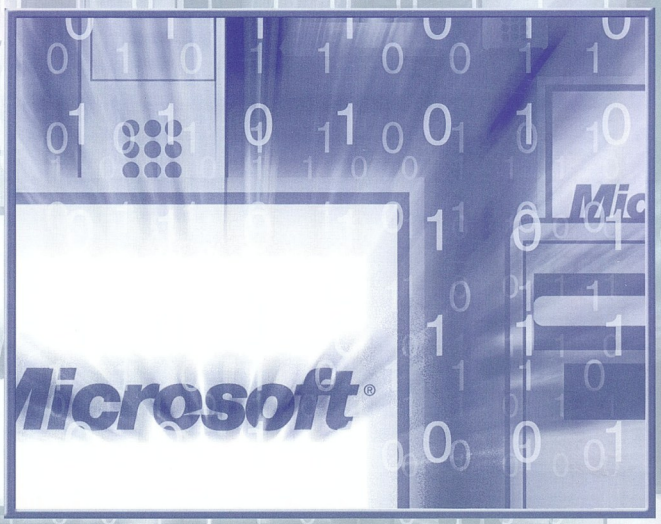
1.344 Ft

II. Évfolyam 1. szám

ISSN 1586-5185 01

tech.net

A Microsoft Magyarország szakmai magazinja



A hálózati forgalom elemzése

Windows 2000 • Policy sablon-fájlok szerkesztése *(II. rész)*

Business Internet • MetaBase: Biztos háttér az IIS mögött

BackOffice • Tárolási megoldások az Exchange 2000 Server-hez

Jóslat

Azt hittem, megúszhatom. A jóslást. De hiába lapítottam, sunyítottam, már január első napján megérkezett az első felkérés, hogy mondjam meg, mit hoz a jövő, a következő év az informatikában. Mintha belátnék a Microsoft boszorkánykonyhájába! Mintha látnám a jövőt! Látom is! Látom a saját jövőmet, a családomét, a közvetlen környezetemét. Miután az ember fél méter sugarú körben mindennek ura, nem is csoda, hogy saját sorsát saját elképzelései alapján vezérli, s látja, formálja a jövőt. De Redmond és Amerika kívül van e bűvös körön. Semmi eséllyünk az ott zajló folyamatok befolyásolására, sőt megértésére sem. Ami a Microsoft jövőjét illeti az kizárólag egyetlen ember „látja", azaz formálja, ez Bill Gates, még akkor is, ha külső tényezők néha megzavarják ebben *(per)*. De jöjjön, aminek jönnie kell, elmondom, mit „tudok" a jövőről.

Feldarabolás

Sokakat érdekel, vajon végül feldarabolják-e a Microsoftot. Natív spekulációval arra a következtetésre jutottam, hogy ***1)*** ebben az évben biztosan nem ***2)*** de ha mégis, akkor sem lesz ennek a lépésnek semmilyen látható hatása sem a Microsoftra, sem az informatikára, mert ma már teljesen máshol található a cég súlypontja, mint ahol a per kezdetén volt. Az Office, a Windows és az Internet Explorer megkérdőjelezhetetlen győzelme miatt talán nem meglepő, hogy a Microsoft más piacok felé fordult. Ha valami katasztrófa folytán akár mindhármat leválasztják a cégről – az is csak levedlett bőr. Alatta már alakul, fejlődik az új. S több év óta először a meghódítandó piac egy egyelőre nem létező piac – a .NET. Nem volt felesleges Billyt kivonni a napi munkából, láthatóan több ideje marad gondolkodni. Egészen 1995-ig ért rá gondolkodni: „PC-t minden asztalra" mondta, amikor a PC még hobbygép volt. „Minden adatbázis megérdemli a relációs adatbáziskezelő motort" amikor az SQL alapú kiszolgálók milliárddolláros beruházást igényeltek. „640 kilobájt mindenre elég lesz…." Tévedni emberi dolog. „Where do you want to go today?" Oda, ahol még senki sem járt.

.NET

A dotnet többet jelent egyszerű ötletnél. Hosszú évek után ez az első ötlet a Microsofttól, mellyel ismét átveszi a jövőformálást, az egész IT világ szakmai fejlődésének vezetését. Hogyhogy? Hát elveszítette? Igen, el. 1995-ben. Még emlékszem a Windows 95 kibocsátása előtti izgalomra, amikor először kínált az operációs rendszer világhálót. A Microsoft-féle Bluebird kódnevű *(erre lehet, hogy már nem jól emlékszem)* saját megoldást. És Bill nem vette észre, vagy nem vette komolyan, hogy pöttöm cégek egész serege valami Internettel, vagy mivel játszik. Nem nagy bűn, a CompuServe és az AOL sem viselkedett másként. Ám az Internet növekedése elképesztő sebességgel, a szokásosan magas IT növekedési rátát messze maga mögött hagyva zajlott. Fél év sem kellett hozzá, és a nagyok is sorban behódoltak. Behódoltak, és alattvalóvá váltak. Milyen érzés lehetett Billnek, hogy egy piacon nemhogy nem ő vezet, de senki nem kíváncsi a véleményére? Megmondom: frusztráló. Az Internet volt szinte az egyetlen piac, amit a Microsoft nem tudott learatni, hanem keményen megdolgozott az eredményekért. Keményen küzdött. Évek teltek el olyan küzdelemben, ami erkölcsileg nagyon sokba került a Microsoftnak. Olyannyira, hogy szinte senkinek sem tűntek fel a termékfejlesztésben végbement nagyságrendi pozitív változások. A termékek megbízhatósága hihetetlen fejlődésen ment át! Gondoljunk csak az operációs rendszerekre: Windows 3.1, 95, NT, 2000 – mindegyik fényévekre van az előzőektől!

Öt kemény év telt el ezen a lélektanilag és egyéb tényezők szempontjából is létfontosságú piacon úgy, hogy a Redmondi Óriás hátrányos helyzetben vergődött! Hja, a sok-sok pénz nem minden! Nem tudták, hogy mit akarnak, hiányzott a vízió, hisz Bill a napi papírtologatással volt elfoglalva. Jó, persze lett IIS és Explorer 2.0, 3.0, 4.0 és 5.0, lett Frontpage, save as HTML és hasonlók, sőt megindult, és jelenleg is tart a hosszútávon életképtelen technológiák kiirtása *(MAPI, X.400, SMB, NetBIOS és WINS)*. De semmi új ötlet. Semmi eget rengető idea öt hosszú éve! De most megszületett!

.NET mégegyszer

Ez nem csupán a Windows DNA legújabb ruhája. Ez egy világmegváltó ötlet: nem egyedi PC-k, nem webkiszolgálók, nem harminc éve zárt funkcionalitású szolgáltatások, hanem valami új. Ahol nem a szolgáltatásokat szabványosítják, hanem végre-valahára a fejlesztést, a kapcsolattartást, és az Internet kiterjeszthetőségét. A .NET teszi azzá az Internetet, amire termett: infrastruktúrává. Hogy én mennyire gyűlölöm a regisztrációs lapokat és a jelszavakat! Hoci nekem Microsoft Passport! Hogy mennyi felesleges erőfeszítés megy el arra, hogy minden szerencsétlen pizzafutár cég saját maga fejleszt(tet)i ki halál pontosan ugyanazt a térképadatbázist! Jöjjenek és sokasodjanak a megaszolgáltatások! És mennyit kell kattintgatni! Miért kell a weben bogarásznom ahhoz, hogy a wordben kijelölt szanszkrit nyelvű szó magyarrá váljon? Hol a jobbklatty-translate? Olyan lehetőség manapság egyáltalán nincs, hogy a meglévő alkalmazásaim az Internetről tetszőleges szolgáltató vagány funkcióival kiegészüljenek. Jó, tudom, ActiveX. De nem fogadom el a választ! Hol is van az a szanszkrit-magyar ActiveX? Mi a címe? Miért függ a szolgáltatás megtalálása az én intuitív keresési kulcs-blöffölő képességemtől? És ki fizeti meg a guberálással elpazarolt időmet? Hol a szolgáltatás-lokátor? Lesz!

LINUX

Rengeteg jelenlegi, esetleg tudatosan át sem gondolt problémára hoz megoldást a .NET. A startpisztoly eldördült, az új Internet lehetőségei minden fejlesztő számára adottak lesznek. A .NET egy vezércsillag, és tudom, hogy a Microsoft a megtévelyedett bárányokáknak is lehetővé fogja tenni, hogy kövessék útjukon. Szomorú, de így van, egy csetepatéval kevesebbnek örülhetünk 2001-ben: a LINUX nem beveendő bástya többé, hanem hátasló. Két évet adok Billnek, hogy mindenki megelégedésére a LINUX is .NET hordozóvá váljon. S akkor beköszönt a világbéke, és mindannyian egymás kebelére borulunk, miképpen mi is megbocsátunk az ellenünk vétkezőknek, BUÉK.

Fóti Marcell
marcellf@netacademia.net

2001 Január

tech.net

A Microsoft Magyarország Szakmai Magazinja

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

Kiadja a **Microsoft Magyarország**
1031 Budapest, Graphisoft park 3.
Tel.:437-2800
A kiadásért felel:
Arany Tóth László
v-laarto@microsoft.com

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

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

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Udvarev Rita**
Tel./Fax: 214-0923
velvethammer@ahol.com
1027 Budapest, Fő utca 67. V. 1.
Grafikai tervezés, kivitelezés, nyomdai előkészítés:
Bársonykalapács Marketing
Művészeti vezető: **Balogh Zoltán**

Nyomda:
Partner's 2000 Kft.
1124 Budapest, Sion lépcső 7.
Felelős nyomdász: **Galambos Sándor**

ISSN 1586-5185

Jön az X-Box!

A Microsoft bejelentette az X-Box-ot, a videojátékok újabb generációját

Bill Gates január 6-án, Las Vegas-ban bejelentette a „videojátékok jövőjét", az X-Box-ot. „Az X-Box kimagasló technológiai paramétereivel, hihetetlen grafikai teljesítményével minden kétséget kizáróan a játékkonzolok piacának mércéje lesz az elkövetkezendő években. A játékprogramkészítők a videojátékok új világát fedezhetik fel, s olyan játékélményt nyújthatnak a videojátékok szerelmeseinek, melyeket korábban elképzelni sem lehetett." – mondta Bill Gates.

Az X-Box játékkonzolt, valamint a hozzá tartozó vezérlőegységet több mint 5000 lelkes videojáték-rajongó és játékprogram fejlesztő visszajelzéseinek figyelembe vételével tervezték. A fekete színű készüléken, mely külalakjában is erőt sugároz, egy nagy X betű domborodik, s ennek közepén található a zöld színű X-Box „ékszer".

Az X-Box logóval szintén ellátott vezérlőegység kialakításakor a biztonságos irányítás és kényelmes használat volt a fő szempont. Végső arculata mögött több ezer tesztelési óra munkája rejlik. Egy nyolcutas irányítóegység található benne, bal és jobb analóg karok, hat darab színes, nyomásérzékeny analóg gomb, valamint kettős bemenet a memóriakártya és egyéb perifériák számára.

Bővebb információ: **[1]**

Gyártósoron a Microsoft ISA Server 2000

Microsoft Internet Security & Acceleration Server

Elkészül a .NET kiszolgálócsalád szolgáltatásainak védőbástyáját alkotó Internet Security and Acceleration *(ISA)* Server 2000 végleges verziója, amely már a CD másoló üzemekben jár. Az ISA Server nagyvállalati szintű tűzfal és Web cache kiszolgáló, amely könnyen felügyelhető, biztonságos és gyors Internet elérést biztosít bármilyen méretű cég részére. A korábbi Proxy 2.0 Serverrel ellentétben az ISA már teljesértékű tűzfal megoldást nyújt, többszintű csomagszűrő szolgáltatással, kapcsolat- és alkalmazásszintű szűrési lehetőséggel. Beépített VPN és betörésjelző képességekkel rendelkezik, valamint képes értelmezni és szűrni az elterjedt Internetes protokollokat, így például képes kiszűrni a bizonyos kulcsszavakat tartalmazó elektronikus leveleket.

A többi .NET kiszolgálóhoz hasonlóan kétféle verzióban kapható. Az Enterprise változat a nagy internetforgalmú nagyvállalatok számára ajánlott. Központosított felügyeleti eszközei segítségével még sok telephely esetén is egy számítógépről vezérelhetők az egész szervezetre vagy csoportokra, telephelyekre érvényesíthető biztonsági beállítások. Többszintű házirendek segítségével finoman hangolható az egyes csoportok, felhasználók, alkalmazások internethozzáférése. A nagy rendelkezésre állás érdekében terheléselosztó fürtbe is kapcsolhatók a gépek, így tetszőleges terhelésre skálázható és hibatűrő képességekkel rendelkező tűzfalmegoldást kapunk. Elsősorban üzleti kritikus nagyvállalati környezethez ajánlják.

A Standard Edition az előbbi verzióhoz hasonló képességekkel rendelkezik, a kis és közepes vállalatok számára is elérhető áron. A termék a kereskedelemben 2001 első negyedévétől elérhető. Bővebb információt a **[2]** címen találhatnak az érdeklődők.

Microsoft Certified Professional Systems Engineer

Február 28.-ig még le lehet tenni az NT4 MCP vizsgákat

Az NT4-es vizsgák utolsó napjaként megjelölt 2000. december 31.-i dátumot 2001. február 28.-ára módosította a Microsoft. A lépésre azért volt szükség, mert a 2000. év végén elárasztották a vizsgázni vágyók a vizsgaközpontokat, akik sok jelentkezőt helyhiány miatt egyszerűen nem bírtak regisztrálni a vizsgákra. Volt olyan vizsgaközpont, ami december folyamán éjfélig nyitva tartott, így igyekezve kiszolgálni a nem várt igényt. A nagy vizsgakedv egyik oka, hogy a Windows 2000-es MCSE fokozat megszerzéséhez szükséges 7 vizsga helyett csak 4-et kell letenni azoknak, akiknek van NT4-es MCSE tanúsítványuk, mert a 7-ből 5 vizsgát egy összevont, maratoni vizsgában is letehetik.

Az NT4-es MCSE vizsgák 2001. december 31.-ig érvényesek, tehát aki szeretne élni az összevont 2000-es alapvizsga lehetőségével, annak még az idén meg kell próbálkoznia az összevont *(egyébként ingyenes)* vizsga letételével.

Elkészült a Microsoft XML Parser 3.0

Hosszú várakozás után, az XML fejlesztők nagy örömére elkészült a Microsoft XML Parser *(MSXML)* 3.0-s verziója. Ez a verzió igen jelentős fejlesztés a Windows 2000-ben található 2.5-ös verzióhoz képest. Milyen újításokat találhatunk benne?

- Teljes támogatás a World Wide Web Consortium *(W3C)* Extensible Stylesheet Language Transformations *(XSLT)-hoz* és XML Path Language *(XPath)-hez*
- Teljes COM felület a Simple API for XML *(SAX2)* támogatáshoz, amelyet Visual Basic, Visual C++, ill. scriptnyelvekből könnyen elérhetővé tettek.
- A Document Object Model *(DOM)* és a névtér támogatás továbbfejlesztése
- Kiszolgálóoldali alkalmazások fejlesztéséhez biztonságos XML elérés HTTP protokollon keresztül
- Az XML 1.0 és a névtér 1.0-hoz kapcsolódó tesztfeltételekhez való nagyon jó alkalmazkodás
- Sok hibajavítás és teljesítmény továbbfejlesztés
- A parser komponens, a dokumentáció és az SDK letölthető a **[3]** címről.

Elkészült a 3. javítócsomag az SQL Server 7-es verziójához

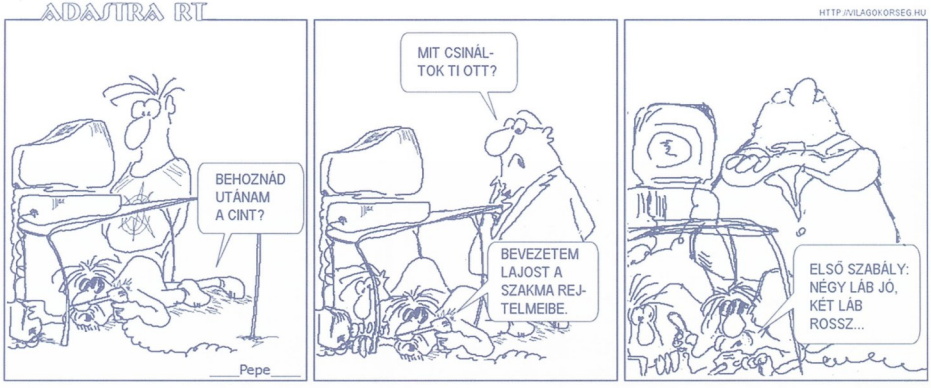
Az új szervizcsomag az SQL Server 7-hez és a Microsoft Data Engine 1.0-hoz nyújt hibajavításokat és fejlesztéseket. Javításokat tartalmaz az adatbázismotorhoz, az Enterprise Manager-hez, valamint az OLEDB és ODBC szolgáltatókhoz. Külön javítócsomag készült az alaptermékhez és az OLAP szolgáltatásokhoz, így fölöslegesen nem kell letölteni azt, amelyiket nem használnák az adott környezetben.
A szervizcsomag által javított hibák leírása valamint a letöltési lehetősége a **[4]** címen érhető el.

Kiadás előtt a Biztalk 2000

A CD gyártósorokon van a .NET kiszolgálócsalád utolsó 2000-es építőkockája, a Biztalk Server 2000. A termék az informatikai piac egyetlen olyan terméke, amely összefogja a nagyvállalati alkalmazásintegrációt *(Enterprise Application Integration, EAI)*, üzlet-üzlet kapcsolatot *(Business-to-business)* és Biztalk Orchestration technológiát, ezzel segítve olyan üzleti folyamatok fejlesztését, amelyek alkalmazásai átfognak több nagyvállalatot is. Nem kevesebbet vállal fel, mint hogy „megoldja a mai IT ipar legnagyobb nehézségét, az elszigetelt és különböző-képpen működő alkalmazások összeintegrálását, egy közös megoldás érdekében".
Az XML-re épülő kiszolgáló kétféle változatban, Enterprise Edition és a Standard Edition kiszerelésben kapható. Az Enterprise változat korlátlan számú külső partner és belső alkalmazás integrációjára használható. Elsősorban nagyvállalatok részére használható ki jól, ahol a nagy rendelkezésre állás érdekében fürtözött módon, valamit a teljesítmény érdekében sokprocesszoros rendszerekre is telepíthető.
A Standard változat kis és közepes cégek számára készült, és maximum 5 külső partner és 5 belső alkalmazás összefogására használható.
A szoftverek 2001. januárjában már a kereskedelemben is elérhetőek lesznek.
További információk: **[5]**

A cikkben használt URL-ek:

[1] http://www.xbox.com/
[2] http://www.microsoft.com/ISAServer/
[3] http://msdn.microsoft.com/xml/general/xmlparser.asp
[4] http://www.microsoft.com/sql/downloads/sp3.htm
[5] http://www.microsoft.com/biztalk



Négy láb, két láb: utalás George Orwell Állatfarm című regényére

Policy sablonfájlok szerkesztése *(II. rész)*

DropDownList

Alkalmazásával egy legördíthető lista jön létre. A lista elemeit a sablonfájlban kell meghatároznunk. A System Policy Editor felületén keresztül tehát nem tudjuk az elemeket megváltoztatni. DROPDOWNLIST esetében az alábbi parancsokat használhatjuk:

- REQUIRED: A beállítás engedélyezésekor kötelező értéket megadni.
- ITEMLIST: Ezzel a paranccsal tudjuk meghatározni a lista elemeit. Az elemek nevét a NAME kulcsszót követően szöveges változóval adjuk meg. Mivel a bejegyzés értéke nem a kiválasztott elem lesz, ezért minden elem után fel kell tüntetnünk a megfelelő értéket is:

```
POLICY !!Sample7
KEYNAME "ADMTeszt\DropDownList"
 PART !!DropDownList_Part          DROPDOWNLIST
   VALUENAME "DropDownList_Value"
      ITEMLIST
        NAME !!Element1 VALUE NUMERIC 10
        NAME !!Element2 VALUE !!Element2
        NAME !!Element3 VALUE NUMERIC 35
DEFAULT
      END ITEMLIST
 END PART
END POLICY
```

A példában a legördíthető listának három eleme lesz. Az első kiválasztásakor a DropDownList_Value néven REG_DWORD típusú bejegyzés jön létre a regisztrációs adatbázisban, melynek értéke 10 lesz. Ha REG_SZ típusú bejegyzést szeretnénk létrehozni, akkor a NUMERIC kulcsszó nélkül kell az értéket feltüntetnünk. Ilyenkor lehetőségünk van az elem nevével megegyező értékű bejegyzést létrehozni a regisztrációs adatbázisban *(lásd a fenti példa második elemét)*. A DEFAULT kulcsszóval meghatározhatjuk, hogy a házirend engedélyezésekor a bejegyzés alapértelmezésben milyen értéket kapjon.

ComboBox

Szintén egy legördíthető lista jelenik meg a Policy Editor felhasználói felületén. Eltérően a DROPDOWNLIST típustól, a listából kiválasztott elem neve lesz a bejegyzés értéke a regisztrációs adatbázisban. Ezen túlmenően a COMBOBOX esetében nemcsak a listából választhatunk, hanem más értéket is beírhatunk. A bejegyzés típusa minden esetben REG_SZ lesz. Az EDITTEXT esetében leírt parancsok a COMBOBOX esetében is használhatóak, de ezeken kívül még két utasítás áll rendelkezésünkre:

- SUGGESTIONS: A listában megjelenő értékek meghatározására szolgál. Az értékeket szóközzel elválasztva kell feltüntetni a sablonfájlban, és csak akkor kell idézőjelbe tenni őket, ha szóköz is szerepel bennük. A felsorolást az END SUGGESTIONS zárja.
- NOSORT: Alapértelmezésben a lista elemei névsorban tűnnek fel. Ezt a parancsot akkor érdemes használni, ha a felsorolásuk sorrendjében szeretnénk megjeleníteni őket.

```
POLICY !!Sample8
KEYNAME "ADMTeszt\ComboBox"
  PART !!ComboBox_Part   COMBOBOX
    VALUENAME "ComboBox_Value"
      SUGGESTIONS
       Egy Hat Tizenegy Kilenc
      END SUGGESTIONS
  END PART
END POLICY
```

Numeric

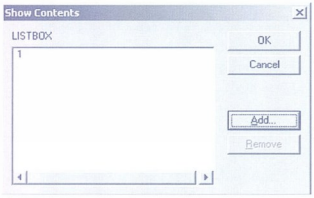
Ezzel a típussal egy szerkeszthető mezőt hozhatunk létre, amelynek értéke csak egész szám lehet. A mezőben szereplő értéket nyilak segítségével növelni és csökkenteni is lehet. A bejegyzés típusa REG_SZ vagy REG_DWORD lesz a regisztrációs adatbázisban. Ennél a típusnál az alábbi parancsok használhatóak:

- MAX <érték>: Meghatározhatjuk, hogy mekkora legyen a mezőbe írható maximális érték. Alapértelmezésben ez a szám 9999.
- MIN <érték>: A mezőbe írható szám legkisebb értékét adhatjuk meg. Alapértelmezésben ez az érték 1.
- SPIN <érték>: Meghatározhatjuk, hogy a nyilakkal mennyivel növelhető, illetve csökkenthető a mezőben szereplő érték. Az utasítás hiányában egyesével változtathatjuk a mezőben szereplő számot. Ha nulla értéket írunk a parancs után, akkor a nyilak nem jelennek meg a mező jobb oldalán.
- DEFAULT <érték>: Megadhatjuk, hogy a beállítás engedélyezésekor milyen érték jelenjen meg a mezőben.
- REQUIRED: Ezzel a paranccsal kötelezővé tehetjük a mező kitöltését.
- TXTCONVERT: A mezőbe továbbra is csak számok írhatóak, de a bejegyzés REG_SZ típusú lesz a regisztrációs adatbázisban.

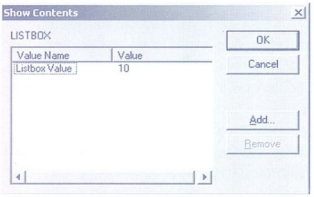
```
POLICY !!Sample9
KEYNAME "ADMTeszt\Numeric"
  PART !!Numeic_Part    NUMERIC   SPIN 5
    VALUENAME "Numeric_Value"
      DEFAULT 20
      MIN 0
      MAX 30
  END PART
END POLICY
```

ListBox

Ez a PART típus lehetőséget ad számunkra, hogy egy kulcs alatt több bejegyzést hozzunk létre, illetve változtassunk meg a felhasználói felületen keresztül. Korábban már szóba került, hogy a LISTBOX esetében nem a VALUENAME kulcsszóval határozzuk meg a bejegyzés nevét. Erre két különböző lehetőség is kínálkozik. Létrehozhatunk olyan LISTBOX-ot, ahol csak a bejegyzés értékét kell megadnunk, az elnevezést a Policy Editor automatikusan rendeli hozzá az értékhez:



A LISTBOX másik típusánál viszont az érték mellett a bejegyzés nevét is meg kell határoznunk:



A regisztrációs adatbázisban mindkét esetben REG_SZ típusú bejegyzések jönnek létre. A LISTBOX esetében a következő parancsok használhatóak:

- VALUEPREFIX <előtag>: Az előtag a bejegyzés nevének meghatározására szolgál. A System Policy Editor automatikusan egy sorszámot fűz az előtagként meghatározott karaktersor után, attól függően, hogy hányadik helyen szerepel az adott bejegyzés a LISTBOX-ban. Például VALUEPREFIX ValueName esetén a regisztrációs adatbázisban ValueName1, ValueName2, ValueName3 stb. néven jönnek létre a bejegyzések. „Üres" karaktersort *(„")* is megadhatunk előtagként. Ilyenkor bejegyzések elnevezése „1", „2", „3" stb. lesz.
- EXPLICITVALUE: Ennek az utasításnak a használatakor nemcsak a bejegyzés értékét, hanem a nevét is meg kell adni a felhasználói felületen keresztül. Nem használható a VALUEPREFIX utasítással együtt.
- ADDITIVE: Alapértelmezésben a regisztrációs adatbázis megfelelő kulcsa alatt csak a LISTBOX-ban megadott bejegyzések fognak szerepelni, a házirend végrehajtása során az összes korábbi kitörlődik. Az ADDITIVE parancs használatával a más forrásból származó bejegyzések megmaradnak.

```
POLICY !!Sample10
KEYNAME "ADMTeszt\ListBox"
  PART !!ListBox_Part    LISTBOX
    EXPLICITVALUE
  END PART
END POLICY
```

Text

Ez a PART típus arra szolgál, hogy az egyes beállítások használatához magyarázó szöveget tudjunk megjeleníteni a Policy Editor felhasználói felületén. Alkalmazásakor tehát a regisztrációs adatbázisban nem keletkezik bejegyzés. Használata nagyon egyszerű. A PART kulcsszó után lévő szöveges változó értéke lesz a magyarázó szöveg. Természetesen a szöveges változó értékét a korábbiakhoz hasonlóan most is a `[strings]` szekcióban határozzuk meg.

```
POLICY !!Sample11
  PART !!Part1 EDITTEXT
    VALUENAME "Sample11_Value"
    MAXLEN 6
  END PART

  PART !!EditText_Tip1  TEXT
  END PART
  PART !!EditText_Tip2  TEXT
  END PART
END POLICY

[strings]

EditText_Tip1 = "A mezőbe maximum hat karakter
irható"
EditText_Tip2 = "Példa a Text Part típusra"
```

Hogyan fogjunk hozzá az új sablonfájl készítéséhez?

Új sablonfájl elkészítésének gondolata általában akkor merül fel, amikor találunk egy olyan beállítást, amit szeretnénk központilag kezelni, de még nem szerepel a házirendben. Legfontosabb lépés, hogy megtaláljuk a regisztrációs adatbázisban a megfelelő bejegyzést vagy bejegyzéseket, és kinyomozzuk, hogy mikor, milyen értékeket vehet fel. Miután végeztünk ezzel a sok esetben időigényes feladattal, akkor elkezdhetjük a regisztrációs adatbázis változásait átalakítani olyan formába, hogy a sablonfájl szintaktikájának megfeleljen. Előfordulhat azonban olyan eset is, hogy valamelyik meglévő házirend beállítást szeretnénk átalakítani. Elkerülendő, hogy a kliensek működésében véletlen hiba folytán zavar keletkezzen, melegen ajánlott a sablonfájlt tesztrendszerben kipróbálni. Csak akkor használjuk éles rendszerben, ha meggyőződtünk róla, hogy minden esetben jól működik. Sok felesleges munkától kímélhetjük meg magunkat, ha a változtatásainkat gondosan letesztelejük.

A sablonfájlok szerkesztésére bármilyen text editor *(például NotePad)* alkalmas. A szintaktikai hibák megtalálásához azonban nem árt, ha van kéznél olyan szerkesztő program, amely kijelzi a sorok számát. A sablonfájl betöltésekor a Policy Editor ellenőrzi, hogy az utasításokat megfelelő helyen és módon használtuk-e. Amennyiben hibát talál, akkor egy rövid üzenetben tájékoztat, hogy melyik sorban bukkant rendellenességre:

A regisztrációs adatbázisban természetesen nem mindegyik bejegyzés érvényes a különböző Windows operációs rendszereknél. Ha a számítástechnikai rendszerünk ebből a szempontból nem egységes, akkor gondolnunk kell arra is, hogy például a Windows 2000-re vonatkozó bejegyzés ne kerüljön mondjuk a Windows NT 4.0 vagy a Windows 98 operációs rendszeren futó számítógép regisztrációs adatbázisába. Ezt a problémát az #if version, #endif utasításokkal lehet megoldani:

```
#if version <= 2
CLASS User
  CATEGORY !!W2K
      POLICY !!GPADM
        KEYNAME "AdmTeszt"

        PART !!Msg1     TEXT
        END PART

      END POLICY
  END CATEGORY
#endif

#if version >= 3
CLASS User
  CATEGORY !!GPTest
   KEYNAME "AdmTeszt"
      POLICY !!Sample12
       VALUENAME "if_version"
      END POLICY
  END CATEGORY
#endif

[strings]

W2K = "Windows 2000 operációs rendszerhez"
GPADM = "Group Policy sablon"
Msg1 = "Ez a sablon nem használható System Policy
↳ Editorral"
GPTest = "Group Policy teszt"
Sample12 = "Példa_12"
```

Ha a fenti példában szereplő sablonfájl mintát a System Policy Editorral töltjük be, akkor az #if version <= 2, #endif által határolt rész jelenik meg. Így a felhasználói felületen csak egy figyelmeztető üzenet tűnik fel. A csoportos házirend esetében viszont az #if version >= 3 utáni konkrét beállítást tartalmazó szakasz lép életbe. Ezáltal elkerülhető tehát, hogy a Windows 2000 előtti operációs rendszerekkel működő számítógépekhez olyan beállítás jusson el, amelynek semmilyen hatása sincs a működésükre.

