nov. 5.
2153	**Implementing Windows 2000 Network Infrastructure**	5 nap	125 000	nov. 19.	dec. 3.	nov. 26.	dec. 3.	nov. 19.	dec. 3.	nov. 26.
2154	**Implementing Windows 2000 Directory Services**	5 nap	125 000	dec. 10.	jan. 7.	dec. 17.	jan. 7.	dec. 10.	jan. 7.	dec. 17.
1572	**Implementing, Administering Exchange 2000**	5 nap	139 000	nov. 19.	dec. 3.	nov. 26.	dec. 3.	nov. 19.	dec. 3.	nov. 26.
1561	**Designing Windows 2000 Directory Services**	3 nap	95 000	dec. 10.	jan. 7.	dec. 12.	jan. 7.	dec. 10.	jan. 7.	dec. 17.
2159	**Deploying and Managing ISA Server 2000**	2 nap	79 000							

További tanfolyamokról és időpontokról érdeklődjön szervezőinknél! Kérésére részletes tájékoztatót küldünk.

Minőség, egyéneknek kedvező konstrukciók, cégeknek partneri kedvezmények!

Microsoft CERTIFIED Technical Education Center

SZÁMALK TOVÁBBKÉPZÉS

SZÁMALK

☎ 203-0304/3050 **www.szamalk.hu/tisza** 1115 Budapest, Etele út 68.