Házirend beállítás átalakítása egy konkrét példán keresztül

A magasabb szintű biztonsági beállítások tervezésekor felmerülhet a programfuttatás korlátozásának igénye. A System Policy révén beépített eszközökkel, központilag tudjuk megadni, hogy a felhasználók mely programfájlokat indíthatják el. A beállítás alkalmazásakor több probléma is felmerül. Egyrészt a korlátozás megkerülhető. Például command promptból *(cmd.exe)* vagy az Office makróból indított programokra nem vonatkozik a tiltás. A felhasználói jogok további korlátozásával azonban lényegesen megnehezíthetjük a szabály kijátszhatóságát. Másrészt a beállítás működéséből fakadóan nagymértékben megnőhetnek a rendszergazdai feladatok. Számos programot mindenki számára hozzáférhetővé kell tenni, bizonyos alkalmazásokat viszont csak néhány dolgozó használhat. Ez utóbbi programok engedélyezését csoportok szintjén érdemes szabályozni. Ennek értelmében minden általánosan nem használt alkalmazáshoz létre kell hoznunk egy globális csoportot. Természetesen a házirendben minden csoportnak csak a hozzá tartozó programfájlok futtatását engedélyezzük. A gondok valójában akkor kezdődnek, amikor egy dolgozó két csoportnak is tagja lesz. A házirend végrehajtása során azt fogjuk tapasztalni, hogy a felhasználó csak az egyik csoportnak engedélyezett programfájlokat tudja elindítani. Mivel a két csoportban ugyanahhoz a beállításhoz eltérő értékeket adtunk meg, ezért a System Policy csak a rangsorban előrébb lévő csoportét veszi figyelembe. *(A System Policy Editor működésének ismertetésekor már volt róla szó, hogy a sorrendet az Options menü Group Priority menüpont alól elérhető párbeszédablakban határozhatjuk meg.)* Kézenfekvő megoldásnak tűnik, hogy egy felhasználó csak egy csoportnak legyen a tagja, de ezáltal lényegesen több csoportot kell létrehozni, ami megnehezíti a rendszergazdák munkáját. A sablonfájl átalakításával azonban megoldható, hogy a System Policy fésülje össze az egyes csoportoknak megadott értékeket.

A programok futtatásának korlátozása a common.adm nevű sablonban található:

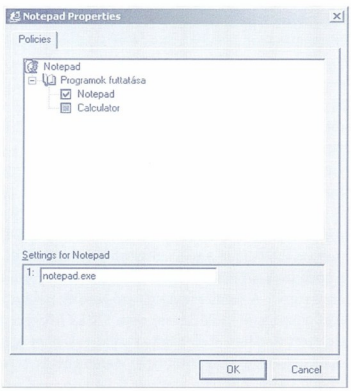
```
POLICY !!RestrictApps
KEYNAME
Software\Microsoft\Windows\CurrentVersion\Policies
↳ \Explorer
  VALUENAME RestrictRun
     PART !!RestrictAppsList LISTBOX
     KEYNAME
Software\Microsoft\Windows\CurrentVersion\Policies
↳ \Explorer\RestrictRun
        VALUEPREFIX ""
     END PART
END POLICY
```

A korlátozás engedélyezésével az Explorer kulcs alatti RestrictRun nevű bejegyzés értéke 1 lesz. Az engedélyezett programok neve az Explorer\RestrictRun kulcs alatt található. A VALUEPREFIX „" utasítás miatt a RestrictRun kulcs alatt „1", „2", „3" stb. néven jönnek létre a bejegyzések. Az egyszerűség kedvéért tegyük fel, hogy a Notepad, valamint a Calculator programok elindítását szeretnénk engedélyezni. Vannak olyan felhasználók, akik csak az egyik, mások mindkét programot használhatják. A beállítások érvény-

re jutásának szabályszerűségeit ismerve az alábbi problémákat kell megoldanunk:
Biztosítanunk kell, hogy a két programhoz tartozó bejegyzés semmilyen körülmények között ne íródjon ugyanolyan néven a regisztrációs adatbázisba. Ezáltal elkerülhető, hogy a házirend végrehajtásakor valamelyik bejegyzés felülíródjon.
Mivel azok a beállítások, amelyek nem fedik át egymást, a csoportok szintjén összeadódnak, a sablonfájlt úgy kell megszerkesztenünk, hogy mindegyik program futtatásának engedélyezéséért külön beállítás feleljen.

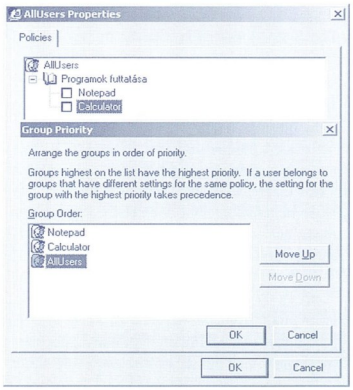
```
CATEGORY !!RestApps
KEYNAME "ADMTeszt\SampleApps"

  POLICY !!Notepad
    PART !!1    EDITTEXT
      VALUENAME "1"
    END PART
  END POLICY

  POLICY !!Calculator
    PART!!1     EDITTEXT
      VALUENAME "5"
    END PART
  END POLICY
END CATEGORY
```

Ezzel a megoldással a korábban megfogalmazott mindkét feltételt sikerült teljesítenünk. Egyrészt a Notepad és a Calculator futását külön jelölőnégyzettel tudjuk engedélyezni. Másrészt mindegyik program önálló bejegyzésnévvel rendelkezik. Mivel erre a feladatra a LISTBOX nem alkalmas, ezért kellett az EDITTEXT típust alkalmaznunk. A házirend beállításakor ügyelnünk kell arra, hogy a Notepad csoportnál a Calculatorhoz tartozó jelölőnégyzetet hagyjuk szürkén *(nem konfigurált állapot)*. Természetesen ugyanez fordítva is igaz.



Programfuttatás engedélyezése

A System Policy működésének sajátosságából fakadóan még egy problémát kell leküzdenünk. Ha az egyik felhasználót kivesszük mondjuk a Notepad csoportból, akkor tapasztalni fogjuk, hogy a programhoz tartozó bejegyzés nem törlődik a regisztrációs adatbázisból. Ezt a gondot a következőképpen orvosolhatjuk. Szükségünk lesz egy csoportra *(például AllUsers)*, amelyiknek minden érintett felhasználó a tagja. Ebben a csoportban az összes programfuttatásra vonatkozó beállítást le kell tiltanunk.
Ezen túlmenően a csoportok sorrendjében ennek a csoportnak kell a leghátul állnia. Így, ha egy felhasználót kiveszünk például a Calculator csoportból, akkor csak egyetlen olyan csoportnak *(AllUsers)* lesz tagja, ahol a Calculator jelölőnégyzete nem szürke. Így a házirend végrehajtása során a beállítás letiltása lép életbe, vagyis a programhoz tartozó bejegyzés kitörlődik a regisztrációs adatbázisból.



Végezetül fontos megjegyezni: a gyakorlati tapasztalatok azt mutatják, hogy csak akkor működik jól ez az átalakítás, ha a beállításokat egy újonnan létrehozott ntconfig.pol fájlba mentjük el.

Tomasz Balázs,
balazs.tomasz@hu.hypovereinsbank.com

Microsoft Network Monitor *(II. rész)*

Mire e cikk megjelenik, a karácsony már a múlté lesz. Mindenki lázasan püföli munkahelyén a billentyűket, és lassan ismét elfelejti milyen is az élet számítógépek nélkül. Pedig a karácsony sokunk számára a billentyűzet elengedésének ünnepe is egyben! Ha pedig az emberre rátör a kóros billentyűzhetnék, akkor sem jut hozzá, hisz – ugyebár – számítógép nincs otthon. Rajtam valami olyan elemi erővel lett úrrá a billentyűzhetnék, hogy kénytelen voltam kicsomagolni azt a hazahozott laptopot, amit suttyomban pont azért csempésztem haza, hogy ha jelentkeznek az elvonási tünetek, ne kelljen orvoshoz fordulnom. Így esett, hogy szent karácsony harmadik napján feladtam a küzdelmet, és a hosszú elvonókúra után elkékült arccal, remegő inakkal billentyűzetet ragadtam, hogy megírjam ezt a cikket a Network Monitorról, mely önmaga is egy narkó. Mármint a NetMon. Nem hagyja az embert békében pihenni, keretekkel és csomagokkal tömi meg a gyanútlan áldozat fejét, aki a világot is üzenetszórásos csatornának látja, és maximum 1514 bájtos csomagokban hajlandó felvinni az emeletre a karácsonyi bevásárlás súlyos, ám - csakúgy mint az Ethernet kártya számára – értelmetlen terhét.

Megfejtés

A múlt hónapban egy rejtvénnyel fejeztem be a cikk első részét, mely így hangzott: mi a PING által szállított „hasznos" adat? Azoknak helyes a megfejtése, akik az angol ABC betűit találták meg a „number of data bytes remaining" szekcióban.

```
ICMP: Packet Type = Echo
ICMP: Echo Code = 0 (0x0)
ICMP: Checksum = 0x3A5C
ICMP: Identifier = 512 (0x200)
ICMP: Sequence Number = 4352 (0x1100)
ICMP: Data: Number of data bytes remaining = 32 (0x0020)

00000000  00 A0 CC C0 71 18 00 20 18 A1 B9 D2 08 00 45 00  .à̀Àq.. ¡¹Ò..E.
00000010  00 3C E4 5A 00 00 80 01 FE 41 AC 10 00 01 AC 10  .<äZ..€.þA¬...¬.
00000020  00 03 08 00 3A 5C 02 00 11 00 61 62 63 64 65 66  ....:\....abcdef
00000030  67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76  ghijklmnopqrstuv
00000040  77 61 62 63 64 65 66 67 68 69                    wabcdefghi
```

A PING „hasznos" terhe

Ez alapértelmezésben 32 bájtnyi ABC. Ha a PING parancsot –l kapcsolóval indítjuk, sokkal precízebben meg tudjuk adni, hogy mennyi legyen a hasznos teher. Minimális értéke 1, maximuma 65500. Az jó sok. Hallottak már a halálos PING-ről? Régi tréfa, korábbi operációs rendszerek elleni méreg. Hogyan is működik a PING? A kezdeményező gép elküld egy bájtsorozatot a célgépnek *(ECHO)*, amelyik ezt hibátlanul megismételve visszaküldi *(ECHO REPLY)*. Ahhoz, hogy meg tudja ismételni, a beérkező „hasznos" adatokat rögzítenie kell, el kell tárolnia a memóriában. Tipikus biztonsági rés, amikor egy protokoll megvalósításánál a programozó nem kezeli le az összes előforduló esetet, csak azokat, amelyeket a szabványban is leírtak – vagy még azokat sem. Nos, a korai ICMP ECHO-k szinte mindegyike „megvalósította" ezt a hibát, még a büszke Linuxok is. Valahogy úgy kell elképzelni, hogy a programozó lefoglalt az ECHO számára egy meghatározott méretű puffert *(mondjuk 10000 bájtot, az nem olyan sok)* és ezzel a dolog el is van intézve. Szépen működik is a pingecske mindaddig, amíg a „hasznos" teher meg nem haladja a lefoglalt puffer méretét. De amint túllépi a bűvös határt, a többletkarakterek már nem a lefoglalt pufferbe kerülnek, hanem valamilyen más memóriaterületre – ami a kernel mód szabályai szerint akármi is lehetett. De messzire elkalandoztam *(hiába, ez az ünnepek hatása)* a halálos PING már nem létezik! Már csak a legbénább hackerek próbálkoznak vele, de azokat is megfogja minden tűzfal, beleértve az ISA Servert is.

PING

Hanem vizsgáljuk meg a PING forgalmát tetőtől talpig, hisz egy csomó olyan dolgot tartalmaz ez a néhány csomag, ami minden hibakeresést megkönnyíthet.

Terminológia I.

Cikkemben teljesen vegyesen használom a hálózatról elkapott adatokra a keret és csomag szavakat. Valójában mindig ugyanarról beszélek. A megkülönböztetés szőrszálhasogatás esetén fontos lehet, de itt nem ezzel foglalkozunk. Történelmi okokból ugyanis ugyanazt az adathalmazt más-más névvel illetjük a rétegzett hálózati architektúra különböző szintjein:

Ehternet szinten keretről beszélünk *(frame)*
IP szinten csomagról *(datagram)*
TCP szinten pedig szegmensről *(segment)*
De én már csak maradok az összevissza keverésnél...

Felmerülhet a kérdés, hogy vajon miért e körül a 8-10, viszonylag egyszerű csomag körül ólálkodunk, miért nem ugrunk bele valami kőkeménybe, például az NTLM hálózati forgalmába? Azért, mert ez a néhány csomag olyan, mint cigarettahamu a gyilkosság helyszínén: Sherlock Holmes nem rohan el ilyen fontos bizonyíték mellett anélkül, hogy meg ne vizsgálná, és következtetéseket vonna le! Ebben a pár bájtban választ kapunk az élet néhány alapvető kérdésére is: vajon miért nem a www.microsoft.com gép MAC Addresse található a feladott Ethernet csomagban, amikor odaböngészünk? A hamu megtalálható lapunk webhelyén a januári előzetesről szóló lapon, neve PING.CAP, és akármelyik telepített NetMonnal megnyitva minden kedves olvasó pontosan ugyanazokat a kereteket, méreteket és sorszámokat láthatja, mint ami itt a cikkben is látható.

ARP, ARP, ARP

Kezdjük az ARP-pal, ezzel a kérdezz-felelekkel:

```
2       1.752293    CIS TEA1B9D2  *BROADCAST    ARP_RARP
                    ARP: Request, Target IP: 172.16.0.3
3       1.752293    LOCAL         CIS TEA1B9D2  ARP_RARP
                    ARP: Reply, Target IP: 172.16.0.1 Target Hd.
```

ARP kérés és válasz

Ami magyarul így hangzana:
– Fiúk, melyikőtöknek az IP címe 172.16.0.3?
– Az enyém! Itt a MAC Addressem!

Itt leolvasható, hogy a kérés a CIS TE-A1B9D2-ként azonosított hálózati kártyáról indul ki, és broadcast címzést használ, és a 172.16.0.3 IP című gépet keressük a magyar határban. Ebből következik, hogy e kérést az összes, ezen a szegmensen található gép értelmezni fogja, és a kérést feldobja operációs rendszerének, hogy a magasabb intelligencia válaszolhassa meg a kérdést: az IP cím az adott gépé. Erre az én gépem *(LOCAL)* válaszol, mivel övé a keresett IP cím, de már nem broadcasttal, mert a kérésből kiássa a feladó MAC Addressét, és közvetlenül annak válaszol.
Kevesen tudják, hogy az ARP tetszőleges hardvercímet *(tehát nemcsak Ethernet, hanem egyéb címeket is)* fordít tetszőleges „magasabb" címre. Erre ékes bizonyíték az ARP csomag belsejében található:

```
ARP_RARP: Hardware Type = Ethernet (10Mb)
ARP_RARP: Protocol Type = 2048 (0x800)
ARP_RARP: Hardware Address Length = 6 (0x6)
ARP_RARP: Protocol Address Length = 4 (0x4)
ARP_RARP: Opcode = Request
ARP_RARP: Sender's Hardware Address = 002018A1B9D2
ARP_RARP: Sender's Protocol Address = 172.16.0.1
ARP_RARP: Target's Hardware Address = 000000000000
ARP_RARP: Target's Protocol Address = 172.16.0.3
```

A kétbájtos Hardware Type mező mutatja, hogy itt a 10 megabites Ethernet bizony csak egy aleset, egy a sok létező fizikai réteg közül, melynek fizikai címei éppenséggel hat bájtosak *(Hardware Address Length = 6)*. Ennél is érdekfeszítőbb ugyanakkor, hogy az IP is csak egy az ARP által támogatott sok protokoll közül - melynek négy bájtosak a címei *(Protocol Address Length = 4)*. Ugyanez az ARP változtatás nélkül ki tudná szolgálni az Ipv6-ot Token Ringen!
Az Opcode = Request jelzi, hogy ez egy ARP kérdés, ennek párja az Opcode = Reply a következő csomagban.
S ezután következik a feladó hardver- és protokollcíme *(MAC Address és IP cím)*, majd a kérdéses gép hardvercíme üresen marad *(hisz fogalmunk sincs, erre irányul a kérdés)*, s végül a célgép IP címe olvasható.
Kristálytiszta?
Nem tűnnek fel a kakukktojások?

1. kakukktojás

Mivégre szerepel a feladó saját MAC Addresse az ARP kérésben, hiszen ugyanez az adat az Ethernet keretben is szerepelt? Minek küldi át a CIS TE-A1B9D2 kétszer ezt az adatot? Lehetséges válaszok:

A) Mert az ARP protokoll írói nem tudhatták előre, hogy a fizikai réteg is hordozni fogja ugyanezt az adatot. Emiatt a költségesebb, de előrelátóbb megismétlés mellett döntöttek.
B) Mert bár ott van a MAC Address az Ethernet keretben, az a célgép „agyáig" nem jut el, mert a rétegezett hálózati architektúrának megfelelően minden magasabb szintű réteg csak az alatta lévő réteg által cipelt adatot kapja meg *(emlékezzünk: „number of data bytes remaining")*, így az a MAC Address nem jut fel – az Ethernet kártya meghajtója éppúgy lefejti és lenyeli, mint például a célgép hardvercímét.
C) Mert az ARP protokollt a Microsoft írta, aki nincs jóban a Xerox-szal, aki pedig az Ethernet szabvány ura.

Ugye milyen csábító az A válasz? Nem-nem! B!

2. kakukktojás

És mit keres ott a kérdező MAC Addresse? Kit érdekel? Hisz a célgép MAC Addressére kérdezünk!

A) Azért van ott, mert az ARP után roppant nagy valószínűséggel kétirányú adatátvitel következik a feladó és a cél között, s ez biztosítja, hogy nem csak a kérdező fogja megtudni a célgép hardvercímét, hanem a célgép is tudomást szerez a kezdeményező címéről – mégpedig nulla darab hálózati csomag elküldésével. Ez az adat „csak úgy" bejön!
B) A kérdező MAC Addresse a szimmetria miatt van ott, az ARP tervezői művészlelkek voltak, és harmóniára törekedtek.
C) Ez arra szolgál, hogy ha már az IP cím beállításánál nem derült ki az esetleges címütközés, akkor majd most, ezzel a broadcast csomaggal felszínre kerül. Hisz itt bekiabáljuk a hálózatba saját IP címünket és MAC Addressünket, mely minden géphez eljut, s ha valamelyik esetleg ugyanezt az IP címet kapta, most észbekaphat.

No nézzük! A **B** válasz komplett hülyeség. Az **A** itt is csábító, és igaz is! Erről meggyőződhetünk az ARP gyorsítótárak kilistázásával: noha a pingfogadó gépen a kisujjunkat sem mozdítottuk, az ARP – a feltárja, hogy a cache bizony gyarapodott! És a **C**? Mi az, hogy egy IP cím ütközés nem derül ki? A **C** válasz helyes, de marad a rejtélyes kérdés: hogyan fordulhat elő, hogy két gépen azonos IP címet állítunk be, és az csak percekkel/órákkal később derül ki?
A megfejtéseket levelezési listáinkon várjuk:

1. feliratkozás a **[1]** címre írott üres levéllel
2. Megfejtés beküldése a **[2]** címre írott, a megfejtést tartalmazó levéllel.

Beküldési határidő nincs, ehelyett szorgos feliratkozás van. Honnan fogja jó előre megtudni a következő szám részletes tartalmát, ha nem iratkozik fel?
Egyébként lassacskán mindent tudunk az ARP-ról. Talán még annyit tennék hozzá, hogy az ARP dolgozik az IP cím ütközések azonnali kiderítésénél *(ami NEM a rejtvény tárgya!)* oly módon, hogy minden gép, amelyik új IP címet kap, „megarpolja" azt mielőtt véglegesítené. Ilyenkor az ARP így hangzik:
– Fiúk, melyikőtöknek az IP címe az én jövendőbeli IP címem?
S ha erre jön válasz, akkor a cím foglalt. Ezt az ARP-ot Gratious ARP-nak hívják, és ezen kívül még egy szerepe van: mivel a fenti kiáltás broadcast, minden géphez eljut, és így

minden gép kijavíthatja az ARP gyorsítótárába esetleg beszorult korábbi értékeket. Az ARP tehát a Microsoft Cluster Server működésének alapja. A következtetés talán egy kicsit hirtelennek tűnik, de így van, ARP nélkül nincs fürtözés. Itt és most nem fejtem ki részletesebben, mert korábban, más újság hasábjain már megtettem.

IP

S most térjünk át a következő néhány csomagra, a PING érdemi részére! A harmadik keret tartalma a következő:

```
+Frame: Base frame properties
+ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD
+IP: ID = 0xE45A; Proto = ICMP; Len: 60
+ICMP: Echo: From 172.16.00.01 To 172.16.00.03
```

Ethernetben IP-ben ICMP. Az Ethernet réteget ismertnek veszem, abban ugyanis továbbra sincsen más, mint a feladó és a célgép hardvercíme és a Frame Type, mely megmutatja, hogy az Ethernetben IP utazik *(0x800)*. Az IP így néz ki részletesen:

```
IP: Version = 4 (0x4)
IP: Header Length = 20 (0x14)
IP: Precedence = Routine
IP: Type of Service = Normal Service
IP: Total Length = 60 (0x3C)
IP: Identification = 58458 (0xE45A)
IP: Flags Summary = 0 (0x0)
    IP: .......0 = Last fragment in datagram
    IP: ......0. = May fragment datagram if
↳      necessary
IP: Fragment Offset = 0 (0x0) bytes
IP: Time to Live = 128 (0x80)
IP: Protocol = ICMP - Internet Control Message
IP: Checksum = 0xFE41
IP: Source Address = 172.16.0.1
IP: Destination Address = 172.16.0.3
IP: Data: Number of data bytes remaining = 40
↳  (0x0028)
```

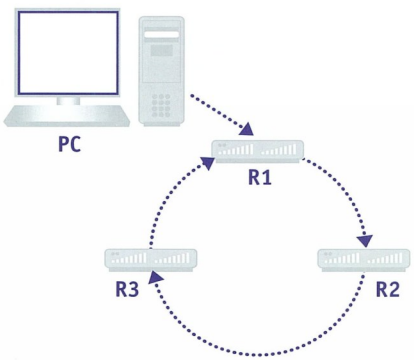
A legelső bájt az IP verziószámát *(felső 4 bit)* és az IP fejléc hosszát *(alsó 4 bit)* adja meg. A verziószám azért áll legelöl, mert ez alapján derül ki, hogy a többi mezőt hogyan kell értelmezni. Ennél kisebb verziószámmal nem találkozunk, de nagyobbal se. A négyest egyébként nem az ötös követi, hanem a hatos, ha elterjed. A következő bájt a Type of Service és Precedence értékeket hordozza. Ennek a bájtnak a 3-5. bitjei a csomag által igényelt késleltetés *(delay)*, megbízhatóság *(reliability)* és átbocsájtóképesség *(throughput)*.
A Total Length az IP és az által hordott hasznos adat teljes összege bájtokban, azaz

Total Length= `Header Length+ Number of data bytes remaining`

Az Identification egy eléggé kifinomulatlan eljárás az ideoda küldött IP csomagok helyes sorrendjének meghatározására. Ne tévedjünk, ez a növekvő sorszám NEM tesz lehetővé hibajavítást azon a szinten, ahogyan majd a TCP dolgozik, sokkal inkább a töredezett IP csomagok összeállításában segédkezik. A következő két bájt *(Flags Summary és Fragment Offset)* a csomag töredezettségével foglalkozik. A mi csomagunk nem töredezett, hurrá. Ezért most ne is foglalkozzunk ezzel részletesen.

Time to Live

Annál fontosabb viszont a TTL *(Time to Live)* érték. Feltételezve, hogy minden olvasóm enyhén jártas az IP útválasztás alaptudományában, minden különösebb magyarázat nélkül tekintsük meg az alábbi hálózatot, ahol R1, R2 és R3 útválasztók *(routerek)* úgy vannak beállítva, hogy a nyílhegy felé mutat az alapértelmezett útvonaluk *(Default Gateway)*:



Legyen továbbá egy PC-nk *(Default Gateway=Router1)*, amelyről megpingetünk egy olyan címet, amihez még csak hasonló sincs ebben a hálózatban. Mit tesz R1 az ismeretlen című csomaggal? R2-nek továbbítja. R2 ugyanígy tanácstalan, ezért R3-nak küldi, aki szintén nem áll a helyzet magaslatán, ezért átdobja R1-nek. S a kör bezárult. Vajon meddig keringene egy hibás csomag ebben a hálózatban? Úgy van, a végtelenségig – hacsak...! Ha csak észre nem vesszük, hogy kering, és ki nem vesszük a hálózatból. Erre való a TTL. Az eredeti szabvány szerint a csomag maximális élettartamát adja meg másodpercben. A mai routerek azonban x-szer gyorsabban dolgoznak elődeiknél, így a legújabb RFC szerint a TTL nem másodpercben értendő, hanem ennyi darab útválasztón juthat át a csomag. Minden router eggyel csökkenti a rajta áthaladó csomagok TTL-jét, és ha ennek értéke eléri a nullát, akkor a csomagnak vége. Intelligensebb útválasztók ilyenkor visszaszólnak az eredeti feladónak egy ICMP Time Exceeded üzenettel, hogy értesüljön a katasztrófáról, és újra tudjon próbálkozni – most már megnövelt TTL-lel.

Protocol

A következő bájt a Protocol mező. Ez határozza meg, hogy az IP mit cipel. A 0x1 jelentése: ICMP. Itt álljunk meg egy szóra. Megfigyelhető, hogy az IP tudja, mit cipel, miközben korábban arról volt szó, hogy számára értelmezetlen a benne található adat. Nem szabad összetéveszteni az adatértelmezést a továbbítást segítő Protocol mező szerepével. Az IP nem azt tudja, hogy PING-et visz a hátán, hanem azt, hogy az általa hordozott adatot a 0x1 azonosítószámú magasabb rétegnek kell továbbadnia – s ez valóban az ICMP. E mező feladata tehát a bejövő csomag útvonalának determinisztikussá tétele, hogy ne kelljen találgatni, hogy ami

beérkezett, az ízlik-e az ICMP-nek, vagy nem, esetleg a TCP fogyasztja majd el, vagy a GRE. Bele van hát írva, hogy hova kell továbbdobni. Ugyanez az elv lejjebb és feljebb is megfigyelhető. Lejjebb, az Ethernet szintjén a Frame Type mező szolgált ugyanerre a célra, hogy ne kelljen találgatni, hogy vajon a beérkezett keret IPX, vagy NetBEUI. Feljebb, a TCP szintjén a portszámok osztályoznak. 80-as? WWW! 25-ös? SMTP! 110-es? POP3!

Terminológia II.

Ahogy a csomag elnevezése is változik rétegről rétegre, ugyanígy az osztályozást segítő mező neve is.
Történelmi okokból ugyanis ugyanazt a mezőt más-más névvel illetjük a rétegzett hálózati architektúra különböző szintjein:
Ehternet szinten kerettípusról beszélünk *(frame type)*
IP szinten protokollról *(Protocol mező)*
TCP szinten pedig portról
Ezt sajnos nem keverhetem össze-vissza, mert a kifejezések *(különösen a port)* hétköznapi szavakká váltak nyelvünkben...

Checksum

Nini, itt is egy CRC! Az Ethernetnek is volt egy! Az igaz, hogy, azt nem látjuk a NetMonnal, mert ha az alsó CRC rossz, a keret nem jut elénk. De akkor minek még egy ellenőrzőösszeg? Azért, mert egyáltalán nem biztos, hogy az IP csomag CRC-vel védett keretben utazik, s biztos ami biztos, itt is van egy, hogy a szoftveresek is örülhessenek. Igen ám, de van itt egy kis baj, kalamajka.

- Amikor egy IP csomag áthalad egy útválasztón, csökken a TTL.
- Ha csökken a TTL, elromlik az ellenőrzőösszeg.
- Ha elromlik az ellenőrzőösszeg, hibássá válik a csomag.

Ez nem történhet meg!
Az útválasztók fontos feladata, hogy a TTL csökkentése után újraszámítsák a CRC-t, és hibátlan IP csomagot küldjenek tovább.

IP címek

És végül az IP fejlécben olvasható a feladó és a célgép IP címe. A célgép címe az odaút kiválasztásához kell, a feladóé pedig a visszaút meghatározásában játszik majd szerepet. E két cím az, mely végponttól végpontig végigkíséri a csomagot hosszú útján – hacsak proxy vagy tűzfal nem állja útját, amely galád módon átirkálhatja mindkettőt. Ezek a címek utazzák végig a teljes útvonalat, ellentétben a MAC Addressel, mely csak egy adott szegmensen él. Az Ethernet rétegbéli MAC Address értelemszerűen nem lehet a nagyon távoli *(például www.microsoft.com)* gép kártyájának a címe, hisz

1. semmi garancia nincs arra, hogy a célgép egyáltalán Etherneten van. Lehet telefonvonalon, műholdon, mikrohullámon, infrán, telepátián és BlueToothon.
2. Az Ethernet réteg feladata NEM a végcél, hanem a kijárat megcímzése, azaz messzire menő csomagoknál a tőlem elindított Ethernet keretben a Default Gateway MAC Addresse, míg IP fejlécében a távoli gép IP címe szerepel!

ICMP

```
ICMP: Packet Type = Echo
ICMP: Echo Code = 0 (0x0)
ICMP: Checksum = 0x3A5C
ICMP: Identifier = 512 (0x200)
ICMP: Sequence Number = 4352 (0x1100)
ICMP: Data: Number of data bytes remaining = 32
      (0x0020)
```

A Packet Type mutatja meg, hogy a negyedik keret egy kérdés *(ECHO=0x08)*, az ötödik pedig válasz *(ECHO REPLY=0)*. Jön még egy checksum. A TCP/IP üldözési mániában szenved. A Sequence Number minden kibocsátott ECHO-nál egyedi érték, amit a válaszban *(ECHO REPLY)* meg kell ismételnie a felelőnek, hogy a kérdező oldalán egyértelműen meg lehessen állapítani, hogy melyik csomag érkezett vissza. Ez ahhoz szükséges, hogy a PING ki tudja számolni a csomagok megfordulási idejét. Tulajdonképpen a PING igen korlátozott módon ugyan, de sávszélesség-mérésre is alkalmas. Jártam már úgy, hogy egy bérelt vonalról meg kellett mondanom az abban lévő „maradék" sávszélességet, s mindehhez a nagy semmi állt rendelkezésemre. Ilyenkor jól jöhet egy kis furfang! Küldjünk át a mérendő vonalon akkora pingeket, hogy az valami mérhető megfordulási időt okozzon *(a lokális hálózatokon alapban „mérhető" <10 ms megfordulási idő valójában mérhetetlenül gyorsat jelent)*. Például:

```
PING –l 10000 www.internetszolgaltato.hu
```

Ha a vonal mondjuk 128 kilobites, akkor körülbelül és durván *(128/8=)*16 kilobájt átvitele egy másodperc alatti-körüli válaszidőt kellene adnia, s ha ettől jelentősen eltér akkor meg tudjuk becsülni, hogy mennyi is a rendelkezésünkre álló sávszélesség.

Fóti Marcell
marcellf@netacademia.net
MCT, MCSE+I, MCDBA, MZ/X

A cikkben található URL-ek

[1] tech.net-join@lyris.netacademia.net
[2] tech.net@lyris.netacademia.net

(folytatjuk...)

Tárolási megoldások az Exchange 2000 Server-hez

Az Exchange 2000 adattárolóinak kialakításakor három fő szempontot kell figyelembe venni: tárolókapacitás, rendelkezésre állás és teljesítmény. A tárolási megoldás tervezésekor és kivitelezésekor hozott döntések nagyban befolyásolják az Exchange 2000 felügyeletével és karbantartásával kapcsolatos költségeket.

Az Exchange 2000 tárterületigénye nagyjából megegyezik a postafiókok számának és a postafiókonként szükséges tárterület szorzatával. Ha nyilvános mappákat is akarunk használni, természetesen hozzá kell adnunk ehhez a számhoz az ezek tárolásához szükséges tárterület nagyságát.

A levelezőrendszerre fordított erőforrások és pénz mennyisége a vállalat igényeitől függ *(vannak olyan cégek, melyek kis mennyiségű elektronikus levelezést folytatnak, és nem is tartják az elektronikus levelezést létfontosságúnak, de ma már a cégek többségénél az elektronikus levelezés nélkülözhetetlen)*, és ez jelentősen befolyásolja az üzembehelyezett megoldás megbízhatóságát és rendelkezésre állását. A rendelkezésre állás redundáns egységek üzembehelyezésével fokozható. A processzorok redundánssá tétele a kiszolgálók fürtözésével, az adatok redundanciája pedig RAID megoldások üzembehelyezésével érhető el.

A teljesítmény iránti igény vállalatonként változó *(ebben a cikkben a teljesítmény alatt az adatátvitel sebességét értjük)*. Az adatátviteli teljesítményt általában a tárolóeszköz és a hozzá kapcsolódó szoftver által másodpercenként végrehajtott írási és olvasási műveletek számával határozzuk meg.

Az Exchange 2000 tárolási megoldásának tervezése előtt meg kell állapítanunk, hogy cégünk milyen fontossági sorrendet állít fel a három fő szempont között, itt főleg a rendelkezésre állás és a teljesítmény közti egyensúly megteremtése fontos.

Ebben a cikkben az Exchange 2000 postafiókjainak tárolására alkalmas tárolók tervezésének alapelveit vizsgáljuk. Nem foglalkozunk sem a gyakorlati megvalósítással, sem a fürtözött megoldások tárolóinak kialakításával, sem a nyilvános mappák tárolásával, de természetesen a cikkben vázolt elvek felhasználhatók ilyen megoldások kialakításakor is.

A tárolási eljárások áttekintése

Az Exchange 2000 telepítésekor alapértelmezésben minden adat azon a meghajtón lesz tárolva, melyre magát az alkalmazást telepítettük. Az ilyen, alapértelmezett beállításokkal telepített rendszer teljesítményét, rendelkezésre állását és tárolókapacitását az alábbi tényezők határozzák meg:

- A processzorok száma és sebessége
- A kiszolgáló feladata *(például postafiókok tárolása, connector (kapcsolódást biztosító) kiszolgáló)*
- A merevlemezek száma

Az Exchange 2000 tárolási megoldásának tervezésekor a következőkre figyeljünk:

- A CPU terhelése csökkenthető RAID tömbök, vagy SAN-ok *(storage area network - tárolóhálózat)* kialakításával.
- Több kisebb merevlemez jobb teljesítményt nyújt, mint egy nagy. Például, ha 4 darab 9 GB-os merevlemezt alkalmazunk egy darab 36 GB-os helyett, a tárolótömb típusától függően akár négyszeres írási és olvasási sebességet is elérhetünk.
- Az Exchange nem ugyanúgy kezeli az összes tárolt adatát, ezért nem az a leghatékonyabb, ha az összes adattípushoz egyféle tárolási megoldást alkalmazunk.
- Azoknál a kiszolgálóknál, melyek nem postafiókok vagy nyilvános mappák tárolására szolgálnak *(például a connector kiszolgálók)*, nem érdemes költséges tárolási megoldásokat használni, mert ezek általában csak rövid ideig tárolják az adatokat, majd továbbítják őket egy másik kiszolgálónak. Nagyon nagy teljesítményigény esetén esetleg indokolt lehet ezeken a kiszolgálókon a RAID-0 alkalmazása.

Az Exchange 2000 tárolási megoldás tárolókapacitásának, teljesítményének és rendelkezésreállásának növeléséhez RAID vagy SAN megoldásokat, esetenként pedig ezek mindegyikét kell használnunk.

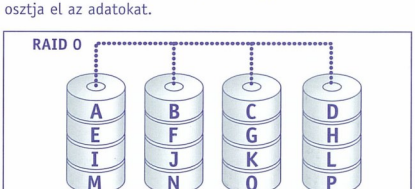
RAID megoldások

Bár sokféle RAID megvalósítás létezik, van két dolog, ami mindegyikben közös. Mindegyiknél több merevlemezt használunk, melyeken eloszlanak az adatok, és mindegyiknél valamilyen rendszer szerint tároljuk az adatokat, ami az adatokat tároló alkalmazástól független.

Ebben a cikkben a három fő RAID típust *(és az egyik altípust)* ismertetjük: a RAID-0-t, a RAID-1-et *(és annak 0+1 altípusát)* és a RAID-5-öt. Bár sokféle RAID megoldás létezik, ezek valamilyen formában a fenti három fő típus kombinálásával hozhatók létre, ezért ezeket nem ismertetjük.

RAID-0

A RAID-0 egy merevlemeztömb, amin egy csíkkészlet van kialakítva. Minden lemez úgy van particionálva, hogy a csík végigfut a tömb összes merevlemezén, így azok egyetlen logikai partícióként jelennek meg. Egy RAID-0-t használó alkalmazás például a D: meghajtóra menti az adatokat, míg a tömb elosztja azokat a merevlemezmeghajtók között. Az 1. ábrán látható RAID-0 tömb például négy merevlemez között osztja el az adatokat.

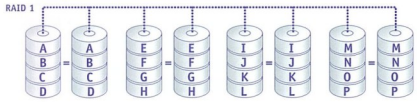


RAID-0 merevlemez tömb

Teljesítmény szempontjából a RAID-0 a legjobb RAID megoldás, mivel *(példánk szerint)* egyszerre négy lemezt használhat az írási és olvasási műveletek végrehajtásakor, ami a merevlemezek leghatékonyabb használatát jelenti.
A RAID-0 egyetlen hibája az, hogy valójában nem is RAID :), hiszen nem redundáns, és éppen ezért nem megbízható. Ha az Exchange postafiókadatbázisait egy RAID-0 tömbön tárolnánk *(ne tegyük!)*, egyetlen merevlemez meghibásodása esetén az összes postafiókadatbázist vissza kell állítanunk a működőképessé tett RAID-0 tömbre, és vissza kell állítanunk a tranzakciók naplófájljait. Ha pedig a tranzakciók naplófájljait is ezen a tömbön tároltuk, akkor csak az utolsó mentésben meglévő postafiókadatbázisokat tudjuk visszaállítani.

RAID-1 *(és 0+1)*

A RAID-1 két tükrözött merevlemezből álló tömb. Ha több mint két lemezt akarunk használni, a lemezek csíkkészletekbe vannak rendezve, és a csíkkészletek vannak tükrözve, így a tömb minden merevlemeze meg van kettőzve. Ez a RAID-0+1. Egy RAID-0+1 tömbben az összes lemez tárolókapacitásának felét használhatjuk ki. Írási műveletkor ugyanaz az adat fog a lemezre, és a lemez tükrözött párjára kerülni.

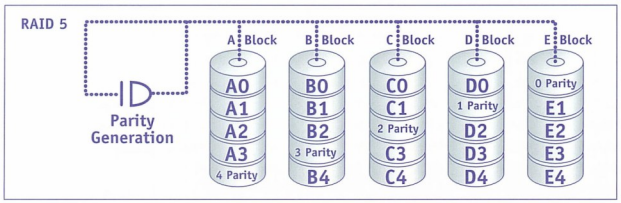


RAID-1 merevlemez tömb

A RAID-1 *(vagy a 0+1)* a legmegbízhatóbb a háromféle RAID tömb közül, hiszen az adatok íráskor tükrözve vannak. Az összes merevlemez tárterületnek csak a fele használható, ezért úgy tűnhet, hogy a RAID 1 *(vagy 0+1)* nem hatékony megoldás, mégis ezt használjuk, ha a lehető legnagyobb adatbiztonságot akarjuk elérni.

RAID-5

A RAID-5 - a RAID-0-hoz hasonlóan - egy olyan merevlemez tömb, melyen egy csíkkészlet van kialakítva, de a RAID-5 tartalmaz paritást is. Ez azt jelenti, hogy a RAID-5 tömbben működik egy olyan mechanizmus, mely biztosítja a tömbben tárolt adatok sértetlenségét. Egy merevlemez meghibásodása esetén az adatok helyreállíthatók a megmaradt lemezekről, ezért a RAID-5-öt megbízható tárolási megoldásnak tekintjük.



RAID-5 merevlemez tömb

A paritás fenntartásához minden egyes gigabájtnyi tárolóterületünkből fel kell áldoznunk 1/n GB-ot *(az n a tömb merevlemezeinek száma)*, vagyis a RAID-5 megoldásnál, egy darab lemezt fel kell „áldoznunk". Hat darab 9 GB-os merevlemezből tehát 45 GB hasznosítható tárterületünk lesz. A paritás fenntartásához egy alkalmazás írási művelete két írási, és két olvasási műveletet jelent a RAID-5 tömbben, így csökken a teljesítmény.
A RAID-5 megoldás előnye az, hogy megbízható, és a RAID-1-nél *(vagy a 0+1-nél)* hatékonyabban használja ki a rendelkezésre álló merevlemezeket.

A RAID megoldások összehasonlítása

Mivel a szükséges tárolókapacitás általában adott, a RAID megoldások költségeit, teljesítményét és a megbízhatóságát fogjuk összehasonlítani. A következő táblázatunk készítésekor az alábbiakat feltételeztük:

- 90 GB adatot kell tárolnunk.
- 9 GB-os merevlemezeket használhatunk.
- A merevlemezeink 100 I/O műveletet képesek elvégezni másodpercenként

RAID megoldás	Eszközök száma (költség)	Írási műveletek maximális száma másodpercenként	Olvasási műveletek maximális száma másodpercenként	Megbízhatóság
RAID-0	10	1000	1000	Nem megbízható
RAID-0+1	20	1000	2000	Nagyon jó
RAID-5	11	275	1100	Jó

A RAID megoldások összehasonlítása

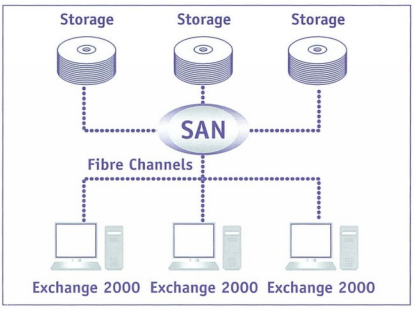
A RAID-1-nél csak két merevlemez használható, ezért nincs feltüntetve a táblázatban.
A megbízhatóságot azzal mérjük, hogy egy merevlemez meghibásodása milyen hatással van az adatok sértetlenségére. A RAID-0-ban nincs redundancia, így egyetlen merevlemez meghibásodása is az összes adat elvesztését jelenti. A RAID-0+1 a legmegbízhatóbb megoldás, mert legalább két merevlemeznek el kell romlania ahhoz, hogy adatvesztés következzen be. Ez is nagyon egyedi eset lenne, mert a csíkkészletekben egymásnak megfelelő lemezeknek kellene ugyanakkor meghibásodniuk. A táblázatunkban feltüntetett megoldás akár tíz merevlemez *(az egyik teljes csíkkészlet)* egyidejű meghibásodását is elviseli. A RAID-5 megoldás is megfelelő megbízhatóságú, de ennél bármely két lemez egyidejű meghibásodása adatvesztést eredményez.
A költséget a tömb kialakításához szükséges merevlemezek számával mérjük. A RAID-0+1 megoldás a legköltségesebb. Ennél a szükséges tárterület kétszeresét kell megvásárolnunk, de teljesítménye is lényegesen nagyobb, mint az azonos tárolókapacitású RAID-5 megoldásé.

SAN megoldások

A SAN-ok *(storage area network - tárolóhálózat)* tárolási lehetőségeket és azok felügyeletét biztosítják. A SAN-ok úgynevezett Fibre Channel switching *(kapcsolt, üvegszálas hálózati)* megoldásokat biztosítanak az alkalmazások, és az általuk használt tárolók közti gyors és megbízható kapcsolatok megvalósításához.

Egy SAN három fő összetevőből épül fel:
- Fibre Channel switching megoldás
- Tárolórendszer
- Tároló és SAN felügyeleti szoftver

A hardvergyártók általában teljes SAN csomagokat adnak el, melyek tartalmazzák a szükséges hardvert, szoftvert és a támogatási szolgáltatásokat. A SAN szoftver felügyeli a hálózatot, és megvalósítja az adatátvitel redundanciáját azzal, hogy a tárolt adatokhoz több elérési utat biztosít. A SAN-ok lehetővé teszik heterogén rendszerek összekapcsolását, akár különböző operációs rendszerek, több gyártótól származó tároló-rendszerekhez való kapcsolását.



SAN tárolási megoldás

Jelenleg a SAN-ok jelentik a legjobb megoldást nagymennyiségű adat biztonságos tárolására. Még a legkisebb SAN-ok is képesek 5 terabájtnyi adat tárolására.

Bár üzembehelyezésük költséges, a hosszú távú teljes birtoklási költség alacsonyabb lehet, mint a több kisebb tömbnél. A SAN megoldások alábbi előnyeit nem lehet figyelmen kívül hagyni:
- Ha több tömbünket több rendszergazda felügyeli, a teljes tárolórendszer központosított felügyelete lehetővé teszi, hogy rendszergazdáink más feladatokat lássanak el.
- A gyártója által támogatott SAN megoldással semmilyen más tárolórendszer nem veheti fel a versenyt megbízhatóság területén. Ha vállalatunknak a levelezőrendszer *(vagy bármely más rendszer)* kiesése jelentős veszteséget okozna, egy SAN költséghatékony megoldást jelenthet.

Egy SAN megvásárlása előtt mindenképpen érdemes összehasonlítani a jelenlegi megoldás kapcsán felmerülő költségeket és a SAN költségeit, valamint érdemes felmérni, hogy valójában mennyire lenne kihasználható vállalatunkban a SAN.

Az Exchange adatainak tárolása

Az Exchange három fő helyen tárol adatokat:
- A Simple Mail Transfer Protocol *(SMTP)* várakozási sor könyvtárban
- .edb és .stm fájlokban
- Tranzakciók naplófájljaiban

Az SMTP várakozási sor könyvtár

Az SMTP várakozási sor addig tárolja az üzeneteket, amíg azok továbbítódnak egy adatbázisba *(az üzenet típusától függően személyes vagy nyilvános adatbázisba)*, másik kiszolgálóhoz, vagy connectorhoz.

Általában az üzenetek rövid ideig vannak az SMTP várakozási sorban, ezért annak tárolási megoldását inkább nagyobb teljesítmény eléréséhez kell igazítani, nem a tárolókapacitáshoz és megbízhatósághoz. Néha – amikor az üzeneteket nem lehet továbbítani – az SMTP várakozó sorban nagymennyiségű adatot kell tárolni, emiatt a RAID-0 csak akkor elfogadható tárolási megoldás, ha az üzenetek elvesztése elfogadható. A RAID-1 megfelelő megoldás lehet, mert megbízható, és jó adatátviteli teljesítményt biztosít.

Az .edb és .stm fájlok

Egy Exchange adatbázis egy rich-text .edb fájlból, és egy natív multimédia tartalmú .stm fájlból áll. Az .edb fájl tárolja az összes MAPI üzenetet, a tárolási folyamat táblákat használ az összes üzenet, az edb és stm fájlok ellenőrző összegeinek és a MAPI üzenetek elhelyezésére. Az .stm fájl natív Internet tartalommal rendelkező üzeneteket tartalmaz. Mivel az edb és stm fájlok elérése általában véletlenszerű, elhelyezhetők ugyanazon a tárolón.

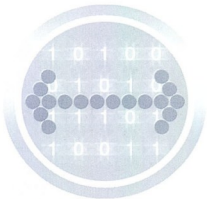
Ezeknek a fájloknak a tárolásához megbízható megoldásra van szükség, tehát a RAID-0 alkalmazása nem ajánlott. A fontossági sorrendben a megbízhatóságot követheti a teljesítmény *(RAID-1)*, de a tárolókapacitás *(RAID-5)* optimalizálása is. Mi inkább a RAID-1 *(vagy 0+1)* használatát ajánljuk. Mivel a nyilvános mappákba általában kevesebbszer írnak, mint ahányszor olvassák őket, ezek fájljaihoz jó megoldás lehet a RAID-5.

A tranzakciók naplófájljai

A tranzakciók naplófájljai nyilvántartják az .edb és .stm fájlok állapotát, és biztosítják azok sértetlenségét, ami azt jelenti, hogy a naplófájlok jelképezik az adatokat. Minden tárolócsoport rendelkezik saját tranzakciói naplófájljával *(adatbázissal)*. Ha meghibásodás történik, és újra üzembe kell állítani a kiszolgálót, az utolsó tranzakció naplófájlok használhatók az adatbázisok helyreállítására. Ha megvannak a naplófájlok, és az utolsó biztonsági mentés, az adatok visszaállíthatók, ha a naplófájlok elvesznek, elvesznek az adatok is.

A meghibásodás utáni visszaállítás mértéke a naplófájlokon múlik. A jó megbízhatóság és teljesítmény elérése érdekében célszerű minden tárolócsoport naplófájljait külön meghajtón tárolni. A tranzakciók naplófájljainak tárolóinál a megbízhatóság és a teljesítmény a legfontosabb, ezért a RAID-1 *(vagy 0+1)* használata ajánlott.

MetaBase: biztos háttér az IIS mögött



A Windows 2000 család szerves részeként az Internet Information Services 5.0-s verziója a 4-es kiadás pozitív hagyatékát erősítendő a lehető legtöbb beállítási információt – a korábbi IIS változatokkal ellentétben amelyek teljesen a regisztrációs adatbázisba dolgoztak – a metabase elnevezésű tárolási helyre menti. A szerverek webkiszolgálási szerepének előtérbe kerülésével szükségessé vált, hogy a gyakran változó, a különböző szerveralkalmazások által kiolvasandó és változtatandó IIS beállítási paraméterek ne olyan relatíve lassú elérésű helyen tárolódjanak, mint a regisztrációs adatbázis. A biztonsági kérdések elkülönítve kezelhetőek - így elkerülhetőek a regisztrációs adatbázison belüli potenciális jogosultság-problémák. A metabase azonban ennél is több: az IIS alapú webkiszolgálók manuális és automatizált rendszeradminisztrációja teljesen rá épül, valamint a szerveroldali webprogramozás finomhangolása és opcionális paraméter-tárolása is általa oldható meg. Remélem, hogy sikerül sok olyan kérdést is megválaszolnom ebben a cikkben az IIS-sel kapcsolatban, amelynél gondolni sem mertünk volna arra, hogy ez a misztikus név mögé rejtőző konfigurációs adatbázis a hunyó.

Otthon, édes otthon

A metabase rejtekhelye kezdetben egy állományban található a merevlemezen, majd az IIS szolgáltatások elindulásakor a memóriába töltődik *(a már korában említett sebesség-okok miatt)*. A változtatásokat a masina automatikusan, rendszeres időközönként menti a merevlemezre, és az IIS szolgáltatások megállásakor is teljesen visszaíródik a fájlba, amely alapértelmezésben az alábbi elérési úton található:

```
%SystemRoot%\winnt\system32\inetsrv\MetaBase.bin
```

Ezt az elhelyezést meg tudjuk változtatni, ha szükséges, erre a regisztrációs adatbázisban a

```
LOCAL_MACHINE\SOFTWARE\Microsoft\INetMgr\
↳ Parameters
```

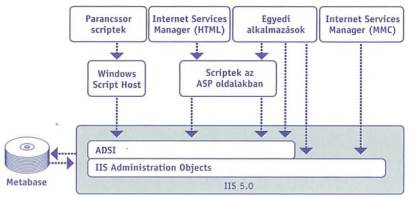
kulcshoz felvett REG_SZ típusú, MetadataFile elnevezésű érték felvételével van lehetőségünk, ahol is az adatmezőbe az általunk manuálisan áthelyezett metabase állomány új, teljes elérési útját és nevét kell beírnunk egyben, pl.: i:\newmetabase\newmetabase.bin. Védjük a fájlt NTFS jogosultságokkal, amit ne felejtsünk el a mentések *(lásd később)* által létrehozott tartalékokon is elvégezni!

A metabase logikai felépítése

A metabase belső logikai felépítése sokban hasonlít a Windows család regisztrációs adatbázisainak felépítésére, amelyben az alap kiindulási csomópontok *(node)* alatt hierarchikus struktúrában kulcsokat és az azokhoz kapcsolódó értékeket találjuk. A kulcsok alatt további alkulcsok találhatóak, az IIS logikai felépítését tükrözve. Minden egyes webkönyvtár rendelkezik egy neki megfelelő kulccsal a metabase-ben. Az értékeknek 3 fajta adattípusa létezik: DWORD, Binary és String; valamint a String típusúnak két további variációja: Expand String és Multi-String. Természetesen egy adott kulcs *(másnéven metabase tulajdonság)* csak bizonyos értékeket, és azokon belül bizonyos értéktípusokat támogathat, behatárolva ezáltal a beállítható adatokat. A magasabb szinten definiált paraméterek értékeit alapértelmezésben öröklik az alacsonyabb szinten találhatóak, azonban van lehetőség egyedi beállításokra is. A metabase hierarchikus felépítése az IIS elrendezését követi, tehát a gyökér web, ftp, smtp szolgáltatások alatt/mellett megtaláljuk a virtuális kiszolgálók, könyvtárak és webhelyek al-csomópontjait. Az egyedi kulcsokra és értékükre a teljes metabase elérési úttal *(path)* hivatkozhatunk, amelyben az egyes szinteket sima per *(/)* jellel választjuk el. Egy példa elérési út tehát így néz ki: W3SVC/1/Root/path, amely az alapértelmezett webkiszolgáló gyökérkönyvtárának a merevlemezen való elhelyezkedését tárolja. A fő-csomópontok között az LM a LocalMachine rövidítéseként az adott gép egyes szolgáltatásainak *(web, smtp)* konténereként szolgál. Az alá tartozó kulcsok közül a W3SVC a web, az SMTPSVC pedig az smtp szolgáltatás alapját képezik. Az LM-mel egyenrangú fő-csomópontként a Schema a kulcsok és értékek összefoglaló katalógusát tartalmazza.

IIS rendszeradminisztráció

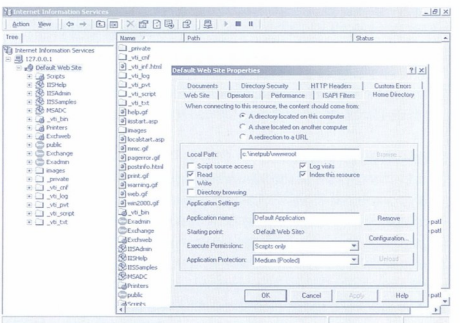
Elérkeztünk a legérdekesebb kérdéskörhöz: hogyan is tudjuk az IIS paramétereit állítani? Hogyan férünk hozzá a „halandó" rendszergazdák elől elrejtett értékekhez? A metabase értékeinek megtekintésére és módosítására rendszerfelügyeleti szempontból sokféle lehetőség is van. Az IIS menedzsment-architektúrája teljes mértékben a metabaseben végződik, azaz a legalapvetőbb kiindulási pontként minden érték, paraméter ide íródik be, és innen olvasódik ki. Tehát az IIS felügyelete valójában a metabase tartalmának manipulálásával egyenértékű. Ez alól kivétel néhány, az inetinfo.exe futtatására vonatkozó globális paraméter, amelyek a regisztrációs adatbázis lakói. Az IIS menedzsmentjének két építőkockája az IIS Administration Objects *(IIS Admin Objects)* és az ADSI. Az ADSI-t részletesen tárgyaló cikksorozat a Tech.Netben lehetővé teszi, hogy átugorjam az ADSI általános részletezését. Az IIS Admin Objects ADSI szolgáltatóként jelenik meg a kliensek felé, ezáltal a szabványt betartva többféle módon is megközelíthető a metabase. Az IIS Base Admin Objectek, amelyek DCOM objektumként az IIS Admin Objectek mögött rejtőznek, az IIS alap rendszerobjektumaiként közvetlenül kezelhetőek az alacsonyabb szintű programnyelvek *(pl. C++)* által *(lásd dokumentálva a* **[1]** *címen)*, így bonyolultabb IIS rendszerkezelő funkciókat is implementálhatunk egy alkalmazásban.



Az IIS rendszeradminisztrációs architektúrája

Internet Services Manager

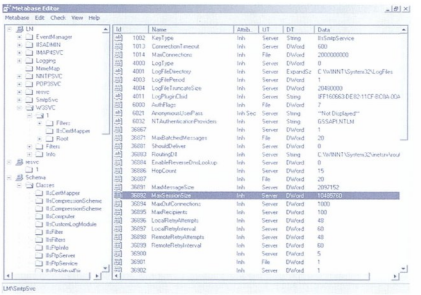
A legegyszerűbb, legismertebb grafikus felületű menedzsment-eszköz az IIS-sel feltelepíthető Internet Services Manager. Két változata közül a Microsoft Management Console *(MMC)* alapú széles funkcionalitást nyújt, mivel közvetlenül a DCOM objektumokkal dolgozik. A HTML alapú változat korlátozott eszközkészletével az ASP oldalakba ágyazott scriptekre épül *(lásd lejjebb)*. Ne feledjük, hogy még az erőteljesebb MMC alapú változatnál is az eredeti metabase hierarchia csak részben látszódik, valamint hogy a kulcsok valódi elnevezései, típusai teljes mértékben rejtettek maradnak előlünk. Előnyük, hogy távoli menedzselésre alkalmasak, mivel az MMC változattal akár egy munkaállomásról csatlakozhatunk más kiszolgálókhoz, míg a HTML Internet Services Manager-t futtató IIS kiszolgálókat egy egyszerű *(IE 4+)* böngésző segítségével, az Interneten vagy az intranetünkön keresztül vezérelhetjük.



Az Internet Services Manager jól ismert MMC változata

MetaEdit

A metabase professzionális szerkesztésére legalkalmasabb eszközként a MetaEdit nevű grafikus segédprogram hitelesen jeleníti meg az eredeti hierarchikus struktúrát, valamint általa módosíthatóvá válik minden érték, amely az Internet Services Managerben még rejtve maradt. A Windows 2000 Server Resource Kit CD-n *(és a Supplement One kiegészítőn)* található 2.0-s verziónál újabb érhető el a Microsoft weboldalán: **[2]**. A MetaEdit 2.1 segédprogram az Internet Information Server 4 és az IIS 5 metabase szerkesztésére egyaránt alkalmas – tehát telepíthető Windows 2000-re és Windows NT 4.0-ra egyaránt, a letöltési cím ne zavarjon meg senkit! A MetaEdit a magasabb szintű, ADSI alapú IIS Admin Objects és az alacsonyabb szintű, DCOM alapú IIS Base Admin Objects vegyes használatával oldja meg a rábízott feladatokat. Jelenlegi verziójában, amely helyi telepítést igényel az IIS-t futtató számítógépre, már letiltották az LM és a Schema főcsomópontok törlési lehetőségét. Kifinomult exportálási és importálási lehetőségekkel rendelkezik, amelyekkel egy vagy több kulcsot lehet szöveges állományba kiíratni és onnan visszatölteni, valamint teljes IIS web és FTP szerverek beállításait tudjuk egy külön opció használatával mozgatni több gép között. Ezzel a funkcióval a globális szolgáltatás-beállítások valamint az anonim webfelhasználó neve és jelszava kivételével minden paramétert átadhatunk, tehát ezt kiegészítő mentésként is alkalmazhatjuk.



MetaEdit 2.1 – az átfogó GUI megoldás

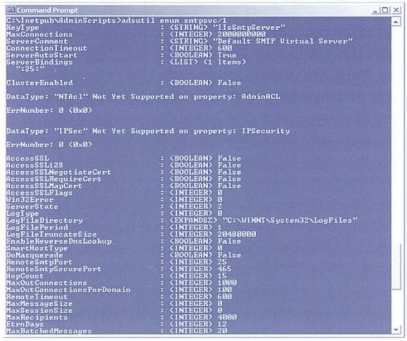
Szabványos ADSI lekérdezések: Adsvw.exe

Kihasználva a tényt, hogy a metabase tulajdonságait ADSI szabványnak megfelelően szolgáltatja az IIS, egy általános ADSI segédprogram is bevethető a lekérdezésekre. Szintén grafikus felületű, és letölthető a webről: **[3]**. Itt a metabase tulajdonságait az AdsPath szintaxissal érhetjük el. Nagy előnye az Adsvw.exe-nek, hogy minden ADSI-nak megfelelő címtárat kiolvashatunk vele, legyen az az Active Directory „csupaszon", vagy az Exchange 2000-rel kiegészítve.

Parancssoros megoldások

A jó öreg parancssoros vezérlés itt sem kikerülhető, ha a MetaEdit-et nem tudjuk/akarjuk használni. A %SystemRoot%\Inetpub\AdminScripts könyvtárban található adsutil.vbs egy előre *(VBScriptben)* elkészített általános IIS felügyeleti eszköz, amellyel közvetlenül a metabase kulcsokat és értékeiket változtathatjuk. Futtatásához a Windows Scripting Host szolgáltatás CScript értelmezőjét kell használnunk, amely nem az alapértelmezett a Windows 2000 esetében. Amikor elindítjuk az adsutil.vbs-t, a Windows Scripting Host felajánlja, hogy alapértelmezett VBScript futtatóvá tegye a CScript.exe-t. Ha nem szeretnénk elállítani a rendszert az eredeti Wscript értelmezőtől, akkor parancssorból a „Cscript.exe adsutil.vbs PARANCS <elérési út> [<paraméter>]" szintaxissal használhatjuk. A parancsok közül *(csak példaként említve)* a HELP kilistázza az elérhető parancsokat és paramétereiket. Az ENUM segítségével egy adott kulcs szintjén található értékeket, és a kulcs alá bejegyzett további kulcsokat listázhatjuk ki. Az

ENUM_ALL az összes kulcsot és értéket listázza, tehát itt elérési utat nem kell megadni. A GET egy konkrét kulcs és értékeinek megjelenítésére szolgál. A SET egy konkrét kulcs értékének megváltoztatását teszi lehetővé, ahol is az új értéket idézőjelek közt, „érték" formában kell megadni paraméterként. A CREATE-tel új kulcsot hozhatunk létre, ahol az elérési út mögött paraméterként a kulcs típusát kell definiálni.



Adsutil.vbs – metabase a'la parancssor

Az adsutil.vbs futtatási példái:

```
Listázás: Adsutil ENUM /smtpsvc/1
Kiíratás: Adsutil GET /smtpsvc/1/queuedirectory
Beállítás: Adsutil SET /smtpsvc/1/queuedirectory
  "d:\kj"
Létrehozás: Adsutil CREATE W3SVC/1/Root/MyVdir
  "IIsWebVirtualDir"
```

Egyéni fejlesztések – scriptelés, ASP oldalak

A korábban említett elérési módok közül az Internet Services Manager HTML verziója használja az alábbi módszert: az ASP programozás részeként az IIS Admin Objects elérési felületet bármilyen Automation-kompatíbilis script-nyelvvel használhatjuk *(pl.: VBScript, Jscript)* vezérelve. Az URL elérésekhez hasonló, az ADSI ADsPath szintaxissal érhetőek el az objektumok, pl: IIS://LocalHost/W3SVC/1/Root/path. Tehát a hagyományos metabase elérési út immáron kiegészül egy kezdeti paraméterrel, az IIS-t futtató gép megnevezésével, amely lehetővé teszi a távoli adminisztrációt. A teljes fejlesztési dokumentáció elérhető a weben: **[4].**

Gyakorlati tanácsok

Az Exchange 2000 Server SMTP szolgáltatás könyvtárai

Komolyan megtervezett kiszolgálók esetében szükséges a különböző típusú adatok-folyamatok elválasztása, külön fizikai lemezre helyezése. *(Lásd RAID cikkünket e lapszámban – a szerk.)* Az Exchange teljesítménye nagymértékben múlik a merevlemez-alrendszeren, főleg nagy terhelésű kiszolgálók esetében. Mivel az Exchange teljes mértékben az IIS SMTP service-re épülve oldja meg a levelek küldését-fogadását, ezért szükség lehet ezen szolgáltatás könyvtárainak áthelyezésére. Az Exchange nélküli, csak az IIS SMTP-re épülő mailrendszerek is igényelhetik ezt a hangolást. Mivel az Inetpub alapértelmezésben a rendszermeghajtóra kerül, az smtp szolgáltatás a Mailroot alkönyvtárban lesz, a WWWRoot mellett. Míg a WWWRoot-ot egy mozdulattal áthelyezhetjük a Internet Services Manager-ben, a default SMTP szerverünk könyvtárait *(pl.: Queue, Badmail)* csak a metabase közvetlen szerkesztésével tudjuk átállítani. További részletek a Q257835-es azonosítójú KB cikkben olvashatók.

Mentés és visszaállítás

A metabase teljes mentésére használhatjuk az Internet Services Manager-t, vagy a MetaEdit-et, *(mindkettő ugyanazt a felületet nyújtja erre a célra)*. Az Internet Services Manager-ben a gép ikonján jobb gombot nyomva érjük el a backup/restore modult. Innen két kattintással készíthetünk pillanatfelvételt a rendszerünkről, mielőtt komolyabban nekiállnánk a potenciálisan végzetes „finomhangolásnak". A MetaEdit 2.1 Metabase menüjének szintén Backup/Restore parancsával indíthatjuk a folyamatokat. Katasztrófa vagy helyrehozhatatlan változtatások esetében a visszatöltés is ugyaninnen végezhető el. A lementett metabase állomány automatikusan a X:\winnt\system32\inetsrv\MetaBack könyvtárba kerül. Ne feledjük, hogy a lementett metabase csak ugyanarra a gépre tölthető vissza, és újratelepített rendszer sem jöhet szóba a visszaállításhoz. Programozott felületek esetében az IIS Admin Objects IisComputer objektum Backup módszerével dolgozhatunk.

Az IIS metabase replikációja fürtözött rendszerekben

Az alábbi információk természetesen csak a Windows 2000 Advanced Server változatra vonatkoznak, mivel a fürtözési technológiát a Standard változat nem támogatja. A fürtcsoport létrehozása után az IIS-t a fürtözött erőforrások közé a dokumentáció szerint felvesszük. A rendszer feltételezi, hogy a webtartalom a megosztott merevlemezen található, vagy a saját merevlemezeken külön szinkronizálva, azonos elérési útvonalon van. Ezután kell lefuttatni manuálisan a forrás *(mester)* kiszolgálón a %Systemroot%\system32\inetsrv könyvtárban az iissync.exe parancsot, paraméterként a célkiszolgáló*(k)* nevét megadva. A replikáció nem automatikus, ezért változtatások után mindig gondoskodni kell a kézi szinkronizációról.

Köszönet Soczó Zsoltnak, aki először hívta fel a figyelmemet a metabase fontosságára!

Radványszki Gábor MCP
SVED Rt.

A cikkben található URL-ek:

[1] IIS Platform SDK Programmer's Guide
http://msdn.microsoft.com/library/psdk/iisref/prog7hwl.htm
[2] MetaEdit 2.1
http://download.microsoft.com/download/iis40/other/4.0/WIN98/EN-US/MtaEdt21.exe
[3] Adsvw.exe – az ADSI 2.5 SDK része
http://www.microsoft.com/ntserver/nts/downloads/other/ADSI25/default.asp
[4] IIS 5.0 Documentation
http://windows.microsoft.com/windows2000/en/server/iis/

A Windows 2000 biztonsága *(I. rész)*

Bevezető

A Microsoft Windows 2000 Server operációs rendszer elosztott biztonsági szolgáltatásai biztosítják a hálózat felhasználóinak azonosítását, és az erőforrások elérésének szabályozását. Az operációs rendszer biztonsági modellje a megbízható tartományvezérlőn történő hitelesítésen, a hitelesítési adatok szolgáltatások közti átadásán és objektumalapú hozzáférésszabályozáson alapul. A fő szolgáltatások a következők:

- Windows 2000 Active Directory szolgáltatás
- Kerberos version 5 hitelesítési protokoll
- a külső felhasználók hitelesítésére nyílt kulcsú titkosítás használata
- titkosított fájlrendszer *(Encrypting File System – EFS)* a helyi adatok védelme érdekében
- Internet Protocol security *(IPSec)* a nyilvános hálózatokon folytatott biztonságos kommunikáció biztosítására

Emellett a Windows 2000 biztonsági elemei felhasználhatók saját alkalmazásainkban, sőt, a Windows 2000 biztonsági rendszere integrálható más, Kerberos hitelesítést használó operációs rendszerekkel is.

Ahogy a fentiekből is látható, a Windows 2000 nemcsak a funkciókban gazdag hálózatok kialakítását, hanem azok biztonságossá tételét is biztosítja.

A Windows 2000 biztonsági szolgáltatásait úgy alakították ki, hogy megfeleljenek az alábbi követelményeknek:

- az összes hálózati erőforrás elérése egyszeri bejelentkezéssel
- erős felhasználóhitelesítés és –azonosítás
- biztonságos kommunikáció a belső és külső erőforrások között
- a szükséges biztonsági házirendek létrehozásának és kezelésének képessége
- automatikus biztonsági ellenőrzések
- más operációs rendszerekkel és biztonsági protokollokkal való együttműködés
- bővíthető architektúra

A Windows 2000 biztonsága egyszerű hitelesítési és azonosítási modellen alapul. A hitelesítés bejelentkezéskor azonosítja a felhasználót, és létrehozza a hálózati szolgáltatásokkal a kapcsolatokat. Ha a felhasználó azonosítása megtörtént, jogosultságai alapján fér hozzá a hálózat megadott erőforrásaihoz. A jogosultságok kiértékelése a hozzáférésszabályozási listákon *(access control list - ACL)* alapul, melyek meghatározzák az objektumokhoz *(nyomtatók, fájlok, hálózati fájl- és nyomtatómegosztások)* való hozzáférés engedélyezett módjait *(például írás engedélyezve, törlés tiltva)*. Ez a biztonsági modell még a nagykiterjedésű hálózatokban sem akadályozza a felhasználók munkavégzését, de jó védelmet nyújt a támadások ellen.

A tartományi ügyfél először bejelentkezik a tartományvezérlőre. Azonosítatlan felhasználó soha nem fér hozzá a hálózat erőforrásaihoz *(kivéve az anonymous elérésű szolgáltatásokat – a szerk.)*. A hálózati szolgáltatások létrehozzák az ügyfél access token-jét, és a kért műveletek végrehajtása előtt a hitelesítéskor keletkezett bizonyítvánnyal azonosítják az ügyfelet. A Windows operációs rendszer kernele az access token-ben levő biztonsági azonosítókat használja annak megállapítására, hogy a felhasználónak az adott objektumra megadhatók-e a kért hozzáférési jogok.

Cikkünkben a Windows 2000 biztonsági szolgáltatásai közül a következőkkel foglalkozunk: Active Directory, hitelesítés és azonosítás, Kerberos hitelesítési protokoll, nyílt kulcsú infrastruktúra *(PKI)*, bizonyítványok támogatása, titkosított fájlrendszer *(EFS)*. Bemutatjuk, mire használhatják az alkalmazásfejlesztők a Windows 2000 biztonsági szolgáltatásait, és hogyan működnek ezek együtt más operációs rendszerek szolgáltatásaival.

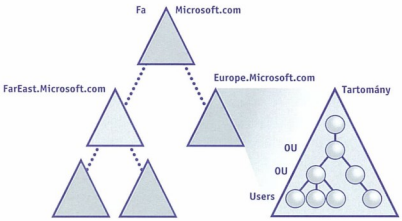
Az Active Directory szerepe

Az Active Directory kulcsszerepet játszik a biztonságos hálózat megvalósításában. Mind a Windows 2000 Server, mind a Windows 2000 Professional képes az egyes PC-ken tárolt adatok megvédésére, de a teljes körű, házirendalapú, a hálózat erőforrásainak elérését szabályozó biztonsági rendszer megvalósításához együttes használatuk ajánlott, mert így képesek kihasználni az Active Directory összes elosztott biztonsági funkcióját. Az Active Directory központosítja a felhasználókról, hardvereszközökről, alkalmazásokról és a hálózaton található adatokról szóló információk tárolását, így a felhasználók könnyen megtalálhatják, amit keresnek. Itt tárolódnak a hitelesítési és azonosítási információk is, melyek biztosítják, hogy csak a feljogosított felhasználók érhessék el a hálózat erőforrásait. Emellett az Active Directory szorosan integrálva van a Windows 2000 biztonsági szolgáltatásaival, például a Kerberos hitelesítési protokollal, a PKI-val, az EFS-sel, a Security Configuration Manager-rel, és a csoportos házirendekkel.

Az Active Directory alapjai

A Windows 2000 biztonsági szolgáltatásainak megértéséhez szükséges az Active Directory architektúrájának alapvető ismerete. Ha a Kedves Olvasó nem ismeri jól az Active Directory-t, hasznára válhat a következő rész, az Active Directory rövid áttekintése.

A Windows NT Server sík címtárszerkezetével ellentétben az Active Directory hierarchiában tárolja az adatokat, mely a vállalat felépítését tükrözi. Ez nagyobb elérhető méretet és egyszerűbb felügyeletet biztosít. A hierarchia létrehozásához az Active Directory tartományokat, szervezeti egységeket *(organizational unit - OU)*, és objektumokat használ, melyek segítségével a hálózati erőforrások egy PC fájl- és mappastruktúrájához hasonlóan szervezhetők.



Az Active Directory hierarchikus felépítése.

Egy tartomány különböző objektumok *(OU-k, felhasználói fiókok, csoportok, számítógépek)* gyűjteménye, melyek egy közös, biztonságos címtáradatbázisban helyezkednek el. A tartományok az Active Directory logikai struktúrájának fő elemei, így nagy szerepet játszanak a biztonsági rendszerben is. Az objektumok tartományokba foglalásával elérhető, hogy a hálózat felépítése tükrözze a cég szervezeti felépítését.

A nagyobb szervezeteknek több tartományuk is lehet, ez esetben a tartományok hierarchiáját *(tartomány)*fának *(domain tree)* nevezzük. Az első létrehozott tartomány a gyökértartomány *(root domain)*, mely az alatta létrehozott *(szép képzavar, feltaláltuk a lefelé növő fát! :))* utódtartományok *(child domain)* szülőtartománya *(parent domain)*. Annak érdekében, hogy nagyon nagy szervezetek is beleférjenek az AD-ba, a fákból erdő *(forest)* alakítható ki. *(Ha több tartományvezérlő van egy tartományban, az Active Directory megadott időközönként minden tartományvezérlőre replikálódik, így az adatbázisok mindig szinkronizálva vannak.)*

A tartomány biztonsági határvonalat képez az egységes belső biztonsági házirendek, és a más tartományokkal való kapcsolat szempontjából. Az adott tartomány rendszergazdája csak a saját tartományában állíthat be házirendeket.

A telephely *(site)* fogalma az Active Directory-ban levő kiszolgálók egy csoportját jelenti, mely a tartományokkal ellentétben nem logikai struktúra, hanem földrajzi elhelyezkedés szerint van kialakítva. Egy telephelyen belül a számítógépek általában nagysebességű vonalakon kapcsolódnak egymáshoz, de előfordulhat, hogy nincs köztük logikai kapcsolat. Például egy vállalat központi épületében más-más tevékenységeket végző, egymástól független szervezeti egységek AD kiszolgálói egy telephelyen vannak még akkor is, ha a a tartományok egymástól teljesen függetlenek.

A szervezeti egységek *(OU)* az objektumok tartományon belüli logikai egységekbe való szervezését szolgáló tárolók. Egy OU tartalmazhat: felhasználói fiókokat, csoportokat, számítógépeket, nyomtatókat, alkalmazásokat, fájlmegosztásokat és más OU-kat.

Az objektumok az egyes dolgok *(felhasználók, számítógépek, hardvereszközök)* jellemző attribútumait tartalmazzák. Például a felhasználó-objektum tartalmazhatja a keresztnév attribútumot, a nyomtató-objektum a színmélység attribútumot, és mindkettő tartalmazhatja a „hányadik emeleten van" attribútumot is.

Az információk tartományokba és OU-kba rendezésével közösen felügyelhetők az objektumcsoportok *(például felhasználók, számítógépek)* biztonsági beállításai. A nagykiterjedésű hálózatok kialakításánál óhatatlanul felmerül a kérdés: Hogyan működnek együtt az Active Directory megbízotti kapcsolatai *(trust relationship)* és a Windows 2000 biztonsági rendszere? Nos, lássuk!

A tartományok közti megbízotti kapcsolatok

Hogy a felhasználók egyszeri bejelentkezéssel *(single sign-on)* használhassák a hálózat erőforrásait, a Windows 2000 támogatja a tartományok közti megbízotti kapcsolatok *(trust relationship)* létrehozását. Ezek tartományok között létrehozott logikai kapcsolatok, melyek segítségével az erdő bármely tartományában hitelesíthetők a felhasználók és számítógépek, továbbá ezek biztosítják, hogy egyszeri bejelentkezéssel elérhetők legyenek a hálózat azon erőforrásai, melyekhez megfelelő jogosultságokkal rendelkezünk. Ehhez kapcsolódik a tranzitív megbízotti kapcsolat fogalma, mely azt jelenti, hogy a hitelesítési adatok akár több köztes tartományon áthaladva is elérhetik a megcélzott tartományt. Vagyis ha A bízik B-ben, és B bízik C-ben, akkor A bízik C-ben is.

A Windows NT alapú hálózatokban egyirányú, nem-tranzitív megbízotti kapcsolatokat lehetett használni. Ezzel ellentétben a Windows 2000 hálózatokban a tartományok fastruktúrába vannak rendezve, és implicit módon kialakult megbízotti kapcsolatok vannak köztük, ami főleg nagyméretű vállalatoknál egyszerűsíti le a tartományok közti megbízások létrehozását *(mert nem is kell létrehozni, hanem az AD megfelelő telepítésekor magától létrejönnek)*. Azok a tartományok, melyek egy fa tagjai, kétirányú megbízotti kapcsolatokat hoznak létre a szülőtartományukkal, ezért az összes fában levő tartomány megbízik egymásban. *(Ha szükség lenne egyirányú, explicit módon megadott megbízásokra, természetesen ilyenek is beállíthatók.)* A Windows 2000-re való áttéréssel különösen a sok tartományból álló vállalati hálózatokban csökken le jelentősen a megbízotti kapcsolatok száma, és ez a tartományok felügyeletét is leegyszerűsíti. A tranzitív megbízások automatikusan létrejönnek egy fán belül, hiszen általában ekkora egy rendszergazda „birodalma", éppen ezért kell külön létrehozni az erdő fái közti tranzitív megbízotti kapcsolatokat.

A megbízotti kapcsolatok szemléltetésére ismét előző ábránkat hívjuk segítségül. A Windows 2000 automatikusan létrehozza a gyökér- *(Microsoft.com)*, és utódtartományok *(FarEast.Microsoft.com és Europe.Microsoft.com)* között a kétirányú megbízásokat. Mivel a Microsoft.com bízik mindkét utódtartományában, ezzel létrejön az utódtartományok közti tranzitív megbízotti kapcsolat is. Ezek a megbízások a Windows 2000 tartományok között automatikusan jönnek létre. A vegyes *(NT és 2000)* hálózatok tartományai között a rendszergazdáknak kell explicit módon megadni az egyirányú megbízásokat, ugyanúgy, mint az NT-s hálózatokban.

A tartományok közti hitelesítést a Windows 2000 a Kerberos V5 protokoll, és a régi rendszerekkel való együttműködés érdekében az NTLM használatával biztosítja. Mivel a Windows 2000 tartományfája tranzitív megbízások rendszerét támogatja, nagyvállalatoknál leegyszerűsödhet a tartományok összekapcsolása és felügyelete. Fontos, hogy a tranzitív megbízások nem biztosítanak automatikusan olyan jogosultságokat, melyek az ACL-ekben nem szerepelnek, „csak" egyszerűbbé teszik azok beállítását.

A biztonság felügyelete csoportos házirendekkel

A csoportos házirendek *(Group Policy)* beállításaival az Active Directory objektumainak viselkedése szabályozható. A csoportos házirendek kezelése az Active Directory igen fontos funkciója, mert segítségével egyidejűleg sok számítógépen egységesen érvényesíthetők különféle beállítások. Például segítségükkel elvégezhetők biztonsági módosítások, és felügyelhetők az alkalmazások. A számítógépek csoportos házirendjei rendszerbetöltéskor, a felhasználóké pedig bejelentkezéskor érvényesülnek, és – ellentétben az NT4 házirendjeivel – időről időre frissülnek még akkor is, ha a felhasználó nem jelentkezik ki-be.

Az Active Directory háromféle tárolótípusára adhatók meg csoportos házirendek: OU-kra, tartományokra és telephelyekre. Az adott tárolóra érvényes csoportos házirend érvényesülhet az összes objektumon, vagy azok meghatározott csoportján.

A csoportos házirendek alkalmazhatók széles körben érvényes biztonsági házirendek megadására. A tartományszintű házirendek érvényesek a tartomány összes felhasználójára, és tartalmazhatják például a minimális jelszóhosszt és jelszóváltoztatás gyakoriságát megadó házirendeket. Beállítható az is, hogy alacsonyabb szinten felülbírálhatók-e ezek a beállítások.

A széleskörű biztonsági házirendek érvényesítése mellett lehetőség van az egyes PC-k biztonsági beállításainak finomhangolására is. A számítógépek helyi biztonsági beállításainál megadhatók a felhasználók és más számítógépek jogosultságai. Ezekkel határozható meg például, hogy ki végezheti el a kiszolgáló biztonsági mentését, vagy ki auditálhatja az asztali gépeken levő adatok elérését.

Az adott számítógép beállításai végül a házirendekben megadott szabályok összessége lesz.

Hitelesítés és hozzáférésszabályozás

A hitelesítés fontos része a rendszer biztonságának. Arra szolgál, hogy a tartományba bejelentkező, vagy hálózati erőforrást használni akaró felhasználó személyazonosságát megállapítsuk. A Windows 2000 hitelesítése része annak a folyamatnak, mely biztosítja, hogy a felhasználó egyszeri bejelentkezéssel elérje a hálózat erőforrásait, és hitelesítse magát a hálózat bármely számítógépén.

A sikeres felhasználóhitelesítés két részből áll. Az elsőnél, az interaktív bejelentkezésnél a felhasználó által megadott azonosítók összevetésre kerülnek a tartományi-, vagy helyi fiók azonosítóival. A második, a hálózati hitelesítés a felhasználói azonosítók alapján jóváhagyhatja bármely hálózati szolgáltatás elérését. Ha a felhasználó hitelesítve van, és elérhet egy objektumot, akkor a neki kiosztott, vagy az objektumra érvényes jogosultságok alapján kerül meghatározásra az engedélyezett elérés típusa. A tartományi objektumokon az objektumtípus objektum-kezelője kikényszeríti a hozzáférési szabályok érvényesítését *(például a rendszerleíró adatbázist csak a kulcsaira érvényes jogosultságok erejéig bántalmazhatjuk)*.

Hitelesítés

A felhasználónak rendelkeznie kell egy Active Directory-ban tárolt felhasználói fiókkal, hogy bejelentkezhessen egy számítógépre, vagy egy tartományba. Az operációs rendszer ezt használja a felhasználó hitelesítésére és a tartományban levő erőforrások elérésének engedélyezésére.

A felhasználói fiókok használhatók szolgáltatás-fiókként, vagyis akár egy szolgáltatás is bejelentkezhet felhasználóként *(hitelesíti magát)*, és így a felhasználói fiók jogosultságainak megfelelő műveleteket hajthat végre.

A felhasználói fiókokhoz hasonlóan, a Windows 2000 számítógép-fiókok is egyfajta hitelesítést biztosítanak a számítógép hálózat- és erőforrás-elérésekor. Minden Windows 2000-es számítógépnek, amelynek erőforrás-elérést akarunk engedélyezni, rendelkeznie kell egyedi számítógép-fiókkal.

Mivel a Windows 98 és 95 nem rendelkezik a Windows NT és 2000 fejlett biztonsági funkcióival, nem osztható ki nekik számítógépfiók, de be lehet jelentkezni róluk a tartományba, és használni lehet annak erőforrásait. *(Ha biztonságos tartományt akarunk kialakítani, ne használjunk benne Windows 9x-et futtató számítógépeket!!! Ezek ugyanis nem biztonságos hitelesítést használnak. Korábbi lapszámunkban olvasható ugyan megoldás arra, hogy ezeken az operációs rendszereken NTLMv2 hitelesítést alkalmazzunk, de ne feledjük, hogy magát a számítógépet és az azon levő adatokat ekkor sem védi semmilyen biztonsági rendszer.)*

A Windows 2000 többféle hitelesítési eljárást biztosít a hálózatra belépő felhasználók személyazonosságának minden kétséget kizáró megállapítására. Amikor a felhasználó belép a hálózatra, hitelesítési információkat kell szolgáltatnia a biztonsági rendszernek, amely így ellenőrizheti személyazonosságát, és eldöntheti, hogy milyen hozzáférést engedélyez a felhasználónak.

Az összes igény kielégítése érdekében a Windows 2000 támogatja a legtöbb szabványos hitelesítési eljárást *(például X.509 bizonyítványok, intelligens kártyák, Kerberos protokoll)*. Emellett a Windows 2000 támogatja az évek óta használt NTLM protokollt, és csatolófelületet biztosít a biometrikus *(ujjlenyomat, retina)* hitelesítési eljárások alkalmazásához.

A csoportos házirendekben a felhasználóknak és csoportoknak szerepkörük, és persze igényeink szerint megadhatók az alkalmazott hitelesítési eljárások. Például érzékeny adatokat használó vezetőinket intelligens kártyák, Internetes vásárlóinkat X.509 bizonyítványok használatára kötelezhetjük, a „mezei" felhasználóknál pedig továbbra is nyugodtan alkalmazhatjuk az NTLM hitelesítést. A hitelesítési eljárástól függetlenül a Windows 2000 mindig az Active Directory-t használja az eljárás során kapott adatok ellenőrzésére. Ha a hitelesítés sikeres, és a hitelesítési eljárás során szolgáltatott adatokhoz felhasználói fiók rendelhető hozzá, a Windows 2000 ellátja a felhasználót meghatalmazásokkal, melyek a hálózat erőforrásaihoz való hozzáférésre használhatók.

Hozzáférésszabályozás

A Windows 2000 a hozzáférésszabályozást az objektumokhoz *(fájlok, nyomtatók, szolgáltatások)* rendelt biztonsági leírókkal valósítja meg. Egy objektum biztonsági leírója egy ACL-t tartalmaz, ami meghatározza, hogy mely felhasználónak milyen jogosultsága van. Ez adja meg azt is, hogy mely objektumelérési eseményeket *(például a fájl írása)* kell naplózni. Az objektum tulajdonságainál beállíthatók a jogosultságok, és naplózhatók a felhasználók objektum-elérései.

A Windows 2000 megújított biztonsági rendszerében az objektumok egyes attribútumaihoz való hozzáférést is szabályozhatjuk. Például az objektum biztonsági leíróinak megfelelő beállításával a felhasználók számára látható lehet egy alkalmazott neve és telefonszáma, de az otthoni címe nem. Az objektum ACL-jét használva a Windows 2000 összehasonlítja az ügyféltől származó információt *(személyazonos-*

ság, csoporttagság) az objektumról szóló információkkal *(ACL)*, és eldönti, hogy a felhasználónak megvannak-e a jogosultságai az objektumon *(például egy fájlon)* a kért művelet végrehajtásához *(például írás/olvasás)*. A hozzáférés ellenőrzése a Windows 2000 biztonsági alrendszerén belül, kernel módban történik. Az összehasonlítás eredményétől függően az ügyfél értesítést kap a szolgáltatás megkezdéséről, vagy annak megtagadásáról.

A hozzáférésszabályozás finomhangolása

A biztonsági beállítások nagyobb rugalmassága érdekében az Active Directory lehetővé teszi a címtárobjektumok hozzáférésszabályozásának finomhangolását. A címtárobjektumok gazdagon felcicomázhatók jellemzőkkel, s ezeket akár egyesével is védhetjük. Egy felhasználónak vagy csoportnak akár több száz attribútuma is lehet *(például telefonszámok, főnök, csoporttagság)*. A felhasználói fiók kiválasztott tulajdonságaira, vagy attribútumaira más-más jogosultságok állíthatók be. A naplózás is tulajdonságszinten állítható be.

A rendszerfelügyelet átadása

Mivel a vállalatok több alkalmazott között oszthatják el a tartományfelügyeleti teendőket, az Active Directory biztosítja az erdőn, vagy tartományon belül a felügyeleti feladatrészek átadását. A szervezeti egységek létrehozásával a tartományfa bármely szintjére leoszthatók a felügyeleti teendők, úgy, hogy egy szervezeti egységbe helyezzük azokat az objektumokat, melyek felügyeletét másra akarjuk bízni, majd az OU felügyeletét a kívánt személyre vagy csoportra bízzuk.
OU szinten a rendszergazdák jogot adhatnak például csoportok létrehozására, a csoportok taglistáinak szerkesztésére, vagy számítógépfiókok tartományhoz való hozzáadására. A csoportos házirendek beállításai és a felhasználói csoportok használata lehetővé teszik a rendszergazdáknak a kiosztott felügyeleti terület pontos meghatározását. Például a felhasználó-rekordok módosításának területe kiosztható egy felhasználói csoportnak, de a csoport fennhatósága egy kiválasztott felhasználókból álló részhalmazra korlátozható.
Emellett, a hozzáférés-szabályozás finomhangolásával leszűkíthető a kiosztott felügyeleti feladatok köre. Például a kiválasztott csoportnak nem kell a felhasználói fiókok rekordjaihoz mindenre kiterjedő jogokat adni, amikor elég a jelszavak törlését engedélyezni.

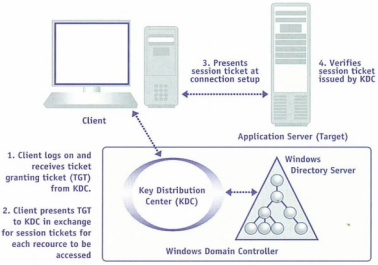
A Kerberos hitelesítési protokoll

A Windows 2000 elsődleges felhasználóhitelesítési módszerként az Internet szabványnak megfelelő Kerberos V5 protokollt *(RFC 1510)* használja. A Kerberos protokoll kölcsönös hitelesítési eljárás, mely a kiszolgáló és az ügyfél között, a köztük megnyíló hálózati kapcsolat létrehozása előtt zajlik le.
A Kerberos hitelesítés jegyeken *(ticket)* alapul. Amikor az ügyfél bejelentkezik a Windows 2000 tartományba, kap egy jegyet, mely arra szolgál, hogy azonosítsa a hálózati erőforrás számára az ügyfelet, és az ügyfél számára a hálózati erőforrást. Ennek megvalósításához a Kerberos shared secreten *(jelszavakon)* alapuló hitelesítést használ. A módszer alapját képező elgondolás igen egyszerű: ha csak a két fél tudja a titkot, bármelyik megbizonyosodhat a másik személyazonosságáról azzal, ha meggyőződik róla, hogy a másik tudja a titkot.

A Kerberos protokoll esetén az ügyfél és a kiszolgáló is nyilvántartásba kerül a Kerberos hitelesítő-kiszolgálón. A Kerberos hitelesítést használó ügyfelek a felhasználó jelszaván alapuló, titkosított információt küldenek a Kerberos kiszolgálónak, amely így megbizonyosodhat a felhasználó személyazonosságáról. A kiszolgáló ugyanígy küld információt az ügyfél Kerberos szoftverének, amely így megbizonyosodhat arról, hogy a kiszolgáló tényleg az-e, akinek mondja magát. Ez a kölcsönös hitelesítési folyamat mind az ügyfelet, mind a kiszolgálót megvédi a másik felet megszemélyesítő rosszindulatú „kalózoktól".
A Kerberos fejlődést jelent a Windows NT 4.0 hitelesítési folyamatához *(NTLM)* képest, ahol az ügyfélnek minden egyes erőforrás elérésekor külön hitelesíteni kellett magát. A Kerberos leváltja az NTLM protokollt, de a régi rendszerekkel való együttműködés érdekében a Windows 2000 támogatja az NTLM-t is. A teljes körű Kerberos protokoll-támogatás nemcsak a Windows 2000 rendszerek esetén biztosít gyors, egyszeri bejelentkezést, hanem minden olyan rendszernél, amely támogatja ezt.

A Kerberos hitelesítési folyamat

A Kerberos hitelesítési protokoll az ügyfelek és a kiszolgálók azonosítóinak ellenőrzését, és megfelelő helyre való eljuttatását végzi. Amikor a felhasználó bejelentkezik a Windows 2000 tartományba, a Windows 2000 keres *(DNS, oh DNS, légy jó mindhalálig!)* egy Active Directory kiszolgálót, és egy Kerberos hitelesítési szolgáltatást, és az ügyfél azonosítóját eljuttatja a Kerberos szolgáltatáshoz. Az ügyfél a jelszóból származó, kiszolgáló által ellenőrizhető kulccsal, vagy *(intelligens kártya használata esetén)* a titkos kulccsal *(melynek nyilvános fele az Active Directory-ban megtalálható)* titkosított információval hitelesíthető. Az ügyfél hitelesítési információinak ellenőrzése után a kulcselosztó központnak *(Key Distribution Center - KDC)* nevezett Kerberos szolgáltatás ad a felhasználónak egy további jegyek kérésére alkalmas jegyet *(ticket-granting ticket - TGT)*, amely ezután az ügyfél azonosítására szolgál, amikor a hálózati erőforrások eléréséhez további jegyeket igényel. *(A TGT használatával a KDC-nek kevesebbszer kell az ügyfélről szóló információkat kérnie.)*
Bár a fent leírt folyamat nagyon bonyolultnak tűnik *(és az is)*, a felhasználónak csak a jelszavát kell beírnia.
Az alábbi ábrán látható az ügyfél, a KDC és a kiszolgáló Kerberos protokoll használata közbeni kapcsolata.



A Kerberos hitelesítés folyamata.

(Folytatjuk...)

Transact SQL

(IV. rész)

Bevezetés

Még be sem fejeződött az SQL Server 7 fejlesztése, és máris több oldalas volt az SQL Server fejlesztő csapat „kívánságlistája", azaz, hogy a megoldásszállító fejlesztők milyen funkciókat szeretnének látni a következő SQL Server verzióban. Ennek eredményeként született – az XML támogatás mellett – az SQL 2000 legnagyobb újítása a felhasználói függvények formájában. A cikkben nagyon tömören megnézzük a téma elméleti hátterét, hogy azután megírjunk néhány függvényt, amelyek segítségével szövegeket manipulálunk, megírjuk a Basic Split függvény SQL párját, megtanulunk körlevelet küldeni felhasználói függvényekkel, és kifejlesztünk egy behatolásjelző programot. Hosszú, de nagyon izgalmas rész lesz ez cikksorozatunkban, érdemes végigolvasni!

Minden, amit tudni akartál a felhasználói függvényekről, de nem merted megkérdezni

Ez a rész a felhasználói függvények lelkivilágával, formai és működésbeli tulajdonságaival foglalkozik. Aki tudja, miért fontos a determinizmus kérdése a függvényeknél, az nyugodtan ugorjon az utolsó fejezetre, ahol fokozatosan bonyolódó példákat találhat. Aki nem, az tartson velem a következő részekben is.

Az SQL Serverben nagyon sok beépített függvény található *(lásd cikksorozatunk előző része)*, azonban ezek nyilvánvalóan nagyon általános függvények, mint például a *(ruhaiparból ismert)* LEN függvény, ami egy szöveg hosszát adja vissza. Nagyon jók ezek a beépített függvények, köszönjük őket, de a gyakorlati problémák megoldásához – pont az általános voltuk miatt – nem elegek. Mint építőkockák kitűnőek, de hogyan építünk belőlük várat? Nos, hosszú várakozás után a Microsoft elékészítette a habarcsot, megalkotta a felhasználói függvényeket, így most már semmi akadálya, hogy megalkossuk a saját PAMUT vagy a GYAPJU nevű függvényeinket, amelyek belső működését mi írhatjuk elő. Egyszerűen megfogalmazva a felhasználói függvény olyan Transact SQL utasítások sorozata, amelyeket azért csomagolunk egybe, hogy több helyen is felhasználhassuk. Nagyon jól kiegészítik a tárolt eljárásokat, mert minden olyan helyen felhasználhatjuk őket, ahol a beépített függvényeket is, azaz ahol a tárolt eljárásokat legtöbbször nem. A legegyszerűbb példa erre a SELECT-ben való felhasználás. Például, ha van egy osszeadas nevű függvényünk, akkor azt felhasználhatjuk két oszlopban található számok összeadására, a SELECT utasítás részeként:

```
SELECT osszeadas(Ár, ÁFA), Termék FROM ...
```

Ennél kevésbé kézenfekvő helyeken is használhatjuk a függvényeinket: WHERE feltételben, HAVING-ben, CHECK CONSTRAINT-ekben, DEFAULT CONSTRAINT-ekben, számított oszlopok képzésében. Mindenhol működnek, ahol a szerver valamilyen kifejezést vár *(mint a>b, c=4 vagy 2x2=5)*.

Azok kedvéért, akik nem szeretnek tömény oldalakat kódok nélkül látni, megmutatom az előbbi függvény deklarációját. Részletes magyarázatot a cikk második felében talál a lelkes Olvasó.

```
CREATE FUNCTION osszeadas
(
  @a INT,
  @b INT
)
  RETURNS INT
BEGIN
  RETURN @a + @b
END
```

Az SQL Server a függvényeket sokszor tranzakciók, illetve SELECT, UPDATE, satöbbi utasítások kellős közepén hívja meg. Emiatt rendetlen az a függvény, ami menet közben módosítja egy tábla tartalmát, miközben egy SELECT *(ami őt hívta meg)* éppen dolgozik rajta – nos ilyen esetben nagy lárma és kalamajka támadhatna. Az SQL Server azonban nem keresi a bajt, ezért megpróbálja megkötni a kezüket, hogy ne csináljunk felfordulást. Azaz a felhasználói függvényekben nem tehetünk meg akármit, csak a következőket:

- Definiálhatunk saját változókat és kurzorokat a DECLARE utasítással. Csak lokális kurzorokat készíthetünk így, globálisakat, azaz amelyek a függvény lefutása után is léteznének nem.
- A függvényben deklarált lokális változóknak értéket adhatunk *(naná, e nélkül akár ki is dobhatnák a függvényeinket)*.
- Használhatunk kurzorműveleteket, de csak úgy, hogy a FETCH utasítás eredményeit lokális változókba rakjuk el *(a kurzorokkal egy teljes cikk fog foglalkozni a következő hónapban)*.
- Bevethetjük a programfolyam-vezérlő utasításokat: if, then, for, while, goto, satöbbi. Ezek nélkül nem is lehetne egy komolyabb függvényt megírni.
- Alkalmazhatjuk az összes adatmódosító utasítást *(INSERT, UPDATE, DELETE)*, ha azok csak lokális táblákon végeznek műveleteket. Ebből következően nem lehet módosítani külső táblákat. Természetesen lekérdezésekben szerepelhetnek.
- Meghívhatunk külső tárolt eljárásokat *(Extended Stored Procedure)* az EXECUTE utasítással. „Hagyományos" tárolt eljárásokat nem lehet meghívni belőlük, hisz azokból már könnyedén beavatkozhatnánk a „külvilágba".

Látható, hogy minden pontban arról van szó, hogy megtehetünk szinte bármit, amit csak akarunk, de csak lokálisan, azaz a függvény nem avatkozhat be a külvilágba. Van egy kis szemétdombunk, ott kapirgáljunk. Bár az utolsó pont, azaz, hogy külső tárolt eljárásokat is meghívhatunk, azért egy nagyon tág fogalom. Mert mit csinálhat egy külső tá-

rolt eljárás? Bármit! Amit akar. Azaz például megteheti azt, hogy visszafelé nyit egy kapcsolatot a kiszolgálóra, és azon keresztül megváltoztatja azt a táblát, amiben éppen dolgozik a kódunk a függvény hívása során. De ez általában már túlmutat a normális használaton. Megtehették volna a fejlesztők, hogy teljesen letiltják a külső eljáráshívásokat, de akkor meg elestünk volna olyan nagyszerű lehetőségektől, mint külső parancsok meghívása *(xp_cmdshell)*, levélküldés *(xp_sendmail)* vagy event log írás *(xp_logevent) (és még sok egyéb hasznos funkció)*. Az imént felsorolt három külső tárolt eljárás azonban pont olyan, aminek nem szabadna lefutni egy függvényben. Miért? Azért mert egy függvény nem változtathatja meg globálisan a rendszer állapotát. A rendszeren nem csak az SQL Server belső lelkivilágát értjük, hanem az egész világot. Így például az xp_cmdshell segítségével akár le is formázhatjuk kollégánk merevlemezét. Fogadjunk, hogy megváltozik a kolléga *(lelki)*állapota. :) Azaz ezeket a külső tárolt eljárásokat nem szabadna meghívni egy felhasználói függvényből, amire nyomatékosan fel is hívja a figyelmet a dokumentáció *(Books Online)*. Azonban, a fordító egy szót sem szól, ha olyan függvényt írunk, amiben felhasználjuk a veszélyes tárolt eljárások valamelyikét! Ezt még ki fogjuk használni a cikk végén található programokban. Pont olyan ez, mint a C programozás: ha meggondoltan csináljuk, miénk a világ. Ha nem, akkor csak General Protection Fault-okat generálunk.

A nemdeterminisztikus jövő

Vannak még más problémás elemek is, amelyeket bizonyos esetekben szintén nem szabad használni függvényekben. Ezek a nemdeterminisztikus függvények. Mik is ezek? Ők a függvények azon fajtái, amelyeknek a működése vagy az általa visszaadott érték időben vagy a szerver állapotától függően nem megjósolható módon változik. Azaz ugyanazokkal a paraméterekkel meghívva egyszer a-t mond, másszor b-t. A legegyszerűbb példa erre a GetDate() beépített függvény, ami a pillanatnyi időt adja vissza *(a GetTime szerencsésebb név lett volna)*. Ez minden egyes meghívás pillanatában más értéket ad vissza, legalábbis addig, amíg jár a gépünkben a kvarckristály. A fordítóprogram nem engedi meg, hogy ilyen nemdeterminisztikus beépített függvényeket helyezzük el a saját függvényeinkben. Például a következő függvény törzsre: RETURN RAND*(10)* a fordító az „Invalid use of 'rand' within a function." hibaüzenettel válaszol.

Miért ilyen problémás pont a determinizmus kérdése az SQL Serverben? Azért, mert vannak benne olyan új szolgáltatások, amelyek nem tudnának helyesen működni a „bizonytalan" nemdeterminisztikus függvényekkel. Két helyen nem lehet felhasználni a nemdeterminisztikus függvényeket:

- Indexelt számított oszlopokon, azaz, ha olyan oszlop ra szeretnénk indexet készíteni, amelynek érékei egy másik *(egy vagy több)* oszlopból származnak, és a számított érték valamilyen nemdeterminisztikus függvényen alapul.
- Olyan nézetekben, ahol a nézetre clustered indexet szeretnénk használni.

A két megszorítás alapján már eléggé érthető, hogy miért kell foglalkozni a determinizmus kérdésével. Mindkét esetben indexet építünk táblában található adatokra. Próbált már valaki megülni egy vásári bikát? Nem egyszerű. Hasonló módon az SQL Server sem tud indextáblát építeni olyan adatokra, amelyek minden pillanatban változnak. A clustered index az adatok fizikai sorrendjét határozza meg. Ezen a héten így legyek sorban az adatok, a következő héten meg másképp, csak azért, mert meggondolta magát a transzformáló függvény? Na nem, ez nonszensz lenne. Ezért nem is tehetünk ilyet.

Ragaszkodás a barátokhoz

Egyetlen apró fogalom maradt már csak hátra, hogy ténylegesen megírhassuk első függvényünket. Ez a séma-kötés fogalma. A felhasználói függvények igen erősen kötődnek azokhoz a táblákhoz, és egyéb objektumokhoz, amelyekre hivatkoznak. Ha azok módosulnak anélkül, hogy erről a függvény tudna, akkor a kapcsolatuk vége barátságtalan lesz, és a függvény nem fog jól működni. Azért, hogy a jó viszonyban ne következhessen be szakadás, a függvény létrehozásakor *(CREATE FUNCTION)* megadhatjuk, hogy a függvény legyen hozzákötve az általa használt objektumokhoz. Ezt az SQL Server megjegyzi, és nem engedi módosítani vagy törölni az ily módon leláncolt objektumokat. A kötés jelzését a RETURNS és a függvény törzsét kezdő BEGIN közé kell írni:

```
...
RETURNS ...
WITH SCHEMABINDING
BEGIN
...
```

Függvénytípusok

Háromféle felhasználói függvénytípust hozhatunk létre az SQL 2000-ben:

- Skaláris függvények, melyeknek visszatérési értéke skaláris, azaz egy érték *(scalar functions)*
- Egy utasításból álló, tábla visszatérési értékű függvények *(inline table valued functions)*
- Több utasításból álló, tábla visszatérési értékű függvények *(multi statement table valued functions)*

Az utóbbi két fajta nagyon hasonlít egymásra, mint ez a részletes tárgyalásból hamarosan kiderül.

Skaláris függvények

A skaláris függvények nagyon egyszerűek: kapnak néhány paramétert, azokon végeznek valamilyen művelet, majd az eredményt egy skaláris értékként visszaadják. Azaz visszaadnak egy számot, egy szöveget, egy dátumot satöbbi. Leginkább a procedurális nyelvek függvényeihez hasonlítanak. Rutinos tárolt eljárás programozók! A felhasználói függvényeknek nincsenek kimeneti paramétereik! Azaz nem lehet valamelyik paramétert megjelölni, hogy az visszafelé fog majd valamilyen információt szolgáltatni a hívónak. Ezt a lehetőséget azért kellett bevezetni a tárolt eljárásoknál, mert azok csak egy egész számot tudnak visszaadni visszatérési értékként, így nem tudtunk volna például egy dátumot visszaadni a hívónak. Erre szolgáltak a kimeneti paraméterek. Hogy teljesen érthető legyen, álljon itt egy tárolt eljárás, amelynek a harmadik paramétere kimeneti paraméter:

```
CREATE PROCEDURE osszead
  @a INT,
  @b INT,
  @c INT OUTPUT
AS
SET @c = @a + @b
--Eddig a tárolt eljárás deklarációja.
--Látható, hogy egy tárolt eljárásban nem
--kötelező a visszatérési értéket megadni
--Azaz lehetne egy záró RETURN ..., de
--nem szükséges, mert most nem használjuk
--fel a visszatérési értéket.

DECLARE @osszeg INT
--hívjuk meg

EXECUTE osszead 1,4, @osszeg OUTPUT
SELECT @osszeg
5  --Működik!
```

Nos, kimeneti paraméter nincs a felhasználói függvényekben. Viszont segítségükkel sokkal egyszerűbben meg lehet fogalmazni az előbbi problémát:

```
CREATE FUNCTION osszeadas(
  @a INT,
  @b INT)
  RETURNS INT
BEGIN
  RETURN @a + @b
END

SELECT dbo.osszeadas(1,4)
```

Azért ez sokkal természetesebb, mint a tárolt eljárásos változat. De azért szedjük csak szét ízekre a függvény deklarációt! A CREATE FUNCTION jelzi, hogy ez egy felhasználói függvény lesz. Ezután jön a függvény neve. Általában a függvényeknek vannak paramétereik, ezeket zárójelben soroljuk fel a függvény neve után. A @ nem opcionális, nem esztétikai okokból raktam bele, vagy azért, mert ettől olyan tudományos lesz, hanem azért, mert Transact SQL-ben minden változót kötelező @-al kezdeni. A paraméter neve után meg kell adni az ő típusát. Itt majdnem az összes, a kiszolgáló által támogatott adattípust fel lehet használni, egykét elvarázsolt image, text vagy cursor típust kivéve. A RETURNS után kell definiálni a visszatérési érték adattípusát. A kötöttségek ugyanazok, mint a paramétereknél, azaz csak „normális" változókat használhatunk. A függvény törzsét, ahol az általunk megálmodott funkcionalitást írjuk le, a BEGIN és END kulcsszavak közé kell elhelyezni. Ennyi.
Mondja azt valaki, hogy bonyolultak a felhasználói függvények! Ha a fenti mintapélda kéznél van, minden problémát csuklóból megoldunk. Persze enyhe túlzással, és ha egy kimeneti érték elég a feladat leírásához. :)
Még egy fontos tudnivaló. A skaláris visszatérési értékű függvényekre minimum 2 tagú névvel kell hivatkozni. Azaz legalább a függvény tulajdonosát meg kell adnunk ahhoz, hogy az SQL Server felismerje a függvényünket. Ennek megfelelően, a:

```
SELECT osszeadas(1,4)
```

hibát fog jelezni. Helyesen:

```
SELECT dbo.osszeadas(1,4) vagy
SELECT Northwind.dbo.osszeadas(1,4)
```

De mi van, ha több értéket kell visszaadnunk? Mi van, ha ráadásul azt sem tudjuk, hogy igazából hány kimeneti értékünk lesz, mert azt a táblákban található információk pillanatnyi állapota szabja meg? Ebben az esetben kapaszkodunk a tábla kimenetű felhasználói függvényekbe. *(A továbbiakban nem írom ki mindenhol a felhasználói jelzőt, de ott van.)*
Ezt értsük úgy, hogy, ha a skaláris függvények egy skaláris mennyiséget adnak vissza, akkor a tábla kimenetűek meg egy táblát? Igen. De, hát nincs is ilyen adattípus az SQL Server 7-ben! Abban tényleg nincs, de az SQL 2000-ben van. És nagyon szeretjük is őket. Képzeljük el: van egy olyan változótípusunk, ami akár egy tízmillió sorból és huszonhat oszlopból álló teljes táblát el tud tárolni. Csoda, hogy szeretjük? Ez a tábla *(table)* adattípus.
Miért olyan szenzációs ez? Eddig is létre lehetett hozni átmeneti táblákat, és azokba is lehetett ideiglenes eredményeket beleírni. Persze, de a tábla adattípus felhasználásával egyrészt átláthatóbban, a természetes gondolkodáshoz közelebb álló kódot hozhatunk létre, másrészt olyan dolgokat is megvalósíthatunk, amelyeket korábban csak nagyon trükkösen vagy sehogyan sem tudtunk megtenni.
Hol használhatjuk fel a tábla kimenetű függvényeket? Minden olyan helyen, ahol eddig egy táblát adhattunk meg. Azaz leginkább a FROM záradék után.

```
SELECT cica, egér
FROM AzElsoTablaFuggvenyem('sajt')
```

Paraméterezett nézetek felhasználói függvényekkel, avagy az egy utasításból álló, tábla visszatérési értékű függvények

Mit tudtunk tenni SQL7-ben, ha azt kérték tőlünk, hogy kellene egy nézet, ami a megrendeléseket listázza ki, de úgy, hogy megadhassuk paraméterként, hogy melyik megrendelőhöz tartozó tételeket kívánjuk látni. Azaz valami ilyesmit akartunk írni:

```
CREATE VIEW OrdersByCustomer(
  @CustomerID varchar(5))
AS
SELECT * FROM Orders
WHERE
  CustomerID = @CustomerID
--Nem működik, nem fordul le!
```

Nos, ilyen nincs SQL7-ben, sőt SQL2000-ben sem! Ilyenkor jön a felmentő sereg, az egy utasításból álló, tábla visszatérési értékű függvény. Az előbbi majdnem működő nézetet könnyen átalakíthatjuk egy tábla visszatérési értékű függvénnyé, ami már az elvárt funkciót valósítja meg:

```
CREATE FUNCTION OrdersByCustomer(
@CustomerID varchar(5))
RETURNS TABLE
AS
  RETURN (
  SELECT * FROM Orders
  WHERE
    CustomerID = @CustomerID)
--Teszt:
SELECT CustomerID, ShippedDate
FROM OrdersByCustomer('THEBI')
THEBI       1996-09-27
THEBI       1997-11-05
THEBI       1998-01-09
THEBI       1998-04-03
```

Mit kellett tennünk, hogy a majdnem-működő, de azért mégiscsak-ramaty nézetünkből egy jólfésült függvény legyen? A CREATE VIEW helyett CREATE FUNCTION-t írunk. Jelezzük, hogy a függvény visszatérési értéke nem holmi skalár, hanem tábla: RETURNS TABLE. Látható, hogy nem specifikáltuk az eredménytábla szerkezetét, csak egyszerűen megadtuk, hogy tábla lesz. Emiatt van, hogy az ilyen típusú függvényekben csak 1, azaz egy darab SELECT utasítás lehet, hiszen annak az eredményhalmaza határozza meg a visszatérési értékként generálódó tábla típusát. Pontosabban lehet benne egymásba ágyazva több SELECT utasítás is, de a teljes lekérdezés csak egy eredményhalmazt adhat vissza. Azaz pont ugyanaz a helyzet, mint a nézeteknél volt.

Több utasításból álló, tábla visszatérési értékű függvények

Bonyolultabb esetben a visszatérési érték nem állítható elő egyetlen SELECT utasítás segítségével, ilyenkor kell használnunk ezt a függvénytípust. Mivel ilyenkor már nem egyértelmű, hogy melyik lekérdezés kimenetét szeretnénk visszaadni, explicit deklarálnunk kell a visszatérési értékként szolgáló tábla szerkezetét egy tábla típusú változóként. A változót INSERT utasítások segítségével feltöltjük *(akárhány lépésben)*, és a RETURN utasítás ezt fogja visszaadni a hívónak. Erre a függvénytípusra összetettebb példákat a következő fejezetben találhatunk.

Praktikus felhasználói függvények

Annak örömére, hogy megkaptuk a felhasználói függvényeket, használjuk ki az alkalmat, és írjuk meg néhány olyan probléma megoldását, ami a minden napi fejlesztések során sokszor előjött-előjön.

Szövegelőfordulás számláló

Gyakori feladat, hogy egy szövegben meg kell keresni azt, hogy egy másik szöveg hányszor fordul elő benne. Milyen algoritmust használjunk? Az egyik legegyszerűbb, bár nem feltétlen a leghatékonyabb módszer az, hogy a keresendő szöveg minden egyes előfordulását cseréljük ki egy üres sztringre a „nagy" szövegben *(amiben keresünk)*, és az eredeti szöveg hosszából vonjuk ki az így kapott szöveg hosszát. Ezt az eredményt már csak le kell osztani a keresendő szöveg hosszával, hisz minden csere után ennyivel csökkent az „nagy" szöveg hossza. Hogy néz ez ki függvényként? *(A bemutatott példa egy nagyon nem normalizált adatbázis, annyira nincs formában, hogy még 0. normál formában sincs. Csak demócélokra szolgál, nem adatbázistervezési minta!)*

```
CREATE FUNCTION StringOccur
(
  @cString  AS varchar(8000),
  @cLookFor AS varchar(100)
)
RETURNS int
AS
BEGIN
  RETURN
  (LEN(@cString)
  -LEN(REPLACE(@cString, @cLookFor, '')))
  / LEN(@cLookFor)
END
--Teszt tábla
CREATE TABLE T1
(
  cMenu varchar(100) NOT NULL
)
--Tesztadatok
INSERT INTO T1 VALUES('Töltött káposzta, Almáspi
↳ te, Diósbejgli')
INSERT INTO T1 VALUES('Pulykarizottó, Mákosbejgli,
↳ Diósbejgli')
INSERT INTO T1 VALUES('Székelykáposzta, Rántottbé
↳ ka, Mákosbejgli')
INSERT INTO T1 VALUES('Stefániasült, Káposztáspi
↳ te, Túrósbejgli')

SELECT
  cMenü AS Menü,
  dbo.StringOccur(cMenu, 'káp') AS
  Káposztásfogás,
  dbo.StringOccur(cMenu, 'bejgli') AS
  Bejglitartalom,
  dbo.StringOccur(cMenu, 'Mákos') AS
  Mákosfogás
FROM T1
```

A kimenet *(nyomdai okokból táblázatban)*:

Menü			Káposztás fogás	Bejgli tartalom	Mákos fogás
Töltött-káposzta	Almás-pite	Diós-bejgli	1	1	0
Pulyka-rizottó	Mákos-bejgli	Diós-bejgli	0	2	1
Székely-káposzta	Rántott-béka	Mákos-bejgli	1	1	1
Stefánia-sült	Káposztás-pite	Túrós-bejgli	1	1	0

A függvény elég trükkös, megér néhány szót. „Izomból" nekifutva hogyan oldanánk meg a példát? Egy ciklusban keresnénk a keresendő szöveg előfordulásait a „nagy" szö-

vegben, mindig a következő pozíción *(karakteren)* folytatva a „nagy" szövegben, mint ahol az előző lépésben abbahagytuk. Ehhez a megoldáshoz ciklust kellene szerveznünk, ami jelentősen megbonyolítaná a megoldást. Ehhez képest a fenti függvény sokkal egyszerűbb, hisz a bonyolultabb funkcionalitást átadtuk a REPLACE függvénynek.
Más kérdés, hogy az imént vázolt algoritmus és a fenti algoritmus más kimenetet ad például a következő szövegekre:

```
-A fenti függvény (a LEN-es)
SELECT dbo.StringOccur('bababababa', 'baba')
2
```

Ezzel ellentétben, ha lenne egy függvényünk, ami az imént említett módon működne, akkor a visszaadott érték 4 lenne, hisz:

```
bababababa
bababababa
bababababa
bababababa
```

A kérdés az, hogy átlapolhatják-e egymást a keresendő szöveg előfordulások? Ha nem, akkor jó a fenti függvény, ha igen, akkor meg kell írni a másik verziót. Ezt a konkrét feladat határozza meg.

Szövegdarabolás

Visual Basic programozók gyakran keresik a Basic Split függvény Transact SQL párját. Mindhiába, mert nincs. A Split egy nagyon hasznos függvény, arra való, hogy egy szöveget valamilyen határoló karakter mentén feldaraboljon, és a darabokat visszaadja egy tömbben. Segítségével egy mondatot feldarabolhatunk szavakra, egy vesszővel elválasztott listát listaelemekre, satöbbi.
Mivel nincs ilyen függvényünk, implementáljunk egyet! Az első akadály, amibe rögtön beleütközünk az, hogy a TSQL-ben nincs tömb típus. Emiatt a függvény kimenete tábla típusú kell, hogy legyen, mert skalárban nem tudunk visszaadni több elemet. Azaz, írjunk egy olyan függvényt, ami a megadott szöveg és az elválasztó karakter ismeretében szétdarabolja a szöveget, és egy táblában visszaadja a szövegkomponenseket. Legyen a visszaadott mező neve cStringPart!

```
CREATE FUNCTION Split
(
  @cOriginalString AS varchar(8000),
  @cDelimiter char(1))
  RETURNS @SplitString table
(
  nID int IDENTITY(1,1) NOT NULL,
  cStringPart varchar(8000) NULL)
AS
BEGIN
  DECLARE @nNumberOfDelimiters AS int
  --Számoljuk  meg, hány határoló
  --karakterünk van.
  --Használjuk fel az előzőleg megírt
  --szöveg-előfordulás számláló
  --függvényünket.
  SET @nNumberOfDelimiters =
  dbo.StringOccur(@cOriginalString, @cDelimiter)
  DECLARE @i AS int
  SET @i = 0
  --Végigmegyünk az összes szövegdarabon
  WHILE @i < @nNumberOfDelimiters
  BEGIN
  --A forrásszöveg baloldalából
  --kivágjuk az ott található szöveget
  --a határoló karakterig,
  --és beszúrjuk az eredménytáblába.
    INSERT INTO
      @SplitString
    SELECT
      LEFT(@cOriginalString,
      CHARINDEX(@cDelimiter,
      @cOriginalString)-1)
    --Levágjuk a már feldolgozott
    --szöveget, így az elején mindig
    --megtaláljuk a következő darabot.
    SET @cOriginalString =
    SUBSTRING(@cOriginalString,
    CHARINDEX(@cDelimiter,
    @cOriginalString)+1, 8000)
    --Továbblépünk a következő darabra

    SET @i = @i + 1
  END
  --Az utolsó határoló karakter után
  --még maradt egy darab, azt is
  --szúrjuk be az eredményhalmazba.
  INSERT INTO
    @SplitString
  VALUES
    (@cOriginalString)
  --Összeállt az eredménytábla, ideje
  --visszaadni azt a hívónak.
  --Itt már nem kell jelezni, hogy mit
  --adunk vissza, mert az már a RETURNS-
  --nél (az elején) megtettük.
  RETURN
END
--Teszt

SELECT
  T1.*
FROM
  Split('Dec 24,Dec 25,Dec 26,Dec 31',',')
  AS T1
--Eredmény:

nID          cStringPart
1            Dec 24
2            Dec 25
3            Dec 26
4            Dec 31
```

Látható, hogy a függvények egymásba ágyazhatók, éppúgy, mint a tárolt eljárások. Ezzel élve nagyon jól átlátható, moduláris programokat írhatunk az SQL Server-re.

Spam-re fel!

Az SQL Server segítségével könnyedén írhatunk körlevele-

ket, ha van egy címzett *(áldozat)* adatbázisunk. A klasszikus megoldásban kurzort használnánk, és az xp_sendmail külső tárolt eljárást hívnánk meg egy ciklusban. Azonban a kurzorok használata elég körülményes dolog. Keressünk egy jóval egyszerűbb megoldást, természetesen a felhasználói függvények felhasználásával! A megcélzott függvény célja egyszerű: a bemenő paraméterekben meghatározott címzettnek elküldeni egy E-mail-t.

```
--Létrehozzuk a függvényt
CREATE FUNCTION SendMail
(
  @cRecepients AS varchar(200),
  @cSubject AS nvarchar(100),
  @cBody AS nvarchar(3000)
)
  RETURNS INT
BEGIN
  DECLARE @nResultCode INT
  EXEC @nResultCode = master..xp_sendmail
  @recipients = @cRecepients,
  @subject = @cSubject,
  @message = @cBody
  RETURN @nResultCode
END
--Egy teszttábla a „célszemélyekhez"
CREATE TABLE SpamTarget
(
  nID int NOT NULL IDENTITY(1,1),
  cTargetEmail nvarchar(400) NOT NULL,
  cFirstName nvarchar(100) NOT NULL,
  cLastName nvarchar(100) NOT NULL
)
--Két áldozat felvitele

INSERT SpamTarget VALUES
('Zsolt.Soczo@w2ktest1.vodafone.hu', 'Zsolt', 'Soczó')
INSERT SpamTarget VALUES ('ECudar@vadmalac.hu',
'Elek', 'Cudar')
--A levelek elküldése.
SELECT
  dbo.SendMail(cTargetEmail,
  cFirstName +
  '! Nyerj 99999999999 Forintot!',
  'Legyél te is milliomos!')
FROM
  SpamTarget
--Az áldozat által kapott levél:
From: sqlacc
Sent: Saturday, December 23, 2000 6:08 PM
To: Zsolt Soczo
Subject: Zsolt! Nyerj 99999999999 Forintot!
Legyél te is milliomos!
```

Anti-hacking toolkit v0.0

Utolsó és egyben legbonyolultabb függvényünkben egy állomány épség *(eredetiség)* ellenőrző programot írunk. A dupla KV ismét javasolt előtte, mert elég bonyolult lesz.
A feladat, hogy dolgozzunk ki egy olyan módszert, amely segítségével a védendő állományok bizonyos jellemzőit letároljuk, majd egy ellenőrző rutint lefuttatva ellenőrizzük, hogy a jellemző azonos-e a letárolt, háborítatlan értékkel. Ha nem, akkor a megfigyelt állományt egy rosszindulatú hacker vagy egy még rosszabb indulatú telepítőprogram módosította. A példa kedvéért az állomány méretet használjuk fel az ellenőrzéshez. Ennél sokkal profibb megoldás lenne, ha az állományokhoz kiszámítanánk valamilyen ellenőrző értéket *(pl. MD5 hash)*, és ezt tároljuk le az adatbázisban. Így sokkal nagyobb valószínűséggel lehetne jelezni, hogy megváltozott egy állomány.
Hogyan látnánk neki a feladat megoldásának? Mivel a fájlok méretét közvetlenül nem lehet lekérdezni az SQL Serverből, kénytelenek vagyunk kinyúlni a szerverből. Ehhez valamilyen külső tárolt eljárásra lesz szükségünk. Az `xp_cmdshell`, amivel külső parancsokat lehet végrehajtani, szinte kínálja magát, hogy bevessük erre a feladatra. Meghívunk egy VBScript programot, ami visszaadja a paraméterként megadott állomány hosszát. A külső parancs futtatásából származó sorokat, azaz a fájl hosszát az `xp_cmdshell` táblaként adja vissza, aminek az első sora tartalmazza a kívánt eredményt. Hogyan nyerjük ki ebből a táblából az első sort? Próbáljuk belerakni egy átmeneti *(temporary)* táblába, és abból leválogatni az eredményt. Ez azonban sajnos nem megy, mert függvényben nem használhatunk temporary táblát.
Próbáljuk meg belemásolni az `xp_cmdshell` kimenetét egy table típusú változóba. Ez sem megy, mert az `INSERT Tábla (EXECUTE xp_cmdshell...)` típusú parancs, *(ami egy tárolt eljárás kimenetét beszúrja egy táblába)* nem megy tábla típusú változóval, csak valódi táblával. „Valódi" táblát viszont nem módosíthat egy függvény. Van ebből kiút?
Utolsó kapaszkodóként elfeledkezünk az `xp_cmdshell`-ről, és megpróbáljuk felhasználni a külső COM komponensek meghívására szolgáló függvényeket. És ez bejön! A FileSystemObject COM komponens közvetlen meghívásával célba érünk. A kódhoz tartozó magyarázatot beleszőttem a kódba, mert kiragadva kevésbé érthető lenne.

```
--A fájlméret lekérdező függvény
--deklarációja.
CREATE FUNCTION GetFileSize
(
  @cFilePath AS nvarchar(4000)
)
 RETURNS INT
BEGIN
  DECLARE @nFileSize int
  DECLARE @hr int
  DECLARE @objFileSystem int
  DECLARE @objFile int
--Hozzunk létre a FileSystemObject-
--ből egy példányt.
  EXEC @hr = sp_OACreate
  'Scripting.FileSystemObject',
  @objFileSystem OUT
--Ha hiba történt egyszerűen visszatérünk
--egy hibakóddal. Ez azért nagyon csúnya,
--mert a kód későbbi részeiben
--bekövetkezett hiba esetén
--felszabadítatlan objektumok maradnak a
--memóriában!
```

```
--Így produkciós környezetben le
--kell kezelni a hibákat megfelelő módon.
--Ehhez az sp_OAGetErrorInfo külső tárolt
--eljárást lehet segítségül hívni.
  IF @hr <> 0 RETURN -1
--Meghívjuk a FileSystemObject GetFile
--nevű metódusát, ami visszatér egy
--File típusú objektummal. Ezt az
--@objFile változóban tároljuk el.
--A hívás paramétere az a fájlnév, aminek
--keressük a méretét (@cFilePath).
  EXEC @hr = sp_OAMethod @objFileSystem,
  'GetFile', @objFile OUT, @cFilePath
  IF @hr <> 0 RETURN -1
--A File objektum Size nevű
--tulajdonságának lekérdezésével
--megkapjuk a keresett állomány méretét.
--A kapott szám az @nFileSize-ba kerül.
  EXEC @hr = sp_OAGetProperty @objFile,
  'Size', @nFileSize OUT
  IF @hr <> 0 RETURN -1
--Felszabadítjuk a létrehozott
-- objektumokat.
  EXEC @hr = sp_OADestroy @objFile
  IF @hr <> 0 RETURN -1
  EXEC @hr = sp_OADestroy @objFileSystem
  IF @hr <> 0 RETURN -1
--Visszatérünk a kapott értékkel.
  RETURN @nFileSize
END
--Ebben a táblában tároljuk a fájlokat,
--és a hozzájuk tartozó méreteket.
CREATE TABLE FileAuthority
(
  nID int NOT NULL IDENTITY(1,1),
  cFileName nvarchar(3000) NOT NULL,
  nFileSize int NOT NULL
)
--Néhány tesztállomány. A -2-vel jelezzük
--hogy még soha nem olvastuk ki az adott
--fájl hosszát.
INSERT FileAuthority VALUES
('c:\winnt\notepad.exe', -2)
INSERT FileAuthority VALUES
('c:\boot.ini', -2)
INSERT FileAuthority VALUES
('c:\ntldr', -2)
INSERT FileAuthority VALUES
('c:\io.sys', -2)
INSERT FileAuthority VALUES
('c:\ntdetect.com', -2)
--Ezzel a tárolt eljárással
--töltjük fel a táblázat méret mezőit.
CREATE PROCEDURE CalculateFileSize
AS
  UPDATE
    FileAuthority
  SET
    nFileSize = dbo.GetFileSize(cFileName)
--Futtassuk le. Ettől kezdve van egy
--táblázatunk arról, hogy melyik fájlnak
--milyen hosszúnak kell lenni.
EXEC CalculateFileSize
--Nézzük meg, mit tartalmaz a táblánk!
--SELECT * FROM FileAuthority
FileName                          FileSize
c:\winnt\notepad.exe              50960
c:\boot.ini                       195
c:\ntldr                          214416
c:\io.sys                         0
c:\ntdetect.com                   34468
--Ezzel a tárolt eljárással össze
--lehet hasonlítani a letárolt és
--a futtatás pillanatában aktuális
--állományhosszakat.
--Csak azokat listázza ki, amelyeknél
--eltérés van a két érték között.
CREATE PROCEDURE CheckFileSize
AS
  SELECT
    nID,
    cFileName,
    nFileSize AS OriginalSize,
    dbo.GetFileSize(cFileName) AS
    CurrentSize
  FROM
    FileAuthority
  WHERE
    nFileSize <>
    dbo.GetFileSize(cFileName)
--Tesztképpen megváltoztattam a boot.ini
--fájl hosszát.
--Ellenőrizzük le!
EXEC CheckFileSize

A kimenet:
nID cFileName    OriginalSize CurrentSize
2   c:\boot.ini 195          197
```

Hoppá, a BOOT.INI-t valaki megváltoztatta! Működik az ellenőrző eljárásunk.

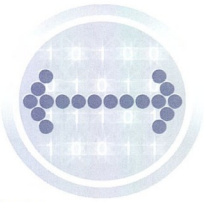
Zárszó

A cikkben felhozott példák két igen fontos dologra világítanak rá. A felhasználói függvények felhasználásával nagyon sok, a gyakorlatban felmerülő feladatot oldhatunk meg, amelyeket eddig csak átmeneti táblák és kurzorok felhasználásával tudtunk megtenni, általában nagyon bonyolultan, és nehezen olvasható módon. A másik tanulság, hogy a külső tárolt eljárások segítségével sok olyan feladatot is megoldhatunk az SQL Server segítségével, amelyeket általában más programnyelven megírt programokkal *(Visual Basic, stb.)* végeztettünk el.
A következő részben a kurzorokról lebbentem fel a fátylat, megnézzük, hogy körültekintő felhasználásukkal a felhasználói függvényekhez hasonlóan igen bonyolult feladatokat is elég egyszerűen megoldhatunk.

Soczó Zsolt
MCSE, MCSD, MCDBA
Protomix Rt.

ASP Suli

(I. rész)



Kedves Olvasóink!

Új sorozatot indítunk útjára: bemutatjuk, milyen lehetőségeket biztosít a Windows NT / 2000 webkiszolgálója, az IIS a dinamikus weboldalak, sőt, akár teljes webalkalmazások készítéséhez. A jelszó: ASP – azaz: Active Server Pages. A cikkben található példaprogramok megtalálhatók a **[1]** címen.

Egy kis alapozás

Amikor annak idején a HTML-t kitalálták, még senki sem gondolt arra, mi lesz a dolog vége. A HTML hipertext-leíró nyelv eredetileg arra való, hogy segítségével egyszerű dokumentumokat hozzunk létre, amelyek egyes részei esetleg hivatkoznak más dokumentumok részeire *(ez a hiperhivatkozás, hiperlink)*. Az eredeti HTML nyelv a hivatkozásokon kívül alig néhány elemet tartalmazott, amelyek különböző szintű címsorok, idézetek, esetleg listák létrehozását segítették. A sors fintora, hogy az Internet megjelenésével éppen a HTML lett az internetes kommunikáció egyik alapja. *(Nincs ezen mit csodálkozni: különféle adatok és közöttük felépített kapcsolatok leírására volt szükség, és a HTML éppen kapóra jött.)* A ma használatos HTML persze már jócskán több, mint egyszerű dokumentumleíró nyelv – pontosan annyi köze van az „ős" HTML-hez, mint a mai dokumentumoknak az akkoriakhoz. Régen az adatok struktúrája képezte az alapot, ma inkább azok megjelenítése.

Ahogy telt az idő, úgy szivárogtak bele a nyelvbe a tartalmat nem, azok megjelenítését annál inkább érintő elemek: képek, táblázatok, keretek *(framek)*, színek, méretek, betűtípusok, külső objektumok, scriptrészletek és ki tudja még mi minden. A HTML 4-es változatát többek között pontosan azért alkották meg, hogy valamelyest *(újra)* szétválaszthassuk a tartalmat a megjelenítéstől, ezzel is csökkentve a HTML oldalak kódjában található nem kis káoszt. A tartalom és megjelenítés szétválasztása azóta szinte minden területen hódít, függetlenül attól, hogy a hálózaton található HTML oldalak nagy része a mai napig nem használja ki a HTML 4 lehetőségeit *(köszönhető ez egyébként a szabványokkal többé-kevésbé hadilábon álló böngésző programoknak is)*.

Mozgásba lendül a kód...

Az idő múlásával egy másik területen is sokat fejlődött a HTML, illetve annak felhasználása. Kezdetben elég volt, ha egy dokumentumot létrehoztunk, annak tartalma nem, vagy csak ritkán változott. Később felmerült az igény arra, hogy a gyakran változó HTML oldalak tartalmát dinamikusan hozzák létre. Kezdettől két irányvonal létezett, attól függően, hogy a kiszolgálóra, vagy pedig a dokumentumot felhasználó ügyfélprogramra bízták a feladatot. Ez utóbbi megoldás nem biztos, hogy működik *(senki sem garantálja, hogy az ügyfélprogram képes ezt a feladatot végrehajtani)*, ráadásul több szempontból előnytelen is: ha a cél az, hogy 100 dologból csak egy valami jelenjen meg az ügyfél képernyőjén, felesleges mind a százat elküldeni neki, majd ott kiválasztani a szükséges egyet – egyszerűbb, biztonságosabb és olcsóbb, ha már eleve csak a számára érdekes adatok kerülnek hozzá. Ehhez viszont a kiszolgálónak kell erőfeszítéseket tennie. A kiszolgálóoldali megoldások közös tulajdonsága, hogy a kiszolgáló mindig kész, feldolgozható HTML kódot küld az ügyfélnek – a kód tartalma viszont dinamikus, időről időre változik. Magyarán: a cél az, hogy HTML kódot generáljunk.

A legkézenfekvőbb megoldás az volt, ha kész, teljes programokat írtak az egyes feladatok végrehajtására. A programok szabványos bemeneten *(stdin)* keresztül kapták a bemenő adatokat, majd a szabványos kimeneten *(stdout)* továbbították az általuk létrehozott kódot. A webkiszolgáló és a programok közötti kapcsolatot az ún. CGI *(Common Gateway Interface)* valósította meg, így a programok látszólag a kiszolgáló részeként működtek *(és működnek ma is)*. Ezt a módszert CGI programozásnak nevezzük, és bár néhány területen még ma is használatos, hátrányai miatt lassan kiszorul.

Mert: mit tehetünk, ha azt szeretnénk, hogy a CGI program mostantól más kódot generáljon? Egy: újraírjuk, újrafordítjuk és kicseréljük a programot. Brrrr. Kettő: Eleve olyan CGI alkalmazást írunk, ami a bemenő adatok segítségével paraméterezhető. Sőt, mi lenne, ha a bemenő paraméter egy valamilyen formában meghatározott parancssorozat, vagy akár egy valamilyen nyelven megírt script kód lenne? A CGI program azt értelmezi, végrehajtja, és visszaadja az eredményt – ez a megoldás a scriptnek köszönhetően teljesen dinamikus és bármire használható lenne – és az is. Jó példa erre a Perl nyelv, ahol a webprogramozás lelke a perl.exe. Bemenete egy Perl nyelven írt script, kimenete pedig a kész HTML kód.

ASP a láthatáron

A fenti megoldás egy kicsit mégis kényelmetlen: milyen jó lenne, ha a HTML oldalak általában statikus részét hagyományos módon, akár egy kényelmes WYSIWYG szerkesztővel készíthetnénk, és csak a dinamikus részt kellene programozni! Az Active Server Pages *(ASP)* elve pontosan ez: amikor azt mondjuk, ASP, tulajdonképpen egy HTML kódba ágyazott, speciális programozási módszerről beszélünk. *(Fontos, hogy az ASP nem egy programozási nyelv, hanem csak egy keretrendszer)*. Az ASP oldal végrehajtásakor a webkiszolgáló végigszalad az oldal tartalmán, és ha abban ASP scriptrészletet talál, végrehajtja. A HTML oldal és a script által esetleg visszaadott kódrészletek együttesen képezik az eredményt, amit azután az IIS elküld a böngészőnek. Lássunk egy példát:

```
<HTML><HEAD><TITLE></TITLE></HEAD>
<BODY>
<%
  Response.Write("<center>Hello World!</center>")
%>
</BODY>
</HTML>
```

A HTML kód belsejében található <% és %> jelzi az ASP kód kezdetét és végét. A köztük található kódrészlet elvileg soha nem jut el az ügyfélhez, csakis a kód futtatása során keletkező kimenet *(ami esetünkben a Response.Write() metódusnak átadott szövegrész)*. Az ASP scriptek beágyazásának módjáról kicsit később még lesz szó, most lássuk, mi lesz az eredmény:

```
<HTML><HEAD><TITLE></TITLE></HEAD>
<BODY>
<center>Hello World!</center>
</BODY>
</HTML>
```

Az ASP kód által generált kimenet tehát összemosódott a HTML kóddal. Ez jó, hiszen ráérünk az oldalt teljes egészében elkészíteni egy külső, kényelmes HTML szerkesztővel, majd utólag beágyazhatjuk az ASP kódot.
Egy gondolat az ASP használatáról: mint látható, az ASP az IIS webkiszolgáló része. A Windows NT 4.0 Option Pack segítségével telepíthető Internet Information Server 4.0 *(Windows NT 4.0 Workstation-ön Personal Web Server)* már tartalmazza az ASP kezeléséhez szükséges komponenseket, amelyek a Windows 2000 IIS5 webkiszolgálójában természetesen alapértelmezett tartozékok. Ha azt szeretnénk, hogy egy fájlt az IIS tényleg ASP oldalként kezeljen, adjunk a fájlnak .asp kiterjesztést *(ha ezt nem tesszük, a kód végrehajtás nélkül eljut az ügyfélhez, mintha a HTML oldal tartalma lenne)*.

Az ASP kódok beágyazása

Az ASP scripte*(ke)*t az oldalba több módon is beágyazhatjuk. Lássuk mindenekelőtt a HTML szabványnak megfelelő módot:

```
<HTML><HEAD><TITLE></TITLE></HEAD>
<BODY>
<SCRIPT runat="server" language="vbscript">
  Response.Write("<center>Hello World!</center>")
</SCRIPT>
</BODY>
</HTML>
```

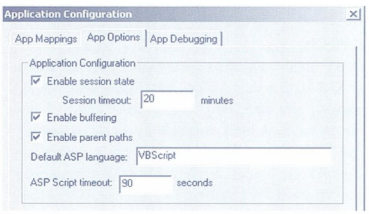
A SCRIPT HTML elem segítségével tehát ugyanúgy ágyazhatunk be kiszolgálóoldalon futó kódot, mintha ügyféloldali scriptet írnánk – csak adjuk meg a runat="server" attribútumot. Hasonlóan az ügyféloldali megoldáshoz, természetesen kiszolgálóoldalon sem muszály az oldalon belül megírni a scriptet, megadhatunk fájlnevet is *(src attribútum)*:

```
<SCRIPT runat="server" src="scfile.vbs">
```

Látható, hogy itt elhagytuk a scriptnyelv meghatározását. Ebben az esetben a kiszolgáló az alapértelmezett scriptnyelvet használja. Ezt két helyen határozhatjuk meg: egyrészt, az adott .asp oldal tetejére írt, úgynevezett ASP direktíva segítségével :

```
<%@ Language=VBScript %>
```

Ha pedig ez hiányzik, a kiszolgáló a saját beállításait használja, amit az adott IIS web vagy virtuális könyvtár tulajdonságai között, a „Home Directory" oldalon, az „Application Settings" részben található „Configuration" gombra kattintva megjelenő „Application Configuration" dialógusablak „App Options" oldalán, a „Default ASP language:" mezőben állíthatunk be. Egyszerű, ugye?



Az ASP alkalmazás beállításai az IIS-ben

Ha sikerült megtalálnunk ezt a dialógusablakot, jegyezzük meg, hogy jutottunk ide, mert sok más, fontos beállítás is itt található.
Az alapértelmezett kiszolgálóoldali scriptnyelv egyébként a VBScript. Ezt a nyelvet „használja" a korábban már bemutatott, rövidített formátum is:

```
<%
  Response.Write("<center>Hello World!</center>")
%>
```

A <% és %> használata rövidebb és kényelmesebb is, ezért cikkünk további részében – hacsak kifejezetten nincs szükség másra – ezt használjuk.
Természetesen egy oldalon belül több scriptblokk is szerepelhet. Az ASP oldal tartalmát az IIS elölről hátrafelé haladva értékeli ki, beleértve magát a HTML kódot is. Az ASP kód által visszaadott kód az eredményben ott jelenik meg, ahol maga a script szerepel, például a ...

```
<p>1<p><% Response.Write("a") %><p>2
<% Response.Write("b") %><p>3
```

... eredménye „1a2b3" és nem „ab123" vagy „123ab". A fenti példában láthatjuk azt is, hogy akár soron belül is készíthetünk scriptblokkot (inline script), nem ritka az alábbihoz hasonló megoldás:

```
<INPUT type="textbox" value="<% =sTxt %>">
```

Ezután a szövegmezőben az sTxt változó tartalma jelenik meg. Újabb újdonsággal találkoztunk: a <% után írt = a Response.Write rövidítése, tehát a <%="Hello!"%> egyenértékű a <%Response.Write("Hello!")%> sorral.

Még egy fontos tudnivaló a több részletben beágyazott scriptekről: nem tilos az sem, hogy a script „közepén" egyszer csak tiszta HTML kód jelenjen meg. Ilyenkor az úgy viselkedik, mintha a kód része lenne, azaz ha az adott szakaszra rákerül a vezérlés, az is megjelenik, különben rejtve marad.

```
<% For i=1 To 10 %>
<center>Hello World!</center>
<% Next %>
```

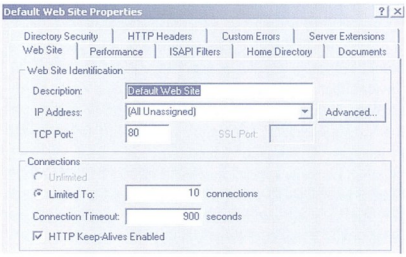
A fentiek hatására például a Hello World! felirat tízszer íródik ki, az alábbi kódrészlet pedig a csillagok pillanatnyi állásától függően hol ezt, hol azt „mondja" *(de sosem egyszerre a kettőt!)*:

```
<% Randomize ' <- Fontos, különben nem lenne
             ' véletlen! %>
<% If Int(Rnd()*10) > 5 Then %>
  <center>Kököjszi</center>
<% Else %>
  <center>Bobojsza</center>
<% End If %>
```

Így nagyszerűen szegmentálhatjuk az oldalt. Ha például egy ASP oldalon több minden jelenhet meg, de nem egyidőben, akkor legjobb, ha HTML szerkesztővel létrehozzuk az oldalt, benne az összes opcionális résszel, majd ezeket a részeket utólag „körbeépítjük" scripttel, ami majd eldönti, hogy az adott szakasz látsszon-e vagy sem.
Bár a <% és a %> nem szerepelnek a HTML szabványban, mi bátran használjuk, hiszen ezek a jelek soha nem hagyják el a kiszolgálót. Ha az ASP script által generált kimenő kód HTML-kompatibilis, nyugodtak lehetünk abban, hogy mi minden tőlünk telhetőt megtettünk a szabványos kommunikáció érdekében.

Ügyfél-kiszolgáló kommunikáció: a HTTP protokoll

A HTML oldalak számítógépek közötti továbbításához ki kellett dolgozni egy adatátviteli szabványt. Ez lett a HTTP, azaz Hypertext Transfer Protocol. A HTTP nem más, mint egy jól definiált ügyfél-kiszolgáló kommunikáció, amit most a jobb megértés kedvéért kicsit ízekre szabdalunk.
A HTTP kommunikációt az ügyfél kezdeményezi: hálózati kapcsolatot létesít a kiszolgálóval és közli vele igényeit: ez a HTTP kérés *(HTTP Request)*. A kérésre a kiszolgáló választ küld *(HTTP Response)*, majd az eredeti definíció szerint megszakítja a kapcsolatot és szükség esetén minden kezdődik elölről.
A kapcsolat megszakítására eredetileg azért volt szükség, hogy a fenntartott, használaton kívüli kapcsolatok ne terheljék feleslegesen a kiszolgálót. Manapság azonban más a helyzet: egy HTML oldalon képek, objektumok tömege lehet, így elvileg külön kapcsolatot kell felépíteni először a HTML kód, majd később minden egyes beágyazott kép és objektum letöltéséhez, és ma bizony éppen az újabb hálózati kapcsolatok létrehozása az, ami túlterhelheti a kiszolgálót *(ráadásul ez nem is igazán hatékony)*. Ezért a HTTP 1.1 verziójában bevezették a Keep-Alive opciót, amiben ha az ügyfél és a kiszolgáló megegyezik, a kapcsolat nem bomlik le azonnal *(magyarul egy kapcsolat során több kérés és válasz is elhangozhat)*. Ha bárki hibát észlel, természetesen azonnal bontják a kapcsolatot. Ma már szinte minden böngésző így próbál kapcsolódni a kiszolgálóhoz, azok pedig általában elfogadják ezt a kérést. Alapértelmezésben így tesz az IIS is, erről a „HTTP Keep-Alives Enabled" opció kikapcsolásával beszélhetjük le *(ezt az adott IIS web tulajdonságlapján, a „Web Site" oldalon találjuk)*.



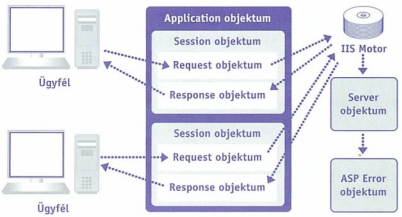
A HTTP Keep-Alive kapcsolója az ábra alján látható

A HTTP kérés- és válaszüzenet egyaránt három fő részből áll:
- Parancs, illetve státuszinformáció
- Fejlécek, metaadatok
- HTTP tartalom

Az első részben *(ami mindig az első sor)* kéréskor a kért parancs, illetve annak paraméterei, valamint a verziószám utazik, válasz esetén pedig a státusz- vagy hibaüzenet kódja és leírása. Az ezután következő fejlécek a kapcsolat és a később következő HTTP tartalom jellemzőt tartalmazzák, ezek vizsgálatára később részletesebben kitérünk, hiszen ami ügyfél-kiszolgáló kommunikáció és nem a HTML kód része, az valahol itt „utazik". Végül a HTTP tartalom, ami válasz esetén maga a HTML oldal, vagy mondjuk egy kép tartalma, kérés esetén pedig – ha van – a kiszolgálónak szánt bemenő adatok tömege. A fejléceket a HTTP tartalomtól egy üres sor választja el. Az ASP a HTML tartalom dinamikus létrehozása mellett természetesen lehetővé teszi a fejrészben kapott adatok feldolgozását és a válasz fejrészének manipulálását is.

Az ASP objektummodell

Az .asp oldalak programozását, a HTTP kommunikáció, a webkiszolgáló egyes szolgáltatásainak elérését külön objektummodell segíti. Az ASP objektummodelljének minden elemét elérhetjük az .asp oldalak kódjaiból. A későbbiek során mindegyik ASP objektumot bemutatjuk részletesen is, most vessünk egy röpke pillantást a teljes objektummodellre és annak elemeire.



Az ASP objektummodellje

Ha a tranzakciós műveletekhez használt ObjectContext objektumot nem számítjuk, az objektummodell hat *(Windows NT 4.0-n öt)* objektumból áll. A Windows 2000-ben megjelent új objektum az ASPError, ami egy bekövetkezett hiba leírását tartalmazza, az .asp-be ágyazott, saját hibakezelést segíti. A Server objektum magát az IIS-t képviseli, néhány kiszolgálószintű beállítással és szolgáltatással. Az Application objektum egy webalkalmazást jelképez. A webalkalmazás különálló egység, általában egy könyvtárban és annak alkönyvtáraiban található .asp kódok összessége, közös objektumokkal és beállításokkal. A Session objektum egy ügyfél és a kiszolgáló között „fennálló" kapcsolatot, munkamenetet jelképez. A „fennálló" kapcsolatot azért írtuk így, mert valójában nem egy kapcsolatról van szó. Az IIS *(a háttérben cookie-k segítségével)* azonosítja a felhasználót és a böngészőt, így az a böngésző bezárásáig saját munkamenetébe térhet vissza. A Request objektum egy HTTP kérést jelképez, segítségével kódból hozzáférhetünk a kérés minden elemához, legyen az HTTP fejléc értéke, a böngészőben tárolt cookie, vagy kérdőív tartama. A Response objektum pedig értelemszerűen a kérdésre küldendő választ jelképezi. Természetesen a Response objektum segítségével sem csak a válasz tartalmát, hanem a HTTP protokoll fejrészét is kezelhetjük. Egy IIS kiszolgálón belül Server és ASPError objektumból egy-egy létezik *(utóbbi csak akkor érhető el, ha hiba történt)*. Application objektum minden webalkalmazás egyedi, globális objektuma, Session objektum minden munkamenethez *(ügyfélhez)* egy jön létre, egyidejűleg tehát több is létezhet. Request és Response objektum pedig mindig az adott kérésre és válaszra vonatkozik, újabb kérés esetén új példány jön létre belőlük is.

Egy HTTP válasz – a Response objektum

Kezdjük a végén: a Response objektummal, ami egy HTTP kérésre adott választ hivatott jelképezni. A Response objektum legegyszerűbb *(és leggyakoribb)* alkalmazását már láthattuk korábban: a **Response.Write()** metódus segítségével állítottuk elő az oldal tartalmát. A Write() használata egyszerű: minden, amit paraméterként átadunk neki, bekerül a válaszként visszaküldött adatcsomagba. A Write() metódusról egyetlen dolgot kell tudni: a kiírt szöveg nem tartalmazhatja a %> jelsorozatot, helyette ezt kell írni: %\> Az ezt tartalmazó szöveget IIS automatikusan visszaalakítja majd az eredeti formára.

A válaszpuffer

Az .asp oldal előállítása természetesen több lépésben történik, az oldalban található scripttől függően előfordulhat az is, hogy az oldal tartalmának egy része csak bizonyos várakozási idő után áll rendelkezésre. Ilyenkor dönthetünk, hogy a már kész tartalmat elküldjük-e az ügyfélnek, majd várunk a folytatásra, vagy kivárjuk, amíg a teljes oldal elkészül, és csak a munka legvégén küldjük a komplett választ. Ez utóbbi esetben a „kimenet" egy pufferbe kerül. A pufferelés az IIS5-ben alapértelmezésben működik, míg az IIS4-ben alapértelmezésben ki van kapcsolva. A Response objektum segítségével mi magunk is kezelhetjük a puffert: mindenekelőtt, a **Response.Buffer** property-nek False értéket adva letilthatjuk, True segítségével pedig engedélyezhetjük a puffer használatát. A pufferelés hatását a bon.asp és boff.asp példaprogramok segítségével mindenki kipróbálhatja. A Clear() metódus kiüríti a puffert *(Legyünk óvatosak! A „külső" HTML kód is a pufferbe kerül, törléskor az is elveszik!)*, a Flush() pedig elküldi azt, ami addig a pufferbe került, majd csak azután törli a tartalmát. E két metódust a bflush.asp és bclear.asp példaprogramokban mutatjuk be.

Az oldal feldolgozását bármikor megszakíthatjuk a **Response.End()** meghívásával. Ha az oldal végrehajtása befejeződik, természetesen a puffer teljes tartalma az ügyfélhez kerül. Ha nyom nélkül szeretnénk befejezni a ténykedésünket, az End() meghívása előtt használjuk a **Response.Clear()** metódust.

HTTP fejlécek küldése

A HTTP válasz a tartalom mellett számos HTTP fejlécet is tartalmaz. Mi magunk is küldhetünk ilyen fejlécet a **Response.AddHeader()** metódus segítségével:

```
<%
  Response.AddHeader("MyHeader", "MyData")
%>
```

A metódus két argumentuma a fejléc neve és értéke – természetesen a fentinél értelmesebb célra is felhasználhatjuk. Számos dolog van, ami közvetlenül programozható a Response objektumon keresztül és végső soron egy-egy HTTP fejléc elküldéséhez vezet *(a Response.ContentType property beállítása pl. egy „Content-Type" HTTP fejlécet küld, stb.)*, de előfordulhat, hogy olyasmit kell használnunk, ami nincs így „kivezetve". A pufferelés befolyásolja a HTTP fejlécek használatát, kikapcsolt puffer esetén HTTP fejlécet természetesen csak az oldal tartalma előtt küldhetünk, tehát a Response.AddHeader() metódus ekkor csak az .asp oldal elején *(az ASP direktívák után)* állhat. Fontos tudni, hogy egy fejléc már nem vonható vissza: amit egyszer létrehoztunk, az a válaszban már benne lesz, még akkor is, ha töröljük a válaszpuffert.

Tartalom és karakterkészlet

A HTTP válasz sokféle tartalmat hordozhat magában. Egyáltalán nem egyértelmű például, hogy egy HTML dokumentum milyen karakterkészlettel íródott. Sőt, még az sem biztos, hogy a válasz egy HTML oldal.

Response.Charset = az oldalban használt karakterkészlet, kódtábla. Ha normális magyar betűket is használni szeretnénk, állítsuk „ISO-8859-2"-re. Természetesen ugyanezt HTML-ből, a <META> elem segítségével is megtehetjük, az alábbi két példa tehát egyenrangú *(habár a fejlécben beállított karakterkészlet általában felülbírálja a HTML-ben meghatározottat – ez a böngészőn múlik)*:

```
<%
  Response.Charset = "ISO-8859-2"
%>
```

```
<HTML><HEAD>
<meta http-equiv="Content-Type"
 content="text/html; charset=ISO-8859-2">
</HEAD>
...
```

Response.ContentType = a válasz MIME típusa *(a tartalom*

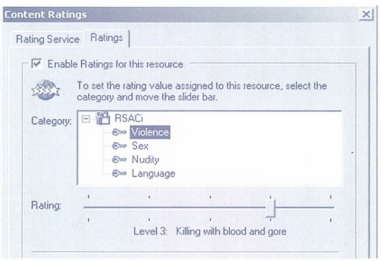
típusa). HTML *(és .asp)* oldal esetén az értéke „text/html", ha nem állítjuk be, ez az alapértelmezés is. Ennek a jellemzőnek az értékét akkor érdemes módosítani, ha a visszaküldött tartalom nem HTML, hanem mondjuk egy képfájl, mint alább *(sendpic.asp)*:

```
<%
  Set oStream =
  Server.CreateObject("ADODB.Stream")
  oStream.Type = 1 ' adTypeBinary
  oStream.Open
  oStream.LoadFromFile(Server.MapPath("ms.jpg"))

  Response.ContentType = "image/jpeg"
  Response.BinaryWrite( oStream.Read )
%>
```

A fent használt **Response.BinaryWrite()** metódus hasonló a .Write()-hoz, a különbség csak annyi, hogy ez binárisan, minden megkötés és konverzió nélkül írja a kimenetre az adatokat. A Server objektum MapPath() metódusáról később lesz szó, a lényege az, hogy egy relatív fájlnévből előállítja a fájl fizikai elérési útját a webkiszolgálón.

Érdemes megfigyelni az előző példaprogramot. Az ott használt ADODB.Stream objektum segítségével binárisan is írhatjuk/olvashatjuk a fájlokat, vagy más adatokat (természetesen adatbázis-mezőket is), míg a FileSystemObject objektum egyelőre csak szövegfájlok kezelésére képes. A Stream objektum az ADO 2.5-ös változatától létezik, ezért előfordulhat, hogy használata előtt telepítenünk kell a Microsoft Data Access Components (MDAC) legújabb verzióját, ami letölthető a [2] címről.

Egy nálunk még – sajnos – elhanyagolt jellemző maradt utoljára: a **Response.Pics** jellemző értéke jelezheti, hogy egy adott oldal milyen mértékben és mennyiségben tartalmaz erőszakra, szexre, meztelenségre és csúnya nyelvre utaló „jeleket". Ha ez az úgynevezett PICS-Label *(ami egyébként nem más, mint egy speciális HTTP fejléc)* utazik az oldallal, a szűrők képesek lennének kiválogatni a gyerekszobába nem való tartalmat – ismétlem, ehhez az kellene, hogy minden webkiszolgáló összes tartalmát minősítse valaki.
Ez az IIS beállításoknál egyébként webhelyenként, könyvtáranként, vagy akár fájlonként is beállítható:



Egy weboldal PICS-minősítése

Visszatérve az ASP-hez, a fenti beállítást kódból a következő módon érhetjük el:

```
<%
  Response.Pics( "PICS-1.0 ""http://www.rsac.org/
  ratingsv01.html"" l by ""[mICK]"" on ""2001.
  01.08T21:26+0100"" exp ""2002.01.08T12:00
  +0100"" r (v 3 s 0 n 0 l 0))" )
%>
```

Így belegondolva, talán mégis egyszerűbb, ha a grafikus felületen beállítjuk :-)

HTTP státusz és átirányítás

Response.Status = A HTTP válasz státuszüzenetét *(ami, ha minden rendben ment, 200 OK)* módosíthatjuk itt. A státuszüzeneteket a HTTP szabvány definiálja, és mivel a legtöbb funkció más módon is elérhető a Response objektumon keresztül, ezt a megoldást viszonylag ritkán használjuk. Hogy mégse maradjunk példa nélkül, lássunk egy átirányítást *(red0.asp)*:

```
<%
  Response.Status = "302 Object Moved"
  Response.AddHeader "Location",
  "http://www.microsoft.com"
%>
```

A fenti kódrészlet egyenrangú az alábbival *(red1.asp)*:

```
<%
  Response.Redirect("http://www.microsoft.com")
%>
```

A **Response.Redirect()** metódus tehát az ügyfél kérésének azonnali átirányítására való. A 302-es kódú HTTP üzenetnek *(tehát az átirányításnak)* egyetlen hátránya van: néhány proxy bizonyos körülmények között nem hajtja végre az automatikus átirányítást, hanem ronda „Object moved" hibaüzenetet küld vissza a böngészőbe. Ez akkor következhet be, ha az átirányítás mellett az adott oldal HTML tartalommal is bír *(magyarul, ha a válaszban nem csak az átirányító fejlécek szerepelnek, hanem más is)*. Három megoldás kínálkozik:

- Redirect() előtt ürítsük ki a puffert *(Clear() metódus)*, vagy eleve ne írjunk bele semmit *(a Redirect() az oldal elején szerepeljen)*
- Ha kiszolgálón belül kell „átirányítani", akkor használjuk inkább a Server objektum Transfer metódusát *(IIS5-től)*, ami kiszolgálón belüli átirányítást végez anélkül, hogy az ügyfél erről tudomást szerezne
- Használjuk a HTML-ben gyakori átirányítási módszert, ilyenkor maga a böngésző kezd automatikus letöltésbe, ami ráadásul időzíthető:

```
<META http-equiv="refresh"
  content="0;URL=http://www.microsoft.com/">
```

A fenti példában az időzítés 0, azaz a böngésző azonnal belekezd az új cím letöltésébe. A <META> elemet a HTML oldal fejrészébe helyezzük el *(ld. red2.asp)*.

Van itt valaki?

Ha egy oldal előállítása sokáig tart, vagy a webkiszolgáló túlterhelt, előfordulhat, hogy a kérés kiszolgálása közben *(vagy előtt)* az ügyfél megunja a várakozást és továbbáll. Az ilyenkor elvégzett munka kárba vész – még szerencse, hogy szükség esetén ellenőrizhetjük, nem hiába dolgozunk-e. Ha a **Response.IsClientConnected** property értéke hamis, nyugodtan befejezhetjük a feldolgozást, ha viszont igaz, érdemes még dolgozni. Természetesen felesleges minden sor előtt ellenőrizni, általában csak hosszú végrehajtási idejű oldalaknál van erre szükség, akkor is csak időközönként – nehogy többe kerüljön a leves, mint a hús.

Az IIS5 olyan komolyan veszi ezt, hogy minden kérés feldolgozása esetén ellenőrzi, hogy a kérés mennyi ideje érkezett. Ha a kiszolgáló túlterhelt, és a kérés több mint három másodperce várakozik, ellenőrzi, hogy megvan-e még az ügyfél, és csak akkor kezd bele a végrehajtásba, ha van kinek elküldeni a választ.

Az IIS4 kicsit felemás módon viselkedik az .IsClientConnected jellemző kiértékelésekor. Ha az oldalunk ilyen kiszolgálón fut, tudnunk kell, hogy az .IsClientConnected csak akkor használható biztonságosan, ha az oldalból valamit már elküldtünk az ügyfélnek (ha például a pufferelést bekapcsoltuk, csak a Flush() meghívása után számíthatunk helyes eredményre).

Gyorsítótárak minden szinten – a cache

Mi lenne velünk, ha nem lennének gyorsítótárak? A hálózati kapcsolatok lelassulnának, áruk az egekbe szökne *(igen, lenne még hova :-))*, az Internet-szolgáltatók bedugulnának. *(Felszabadulna néhány száz megabájt a merevlemezünkön :-))*. Ez senkinek sem lenne jó. Még szerencse, hogy ugyanarra a tartalomra sokan, sokszor kíváncsiak, és nem szükséges a dolgokat újra és újra létrehozni és letölteni az eredeti származási helyükről.

Gyorsítótárat alkalmaz a böngészőnk, saját gyorsítótárból dolgozik az Internet-szolgáltató, úgynevezett reverse-cache segíti a webkiszolgálókat a kérések gyors kiszolgálásában. Általánosságban elmondhatjuk, hogy – szerencsére – aki tud, tartalékolja a dolgokat, hátha később még szükség lehet rá.

Ezzel általában nincs is baj, de lehetnek dolgok, amelyeket felesleges elmenteni, mert a nevük, esetleg méretük hiába változatlan, tartalmuk gyakran eltérő. Kell-e jobb példa erre, mint a dinamikusan létrehozott weboldalak? Ugye, nem? Szerencsére a gyorsítótárak többsége távirányítható – a tartalom önmaga hordozhat olyan jeleket, amiket felismerve a gyorsítótár nem próbálkozik a tárolásával. Ilyen „jelek" *(természetesen HTTP fejlécek)* létrehozásában segít nekünk a Response objektum alábbi néhány szolgáltatása:

Response.CacheControl = Az oldal tárolásának szabályai. Ha értéke „private", akkor proxy kiszolgálók nem, csak és kizárólag privát, böngészőbeli gyorsítótárak tárolhatják az adatokat. Ez az alapértelmezés is. Ha a jellemző értéke „public", akkor az oldalt bármelyik proxy kiszolgáló tárolhatja. Ha azt szeretnénk, hogy egyáltalán senki ne tárolja az oldalunkat, a property-nek adjuk ezt az értéket: „no-cache".

Természetesen semmi sem tart örökké: még ha tárolunk is valamit, időnként érdemes frissíteni. Minden tárolt tartalomnak van egy „lejárati ideje", amit mi magunk állíthatunk be a **Response.Expires** és **Response.ExpiresAbsolute** jellemzők segítségével. Az előbbi az adott pillanattól számított lejárati időt jelenti *(percben)*, míg az utóbbinak konkrét időpontot adhatunk meg, pl.:

```
<% Response.ExpiresAbsolute=
   #May 31,2001 13:30:15# %>
```

Az oldal tárolását elkerülhetjük úgy is, ha a lejárati időt -1-re állítjuk:

```
<% Response.Expires = -1 %>
```

Régebbi, a HTTP 1.1 szabvánnyal nem kompatibilis kiszolgálók nem értelmezik a CacheControl értékét, ezért néha speciális HTTP fejlécre van szükség:

```
<% Response.AddHeader "Pragma", "no-cache" %>
```

A legbiztonságosabb természetesen az, ha mindhárom módszert *(CacheControl, Expires, Pragma)* kombináljuk. Ha nem akarunk ASP-t használni, ezt a szokásos módon, <META> elemek segítségével tisztán HTML-ből is megtehetjük:

```
<META HTTP-EQUIV="CacheControl"
   CONTENT="no-cache">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="-1">
```

HTTP kérés – a Request objektum
Adatküldés a HTTP protokoll segítségével

A dinamizmus, az interaktivitás egyik mozgatórugója természetesen az, ha menet közben adatokat kapunk az ügyfél oldaláról. Ezeket az adatokat a Request objektum segítségével érhetjük el.

Adatok átadása az URL-ben *(QueryString)*

A klasszikus adatbeviteli módszer, amikor az adatokat a kérés URL-jéhez csatoljuk, ilymódon:

```
http://localhost/qs.asp?input1=val1&input2=val2
```

Ekkor ilyen HTTP kérés indul a kiszolgáló felé:

```
GET /request.asp?input1=value1&input2=value2&
   submit=Submit HTTP/1.1
```

Ennek a módszernek több hátránya is van: egyrészt, a bemenő adatok növelik az URL hosszát, a kiszolgálónak elküldhető URL-ek mérete pedig biztonsági okokból általában korlátozva van. *(Próbáljunk meg az IIS-nek elküldeni egy többszáz bájt hosszú címet! – A válasz szabványos HTTP hibaüzenet: „404 – Request – URI too long").* Másrészt nemcsak kényelmetlen, de nem is igazán biztonságos, hogy az átadott adatok *(amelyeket esetleg nem is mi írtunk be, hanem mondjuk egy kérdőív rejtett részei voltak)* megjelennek a böngésző címsorában.

Az átadott adatmezők név=adat formájúak *(ld. fent: input1=val1)*, az egyes adatmezőket & jel választja el egymástól, az egészet pedig kérdőjel a fájlnévtől. Egy adott mező értékét a **Request.QueryString(*„mezőnév"*)** függvény adja vissza. Ha az adatok között ilyen mező nem szerepel, a visszaadott érték („"). A *qs.asp* fájl az alábbi kódrészletet tartalmazza:

```
<%
  If Len( Request.QueryString("nev") ) Then
    Response.Write( "Szia " &
    ↳   Request.QueryString("nev") & "!" )
  End If
%>
```

Ha nevünket megadjuk az URL-ben *(pl. qs.asp?nev=mick)*, akkor az oldal illendően köszönt minket. Egy mező „meglétét" a legegyszerűbben úgy ellenőrizhetjük, ha lekérdezzük a hosszát *(ezt teszi a Len() függvény a példában)*. Ha ez nem 0, lehet dolgozni.
Egy mezőnek azonban nem csak egy értéke lehet, a HTTP kérésben egy mezőnév egynél többször is szerepelhet:

```
http://localhost/qs2.asp?nev=Piroska&nev=Farkas
```

A qs2.asp fájlban látható, hogyan lehet ezt feldolgozni:

```
<%
  Response.Write("Nevek száma: " &
  ↳ Request.QueryString("nev").Count & "<br>")
  For i=1 To Request.QueryString("nev").Count
    Response.Write( i & ": " &
    ↳ Request.QueryString("nev")(i) & "<br>")
  Next
%>
```

A **Request.QueryString(„mezőnév").Count** érték visszaadja az adott nevű mezők számát. Ha a sok közül egy konkrét értékre vagyunk kíváncsiak, akkor átadhatjuk az indexet is *(a számozás 1-től kezdődik)*. Maradjunk a fenti példánál, a Farkas-ra így hivatkozhatunk:

```
Request.QueryString("nev")(2)
```

Ha egy mezőnek több értéke van, és mi mégis közvetlenül kérdezzük le *(pl. Request.QueryString(„nev"))*, akkor az értékek vesszővel elválasztott listáját kapjuk válaszként: „Piroska, Farkas". Ha pedig egyszerűen csak Request.QueryStringre hivatkozunk, akkor visszakapjuk a teljes kérdést: „nev=Piroska&nev=Farkas".
Ez volt tehát az URL-be ágyazott lekérdezés feldolgozása. Egy fontos és sokszor zavaró tényezőre még szeretném felhívni a figyelmet: az URL-ek formátuma kötött, és mivel a lekérdezés *(és főleg az átadott adatok)* ilyenkor az URL részét képezik, ezeknek az adatoknak is meg kell felelniük bizonyos szabályoknak: például, minden írásjel, neadj' Isten ékezetes karakter csakis kódolt formában *(pl. egyenlőségjel: %3D)* szerepelhet az URL-ben. Ez a kódolás pedig sokszor körülményes és kényelmetlen.

Adatfeltöltés a POST HTTP paranccsal

Szerencsére a HTTP protokoll tartalmaz egy, a fentinél fejlettebb megoldást is. A POST parancs használata esetén a feltöltendő adatok a HTTP üzenet törzsrészébe kerülnek. Az adatok kódolását persze így sem ússzuk meg, de az esetek többségében ezt a munkát nem mi, hanem a böngésző végzi. Kérdőív *(Form)* kitöltése esetén például, ha a FORM elem method attribútumát postra állítottuk, a következő kérés indul a kiszolgáló felé:

```
POST /request.asp HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 41
Connection: Keep-Alive
input1=value1&input2=value2&submit=Submit
```

Láthatjuk, hogy a kiszolgálónak szánt adatok nem a POST parancs paraméterében, hanem a HTTP üzenet törzsrészében utaznak. Érdemes megfigyelni a törzs kódolására vonatkozó Content-Type HTTP fejléc értékét is.

És itt álljunk meg egy szóra! Aki ismeri a HTML nyelvet, az tudhatja, hogy a HTML kérdőív elemének (FORM) method attribútuma határozza meg az adatok küldésének módját. Ha a method attribútum értéke „get", a kérdőív tartalmát az URL-be ágyazva, ha viszont az attribútum értéke „post", akkor a HTTP kérés törzsében küldi el a böngésző a kiszolgálónak. A form.asp példaprogrammal bárki kipróbálhatja a különbséget. Lehetőség szerint használjuk tehát a post módot!

A kérdőív egyes mezőinek értékét a **Request.Form** kollekció tartalmazza. Az előző példához hasonlóan minden mező értékét lekérdezhetjük, ha a mezőnek több értéke van, akkor itt is használhatjuk a .Count jellemzőt és az indexeket is, pl.:

```
Request.Form("nev")
Request.Form("nev").Count
Request.Form("nev")(5)
```

Érdekesség: Kérdőív feldolgozása esetén a Submit nyomógomb értéke *(felirata)* is eljut a kiszolgálóhoz, de ha több ilyen van, akkor csak annak az egynek, amelyikre kattintottunk. Így megtehetjük azt, hogy ha egy kérdőívbe két Submit nyomógombot teszünk:

```
<INPUT type="submit" name="submit" value="Egyél">
<INPUT type="submit" name="submit" value="Igyál">
```

... a feldolgozáskor könnyen kitalálhatjuk,
mit szeretne a kedves ügyfél:

```
<%
  If Request.Form("submit") = "Egyél" Then
    ' Eszem
  Else
    ' Iszom
  End If
%>
```

A For Each utasítás segítségével *(a többi kollekcióhoz hasonlóan)* a .Form kollekció elemeit is kilistázhatjuk *(ld. request.asp)*:

```
<%
For Each mezo In Request.Form
  ' a mezo-be a mezonev kerul
  Response.Write( "<b>" & mezo & " = </b>" &
  ↳ Request.Form(mezo) & "<br>" )
Next
%>
```

A HTTP tartalom kiolvasása

Nem kötelező a Request objektum kollekcióira támaszkodnunk, ha ki szeretnénk olvasni a HTTP kérés törzsében elküldött adatokat:

```
<%
  bytes = Request.TotalBytes
  data = Request.BinaryRead(bytes)
%>
```

A **Request.TotalBytes** jellemző visszaadja a törzsben található adatok méretét, a **Request.BinaryRead()** metódus pedig adott mennyiségű adatot olvas be, természetesen minden átalakítás és konverzió nélkül.
Fontos! A Request.BinaryRead() metódus használata után már nem használhatjuk a Request.Form kollekciót, és fordítva: ha a Request.Form-hoz már „hozzányúltunk", a **Request.BinaryRead()** már hibát okoz.

Cookie

Szüleink azt tanították, hogy idegenektől ne fogadjunk el édességet. Én most mégis azt mondom, ebben az egy esetben kivételt tehetünk. A cookie kis adatcsomag, amit a kiszolgáló kérésére a böngésző az ügyfél számítógépén tárol, és szükség esetén visszaküldi azt.
Alapjában véve két különböző típusú cookie létezik: az első típus csak addig „él", amíg a böngészővel egy kiszolgálónál tartózkodunk. A böngésző bezárásával az ilyen cookie tartalma elveszik. Ezeket a cookie-kat elsősorban átmeneti célra használják *(például az ASP Session fenntartására)*. A másik fajta, felhasználói szemmel gyakrabban megfigyelt cookie annyiban különbözik az előzőtől, hogy a böngésző bezárásakor nem veszik el, hanem a számítógép lemezére kerül. Az ilyen cookie-knak érvényességi idejük van. Amíg ez az érvényességi idő le nem jár, addig a böngésző megőrzi az értéküket, és tartalmukat minden egyes látogatáskor visszaküldi a kiszolgálónak. A két cookie-fajtát csak a lejárati idő különbözteti meg egymástól.
Lássuk tehát, hogyan küldhetünk egyszerű, átmeneti célra használható cookie-t a böngészőnek:

```
<%
  Response.Cookies("nev") = "ertek"
%>
```

Ha a felhasználó legközelebb felénk jár, ezt az adatot így olvashatjuk ki:

```
<%
  s = Request.Cookies("nev")
%>
```

Egy cookie nem csak egy szöveget tartalmazhat, hanem többet is, egyfajta táblázatot, almezőket:

```
<%
  Response.Cookies("nev")("mezo1") = "ertek1"
  Response.Cookies("nev")("mezo2") = "ertek2"
%>
```

A **.HasKeys** metódus segítségével eldönthetjük, hogy egy cookie egyszerű szöveg, vagy almezők gyűjteménye, és ettől függően kezelhetjük is:

```
<%
  If Request.Cookies("nev").HasKeys Then
    For Each mezo In Request.Cookies("nev")
     Response.Write( Request.Cookies("nev")(mezo))
    Next
  Else
    Response.Write( Request.Cookies("nev") )
  End If
%>
```

A **.HasKeys** elérhető a Response objektumon keresztül is. Erre azért van szükség, mert ha egy szöveges típusú cookieban almezőket hozunk létre, elveszik a szöveges érték – és fordítva, ha almezőkkel rendelkező cookie-nak szöveges értéket adnánk, elvesznének az almezők és azok értékei.
A cookie érvényességét a Response objektum segítségével korlátozhatjuk időben és „térben":

```
<%
  Response.Cookies("nev").Expires = "01-Jan-2003"
  Response.Cookies("nev").Domain =
  ↳ ".netacademia.net"
  Response.Cookies("nev").Path = "/dir1/dir2"
%>
```

Az **.Expires** jellemző értéke határozza meg, hogy az cookie meddig marad életben. Ha nem adjuk meg, a böngésző lezárásakor elveszik. A **.Domain** jellemző segítségével beállíthatjuk, hogy a böngésző milyen domainek elérése esetén küldje vissza a cookie-t. A **.Path** pedig kiszolgálón belüli részletezést jelent: ha két azonos nevű cookie létezik, ugyanarra a domain-re, akkor annak az értékét fogjuk visszakapni, ahol a .Path értéke közelebb van a valósághoz. Ha tehát az oldal a /dir1/dir2 könyvtárak mélyén található, a két cookie .Path értéke pedig a /dir1 és a /dir1/dir2, akkor az utóbbit fogjuk viszontlátni.
Ha azt szeretnénk, hogy egy cookie az adott pillanattól számított egy évig legyen érvényes, használjuk a DateAdd() és a Now függvényt:

```
<%
  Response.Cookies("egyevigjo").Expires =
  ↳ DateAdd( "yyyy", 1, Now)
%>
```

Fülöp Miklós
mick@netacademia.net

A cikkben található URL-ek:
[1] http://technet.netacademia.net/feladatok/asp/1
[2] http://www.microsoft.com/data
ASP objektummodell-referencia a weben:
http://msdn.microsoft.com/library/psdk/iisref/vbob74bw.htm

Microsoft Visio 2000 a nagyvállalatoknál

Bevezetés

A vállalatoknak biztosítaniuk kell alkalmazottaiknak a változások és lehetőségek azonnali felismerését, és az azokra való gyors válaszadást, hogy alkalmazkodni tudjanak a változó viszonyokhoz.

A sikeres vállalatok ezt az információ megosztásával érik el, ezért szükségük van olyan eszközökre, melyek elősegítik az elgondolások, a folyamatok és a rendszerek megértését. Ez különösen akkor igaz, ha a rendszerek túl összetetté válnak ahhoz, hogy szavakkal egyszerűen leírhatók legyenek. A számítógépes hálózatok leírása például elképzelhetetlen lenne ábrák nélkül.

Ez a cikk egy új vállalati kommunikációs módszert mutat be, melyet a Microsoft vizuális munkamódszernek *(visual workstyle)* nevez. Bemutatjuk, hogyan válnak jobban érthetővé az összefüggések, és hogyan javítható a kommunikáció a Visio 2000 *(mely az egész világon szabványként elismert üzleti célú rajzoló és ábrakészítő program)*, a vizuális munkamódszer megvalósítási eszközének vállalati szintű bevezetésével. Látható lesz, hogy a Visio 2000-ben megtalálható a hatékony vizuális kommunikációs eszközök négy fő jellemzője. A Visio 2000:

- Alkalmazkodóképes. A világ nagyvállalatainak szembe kell nézni a globalizáció, az egybeolvadás *(például Daimler és Chrysler)*, a felvásárlás *(például Microsoft és Visio :))*, a távoli és mobil munkaerő és a túlzott mennyiségű információ okozta kommunikációs kihívásokkal. Ezeknek a vállalatoknak olyan eszközökre van szükségük, melyek biztosítják a piaci változásokra és munkahelyi problémákra való gyors reagálást. A vizuális munkamódszer a természetadta emberi tulajdonságokon alapul, és a megfelelő eszközök használatával nagyon hatékonnyá tehető.
- Rugalmas. A sikeres vállalatok megértették, hogy a versenyelőny biztosításának kulcsa az, hogy az elgondolásokat, folyamatokat és rendszereket az alkalmazottaik gyorsan megértsék, és ezzel biztosítva legyen számukra a nagyobb hatékonyság és a hibátlan munkavégzés. Ezek megvalósításához rugalmas eszközökre van szükség.
- Programozható. A versenyelőny és alacsony működési költség biztosításához olyan platformra van szükség, mely a lehető legtöbb "dobozból kivett" eszköz felhasználásával biztosítja gyorsan üzembe helyezhető és jól testreszabott alkalmazások készítését. A Visio 2000 jól átgondolt fejlesztői platform, mely az egyedi megoldásoknak jó alapot biztosít.
- Költséghatékony. A vállalatok egyre több pénzt költenek az informatikai infrastruktúra felépítésére és karbantartására. Az átgondoltan építkezők olyan eszközöket keresnek, melyek beépülnek a meglévő rendszerekbe, és képesek együttműködni azokkal, így csökkentve a teljes birtoklási költséget.

Az új vizuális munkamódszer

A mai üzleti viszonyok rákényszerítik a vállalatokat a piaci változásokhoz való azonnali alkalmazkodásra. Ehhez szükség van a megfelelő rendszerek birtoklására, melyek lehetővé teszik az akár világszintű hálózatra kiterjedő reakciókat. A szellemi munkásoknak képesnek kell lenni a változások gyors megértésére és kezelésére, valamint olyan eszközökkel kell rendelkezniük, melyek biztosítják számukra az új lehetőségek felismerését és kihasználását. A vizuális munkamódszer a folyamatos változások közben létrejövő információk megértésének és megosztásának az emberek képességeihez legjobban alkalmazkodó, intuitív módja.

Ebben a részben a vállalati kommunikáció és tanulás legnagyobb kihívásait ismertetjük, és bemutatjuk, hogyan alakíthatják a szellemi dolgozók vizuális munkamódszer segítségével ezeket a kihívásokat lehetőségekké.

Kommunikációs kihívások

A munkamódszerek változása. A vállalati hatékonyság előtérbe kerülése a vállalati vezetés struktúrájának ellaposodásához, és az önállóan dolgozni képes munkaerő iránti igény megnövekedéséhez vezetett. Emellett a munkahelyi struktúrák is megváltoztak. 30 százalékkal több munkát végeznek ma a munkacsoportok, mint tíz évvel ezelőtt, a megbeszéléseken való távoli részvétel pedig mintegy 25 százalékkal nőtt.

Túl sok információ. A szellemi munkásokat folyamatosan bombázzák információkkal *(általában szöveges dokumentumokkal)*, ezért ritkán van idejük a megszerzett információk megosztására, vagy egy saját tudásbázis kialakítására és bővítésére. A vezetők fele érzi úgy, hogy gyakran képtelen a megkapott információmennyiség kezelésére, és 43 százalékuk arra panaszkodik, hogy fontos döntések meghozatala csúszik, mert csökken a döntéshozó képességük a túl sok információ miatt.

Az Internet térhódítása. Óvatos becslések szerint is több mint 160 millió ember kapcsolódik az Internetre. Az Internet és az intranetek a munkatársak és az ügyfelek közti hatékony kommunikációs, és kapcsolatépítési lehetőségeket biztosítanak, de rengeteg információ keveredik rajtuk össze, melyek egy részét a dolgozóknak meg kell jegyezni, és hogy dolgozni tudjanak velük, osztályozniuk kell, és sorrendbe kell rakniuk őket.

Útkeresés a kommunikációban

Az ismeretek megértése és megosztása. A vizuális munkamódszer bevezetése biztosítja a szellemi munkásoknak a felmerülő problémák újfajta, kreatív megoldását. Az elfogadott tudományos álláspont szerint, ha az egyén agyának mindkét felét használja *(a balt a szöveges, a jobbot a képi információk feldolgozásához)*, akkor gyorsabb a megértés, és alaposabban elraktározódnak az információk. A folyamatosan változó tudományos álláspontoktól függetlenül kijelenthetjük, hogy az olyan eszközök használatával, melyek igénybe veszik mind a képi, mind a nyelvi felfogóképessé-

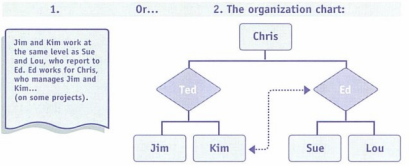
get, a dolgozók hatékonyabban képesek értelmezni és megosztani a rendelkezésükre álló információkat.

A képi gondolkodás gyakran elősegíti a problémamegoldást. Az adatok összegyűjtése és képi formába rendezése segít elképzelni összetett dolgokat, például létesítmények felügyeleti tervét, hálózati architektúrákat és termékfejlesztési életgörbéket.

Hatásketltés. A szellemi munkások kezdik belátni, hogy a képi üzenetek jobban átjutnak a munkatársakhoz és vásárlókhoz, és szinte áttörnek az információáramlás zűrzavarán. A felmérések bebizonyították, hogy a képi anyagot tartalmazó bemutatók 43%-kal meggyőzőbbek az azt nélkülözőknél. Az erős asszociatív elképzelések, melyek a képi kommunikáció eredményei, 250%-kal jobban megmaradnak az emberek emlékezetében, mint a szóbeli kommunikáció.

Változások kezelése. A globalizáció, a vállalatok összeolvadása és a folyamatos üzleti átszervezések során a szervezeti és ügyviteli változások sikeres kommunikálása csökkenti a fejetlenséget és az eredménytelenséget.

A képi gondolkodás gyakran elősegíti a problémamegoldást. Az adatok összegyűjtése és képi formába rendezése segít elképzelni összetett dolgokat, például létesítmények felügyeleti tervét, hálózati architektúrákat és termékfejlesztési életgörbéket. A szervezeti és üzleti modell diagramok biztosítják a szellemi munkásoknak a vállalati fejlődés hatásának és saját feladataik, felelősségük változásának megértését és bemutatását.



Ki a főnök? A képi kommunikáció intuitív, nyelvtől független, szövegnél hatékonyabb eszköz, mely elősegíti az alárendeltségi viszonyok gyors megértését.

A hatókör megnövelése. A globalizáció a kultúrák közti és a földrajzi távolságokat egyaránt áthidalni képes eszközöket igényel. A globális kommunikációnak meg kell valósulnia, és gyorsabban kell bonyolódnia, mint eddig bármikor. A képi eszközök nagyon leegyszerűsítik a különböző kultúrák közti kommunikáció megvalósulását, és segítenek az új és a távoli dolgozóknak a vállalat munkatempójához való problémamentes alkalmazkodásban.

A képi kommunikáció működés közben

A Microsoft Visio 2000 a vizuális munkamódszerek kivitelezésének és bevezetésének nélkülözhetetlen alapja. Nyílt architektúrája segítségével a Visio 2000 kielégíti a növekvő nagyvállalatok összetett üzembehelyezési és üzemeltetési igényeit. A következő három részből álló mintapélda bemutatja, hogy hogyan is növelik a Visio 2000 kommunikációs eszközei egy éppen megváltozó vállalaton belül a hatékonyságot és termelékenységet.

1. rész: Az egyesülés

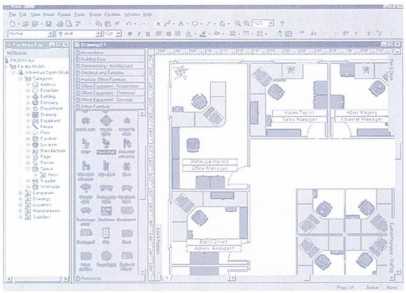
Az Adventure Works és a Fabrikam Inc. elhatározták, hogy összevonják erőiket, hogy piacvezetők legyenek a szabadidőruházat eladásban. Az egyesülés feltételei között szerepel az összes alkalmazott megtartása. Mivel jelentős átfedések vannak a munkakörökben és a kötelezettségekben, az alkalmazottak tétovázva haladnak előre a projektekkel, nehogy duplán végezzék el ugyanazt a munkát.

A két humánerőforrás csapat összeül, és egyesítik adatbázisaikat. Az így kapott információk felhasználásával automatikusan létrehozhatnak egy összevont szervezeti diagramot, mely tartalmazza az összes dolgozó elérhetőségét, és közzétehetik az egyesített vállalati intraneten. Az alkalmazottak így képesek lesznek azonosítani a velük megegyező munkakörben dolgozókat, és látják, kivel kell együtt dolgozniuk. A vezetés pedig elkezdhet azon gondolkozni, hogy hogyan tudják a legjobban hasznosítani az összevont szaktudást.

A két humán erőforrás csapat egyesíti adatbázisát, hogy elkészülhessen az összevont szervezeti diagram.

2. rész: A költözés

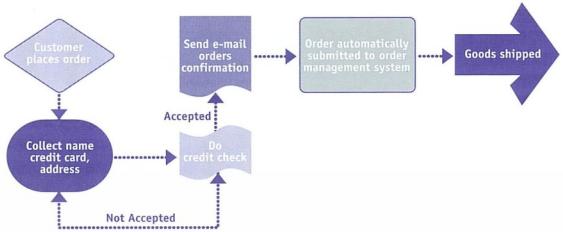
A vezetőség úgy határozott, hogy összevonják a két irodát is. Az Adventure Works telephelye a külvárosban van, és könnyen bérelhetnek az irodájukhoz kapcsolódó további területeket. A Fabrikam alkalmazottainak viszont el kell hagyniuk a belvárosi telephelyüket. Az épületek felügyeletének vezetője lehetővé teszi a Fabrikam dolgozóinak az új irodabútor kiválasztását, és új helyük elrendezésének meghatározását. A vezető készít egy testreszabott Microsoft Visio 2000 sablont, mely tartalmazza a megfelelő bútoralakzatokat és irodafelszereléseket. Az új iroda Visio 2000-rel készült alaprajzát, melyen ki van jelölve minden alkalmazott helye, közzéteszi a vállalat belső hálózatán. A rajzra mutató hivatkozásra kattintva az alkalmazottak a számítógépükre telepített Visio 2000 segítségével megnyithatják az ábrát. Amikor az alkalmazottak kiválasztották a nekik szükséges dolgokat, az épületek felügyeletének vezetője lefuttathat egy jelentést a kész rajzon, és létrehozhat egy megrendelőlistát, melyen szerepel az összes szükséges bútor. Ez a lista az ábrán elhelyezett bútorokhoz kapcsolódó adatok alapján készül.



A testreszabott bútoralakzat- és irodafelszerelés-sablonok használatával az alkalmazottak a kívánt elrendezésnek megfelelően a helyükre mozgathatják a bútorokat a képernyőn, majd a megváltoztatott ábrát a hálózatra menthetik.

3. rész: Az elektronikus kereskedelem kiépítése

Az irodákat sikeresen összevonták, és összeálltak a munkacsoportok is. A piac alapos elemzése során kiderült, hogy az új cég vásárlói közül sokan használják az Internetet. A vezetőség úgy dönt, hogy a meglevő értékesítési csatornákat kibővítik egy elektronikus kereskedelmi webhellyel. Az Internet munkacsoport egy diagram segítségével mutatja be a tervezett elektronikus rendelési folyamatot, hogy elkerülhetők legyenek a hibák, és a lehető legegyszerűbb legyen a folyamat.



Az eredeti rendeléskezelési diagram az elektronikus rendelési folyamat általános ábrázolása, mely segít az Internet csapatnak átláthatóvá tenni a folyamatot. Rájönnek, hogy az ábra nem tartalmazza azokat a vásárlókat, akiknél szükségtelen a fizetőképességet ellenőrizni.

Az eredeti diagram nem veszi figyelembe, hogy az Interneten vásárlók között sok olyan lehet, aki a cég törzsvásárlója. Náluk fölösleges a fizetőképesség ellenőrzése, és csak lassítaná a rendelési folyamatot. A folyamat ábrázolásának segítségével a munkacsoport észreveheti, hogy mely lépéseket kell módosítani, és kialakítható a rendelési folyamatot legjobban elősegítő webhelyszerkezet. Az átalakított folyamatábra használható a webhely építőinek és a beszállítók tájékoztatására is. További folyamatábrák is készülnek, melyek a kereskedelmi folyamatot mutatják be.

Az új folyamatábra megmutatja, hogyan segít az információk képi megjelenítése a problémák megoldásában, a vállalaton belüli kommunikáció minőségének javításában és a fontos folyamatok dokumentálásában.

Erőteljes eszköz, mely segít az információk jobb megértésében és a kommunikáció javításában.

Egy nemrégiben megjelent tanulmány szerint a Fortune 500 cégek döntéshozói hangsúlyozták, hogy az egyik legfontosabb dolog az, hogy az eredmények gyorsan eljussanak hozzájuk. Az elgondolások, folyamatok és rendszerek átlátásához az egyik legjobb módszer azok ábrázolása egy diagramon, vagy rajzon. Azoknak a vállalatoknak a szellemi munkásai, akik már alkalmazzák a vizuális munkamódszert, alátámaszthatják ezt az állítást. Az általuk használt eszközöknek támogatniuk kell a kommunikációnak ezt a módját. Ha az összes előnyét figyelembe vesszük, megállapíthatjuk, hogy a Microsoft Visio 2000 platform vitathatatlanul a jelenleg piacon levő legjobb eszköz a képi kommunikáció megvalósításához.

Egyszerű használat

A vállalatoknak olyan eszközökre van szükségük, melyek nem nehezítik meg az alkalmazottak munkavégzését. Ezeknek az eszközöknek egyszerűnek kell lenni, hogy a tapasztalatlan felhasználók is kezelni tudják őket, de a tapasztalt felhasználók számára biztosítaniuk kell a legbonyolultabb feladatok elvégzését is. A Visio 2000 kezelőfelületét úgy tervezték, hogy kielégítse ezeket az igényeket.

A sablonok és minták jó szolgálatot tesznek azoknak a felhasználóknak, akiknek nem kell vázlatok alapján objektumokat készíteni. Az ezeket használók egyszerűen, „húzd és ejtsd" *(drag and drop)* módszerrel állíthatnak össze az alakzatokból diagramokat. A nagyszámú diagramtípus és beépített intelligens objektum—melyeket SmartShapes szimbólumoknak neveznek—lehetővé teszi a felhasználható ábrák egyszerű elkészítését.

Rugalmasság

A vállalatok nemcsak azért szabványosítják az asztali gépek alkalmazáskészletét, hogy egyszerűsítsék a szoftverbeszerzést és az üzembehelyezést, hanem azért is, mert ez megkönnyíti a felhasználók képzését, a felhasználók közti fájlcserét és a használatot. A Visio 2000 intuitív felhasználói felülettel rendelkezik, iparáganként jellemző minták és alakzatok ezreit tartalmazza, így biztosítja mind az egyszerű, mind az összetett rajzok készítését. Még a szokatlanul nagy, akár több tízezer objektumból álló rajzok is egyszerűen létrehozhatók és kezelhetők a termék navigációs képességeinek *(például az automatikus számozásnak és egyéb alakzatkezelő eszközöknek)* köszönhetően.

A még nagyobb rugalmasság elérése érdekében a Visio 2000-be beépítették többféle dokumentumtípus kezelését. A saját natív .VSD fájlformátuma mellett a Visio 2000 képes kezelni *(importálni)* más, népszerű rajzolóprogramokkal készített rajzokat *(például CorelDRAW! 3–7 verziók, a Micrografx Designer és az ABC Flowcharter bizonyos verziói)*. Ha egy Visio 2000 diagramot más célra akarunk használni, akkor rengeteg egyéb formátumba *(például BMP, GIF, JPEG, HTML, és VML - Vector Markup Language)* menthetjük el.
A Visio 2000 Technical Edition képes írni és olvasni a piacvezető .DWG és .DGN CAD *(computer-aided design – számítógéppel támogatott tervezés)* fájlformátumokat, ráadásul képes átalakítani a statikus CAD objektumokat dinamikus SmartShapes szimbólumokká.

Intelligens funkcionalitás

A Microsoft Visio 2000 rugalmas megoldás, mely kielégíti a nagyvállalati vásárlók folyamatosan változó igényeit. A Visio 2000 a robusztus funkcionalitás megtartása mellett válik egyre intelligensebbé és gyorsabbá.
A Visio 2000 talán a leggyorsabban piacra dobott termék, elődjéhez képest jobb teljesítményt és skálázhatóságot mutat. A rajzolási és diagramkészítési feladatok kevesebb időbe telnek, és a meglévő kiegészítő alkalmazások is gyorsabban futnak. Emellett a javított snap-to *(megadott helyekre ugrás)* és elrendezési vezérlők nagyban meggyorsítják a rajzolási folyamatot. Az új, madártávlati nézet ablak, a gyors képcsúsztatási és nagyítási *(pan-and-zoom)* lehetőségek segítségével sokkal jobban kezelhetők a nagy rajzok.

Internetes és intranetes képességek

A vállalatok a kommunikáció gyorsítására és új üzleti lehetőségek kiaknázására használják az Internetet. A Visio 2000 ebben is kiváló partner, hiszen az Office család többi tagjához hasonlóan, kiváló webes képességekkel rendelkezik *(például a kész rajzok megosztása, és a rajzokon belül hivatkozások elhelyezése)*.
A Visio 2000 diagramjai HTML formátumban is elmenthetők *(ezután pedig természetesen közzétehetők egy webkiszolgálón, és egy egyszerű webböngésző segítségével megtekinthetők)*. A Visio 2000 támogatja a VML *(Vector Markup Language – vektorleíró nyelv)* formátum használatát, tehát az elkészült rajzok ilyen formában is elmenthetők. Miért is jó ez? Azért, mert így az Internet Explorer segítségével, a Visio 2000-et megközelítő rajznézegetési funkciókat kapunk *(például nagyítás)*.
A Visio 2000 SmartShapes szimbólumai támogatják a beágyazott hivatkozásokat. Bármely alakzatban elhelyezhető egy másik „rajzlapon", fájlban, vagy akár weblapon található információra való hivatkozás, ebből következik, hogy a Visio 2000-el készített rajzok dinamikussá tehetők.
Hogy a felhasználónak ne kelljen sokat keresgélnie a frissítéseket, kiegészítéseket, vagy a termékről szóló híreket és új információkat, a Visio mindegyik változata tartalmazza az adott termék webhelyére mutató hivatkozásokat. Itt letölthetünk különféle kiegészítéseket, technikai támogatást, sőt felhasználói képzést is kaphatunk, tehát minden segítséget megkapunk ahhoz, hogy a legtöbbet hozzuk ki a Visio-ból.

Jól kidolgozott platform az üzleti megoldásokhoz

A vállalatok dolgozóinak feladataik megoldására alkalmas eszközökkel kell rendelkezni, ha lépést akarnak tartani az egyre gyorsuló üzletmenettel. A Microsoft Visio 2000 rugalmas platformot biztosít az asztali gépek testreszabott rajzoló és diagramkészítő megoldásai számára.

Jól használható megoldások gyors elkészítése

A testreszabott vállalati megoldások elkészítésére fordítható idő és létszám jelentősen lecsökkent az utóbbi években. Amikor a vállalatok robusztus és hatékony megoldásokat keresnek problémáik megoldására, előnyben részesítik a „dobozból kivett" alkalmazásokat, mert ezek lehetővé teszik számukra az üzembehelyezés és alkalmazások testreszabásának meggyorsítását.
Fejlesztői platformként a Visio 2000 egy könnyen bővíthető alapot biztosít, melyre a vállalatok meglévő üzleti rendszereikkel integrált megoldásokat építhetnek *(például követelések követése, kereskedelmi folyamatok automatizálása, automatikus időbeosztások elkészítése, adatok képi megjelenítése)*. Ezek a megoldások nagy értéket képviselhetnek egy vállalat számára, ugyanakkor minimális időráfordítással elkészíthetők:

- A legfőbb képesség a testreszabott alakzatok, sablonok és minták elkészítésének lehetősége. Minden Visio 2000 alakzatot paraméterezett képletek írnak le. Ez biztosítja a fejlesztőknek az intelligens, adatfüggő alakzatok létrehozását, melyek intuitívan viselkednek a különféle problémák megoldásakor. Ez a programozás egy számolótáblákhoz hasonló, ShapeSheet nevű környezetben végezhető.
- A Visio 2000 tartalmazza a Microsoft Visual Basic for Applications *(VBA)* 6.0 verzióját. Ez ugyanaz a VBA, amely a Microsoft Office 2000 alkalmazásaiban is megtalálható, és olyan fejlesztői környezetet biztosít, mely az egyszerű makrók írásától a teljes alkalmazások írásáig mindent biztosít.
- Ha ez valakinek nem lenne elég, a Visio 2000 tartalmazza az alakzatok információinak adatbázisokhoz való kapcsolását is. Ez azt jelenti, hogy minden Visio 2000 alakzat tartalmazhat adatokat, tehát gyakorlatilag bármilyen információt, amit a fejlesztő el akar helyezni benne. A fejlesztő ODBC-n *(Open Database Connectivity)* keresztül kapcsolhatja az adatokat bármilyen ODBC kompatibilis adatbázishoz. A rugalmasság további fokozásának érdekében a fejlesztők arra is használhatják a VBA-t, hogy az adatokat bármilyen külső adatbázishoz kapcsolják, és teljes mértékben szabályozni tudják a Visio 2000 és az adatbázis közti adatmozgásokat.

Magasszintű fejlesztési lehetőségek

A vállalatok üzleti alkalmazás-portfoliójuk összeállításakor egyre inkább a „polcról levett" összetevőket keresik. A Visio 2000-ben jól kidolgozott, negyedik generációs programozói API található, mely kifinomult objektummodellt tartalmaz. Ez biztosítja, hogy a Visio 2000 felhasználásával készült egyedi alkalmazások fejlesztésekor ne csak a VBA, hanem a Visual Basic, a C++, a Delphi, a Java, vagy bármely más programnyelv is használható legyen.
A főbb újítások a következők:

- Bővített eseménymodell: meghatározott események végrehajtásának szabályozása programozással.
- Átdolgozott felhasználói felület rendszer: a felület ele-

meinek *(menük, eszköztárak)* alaposabb szabályozása.
- Visio 2000-ben megnyíló felhasználói ablakok: lehetőség egyedi ablakok elhelyezésére a Visio 2000 többdokumentumos felület *(multiple-document interface - MDI)* keretben. Ez biztosítja a fejlesztőnek az alkalmazástól függő ablakok hozzáadását a Visio 2000 felhasználói felületéhez, és azt, hogy ezeket a Visio 2000 kezelje.
- Teljesítmény: megnövelt teljesítmény, valamint számos ShapeSheet és API bővítés.

Minden Visio 2000 megoldás a végfelhasználók számára is elérhető API-kkal és módszerekkel lett kifejlesztve.

Egy program az összes diagramkészítési feladathoz

A vállalatok egyre inkább szabványosítják széles körben használt programjaikat. Néhány példa, ahol a Visio 2000 használatával hatékonyabbá tehetők a vállalati tevékenységek: újrahasznosított ismeretek, infrastruktúra felügyelet, szoftver- és adatbázis-fejlesztés, üzleti folyamatok javítása, minőségbiztosítási rendszerek.

A Visio 2000 sok diagramfajta elkészítéséhez használható, automatizálásának lehetősége pedig biztosítja a diagramok adatokból való létrehozását, valamint információk összegyűjtését és adatbázisba mentését. A létrehozható alkalmazások bonyolultságára maga a Visio 2000 a legjobb példa, amelyet Visio technológiával fejlesztettek ki, és azok az alkalmazások, melyeket független gyártók a Visio 2000 platformon építettek.

Integráció a nagyvállalati információkkal

A vállalatok keresik az ERP rendszereikben felhalmozódott rengeteg adat felhasználásának módját. A Visio 2000 a legjobb megoldás arra, hogy ezekből az adatokból használható képi információt „varázsoljon", és ezzel jelentést adjon nekik. A Visio 2000 például automatikusan képes szervezeti diagramot készíteni az ERP rendszerben tárolt jelentési struktúrából, vagy adatot nyerhet ki egy Visio 2000 diagramon feltüntetett ipari rendszerről, hogy automatikusan anyagszámlákat és költségbecsléseket készíthessen.

Kisebb birtoklási költség

A legtöbb vállalat célja a legnagyobb érték megszerzése a lehető legalacsonyabb áron. Hogy segítse ennek a megvalósítását, a Visio 2000 meglévő és kialakulóban levő szabványokra épül, ezzel csökkentve a nagyvállalat számára a Visio 2000 teljes birtoklási költséget.

A legtöbb nagyvállalatnál a Visio 2000 szoftvert vagy a központi IT osztály helyezi üzembe és felügyeli, vagy az egyes szervezeti egységek alakítanak ki maguknak egyedi beállításokat. Az ilyen vállalati környezetben nagy segítséget jelent a Visio 2000 rugalmassága és egyszerű telepíthetősége.

Kisebb telepítési és üzembehelyezési költség

- A Visio 2000 a Microsoft Windows Installer-t használja, mely segíti a megfelelő összetevők megfelelő időpontban történő telepítését.
- A rendszergazdák megadhatják, hogy a Visio 2000 összetevői vagy fájljai az ügyfélgépeken, vagy a kiszolgálón helyezkedjenek el. Ez jobb erőforrás-kezelést eredményezhet.
- A Visio 2000 telepíthető Windows Terminal Server-re, így több ügyfél használhatja egyszerre. Ez a módszer a központosított rendszerfelügyelet miatt csökkenti az üzembehelyezési és karbantartási költséget.

Kisebb támogatási költség

Egyes felmérések szerint az asztali gépek költségeinek húsz százalékát a végfelhasználók támogatásának költségei teszik ki. A Visio 2000 kiváló súgója csökkenti a támogatási terheket, mert interaktívabb és a beszélt nyelvet is megérti. Emellett a Visio 2000 felhasználói felülete az Office 2000-ehez hasonló, így az Office-t használóknak kevesebb nehézséget fog okozni a Visio használata. Az Office 2000-ben alkalmazott megoldásokat *(például HTML súgó)* integrálták a Visio 2000-be.

Egyszerű licenszelés

A Visio 2000 a Microsoft licenszprogramjai keretében vásárolható meg, melyek egyszerűvé és költséghatékonnyá teszik a nagyvállalatnak a Visio 2000 termékek licenszelését. A licenszelésről részletes információ található a **[1]** címen.

Tanácsadószolgálat és terméktámogatás

További erőforrások segítik a Visio 2000 platformra fejlesztő vállalatokat. A Visio Consulting Services a legjobb Microsoft Visio 2000 szakértők tanácsait és tapasztalatát biztosítja a vásárlóknak. A Microsoft megoldásszállítói is segítséget nyújthatnak a vállalatoknak testreszabott megoldások elkészítésében.

Összefoglaló

A Microsoft Visio 2000 platform a vállalatok alkalmazásait olyan nagyvállalati szintű eszközökké változtathatja, melyek növelik a rugalmasságot és csökkentik a költségeket. Olyan funkcionalitást biztosít, mely csökkenti a PC-k birtoklási költségét, egyszerűsíti az alkalmazások használatát, és az eddigieknél hatékonyabb eszközöket biztosít az információk létrehozására, elérésére és cseréjére.

Mivel kiválóan integrálódik az összes nagyvállalati alkalmazáshoz és erőforráshoz, a Visio 2000 remek platform az üzleti megoldások építéséhez. Új, továbbfejlesztett módszereket tartalmaz, ezzel biztosítva, hogy a vállalat sikeresebb és versenyképesebb legyen.

A cikkben használt URL-ek:

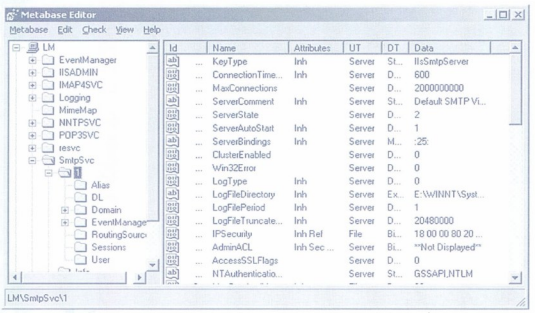
[1] http:// www.microsoft.com/shop

További információk: http://www.microsoft.com/office/visio/

Kérdezz, Felelek!

K: *Hogy tudnám lementeni az Exchange 5.5 mailboxait? Olyasmire gondolok, hogy készülne például egy mailboxnév.pst (személyes mappafájlt kellene csinálni minden mailboxból). Megoldható? Mivel, hogyan?*
V: Erre való a Backoffice Resource Kit segédeszköze, a Mailbox Migration Tool. Ennek segítségével ugyanis tetszőleges Exchange Serverek között lehet leveleslådákat átmozgatni, ami ugyan most válasz a kérdésre, de érdemes tudni, hogy a mozgatáshoz nem kell két futó Exchange Server, mivel az átmozgatás PST fájlokon keresztül történik. De ha már itt tartunk az eszköz nemcsak a leveleket teszi ki PST-be, hanem a szabályokat és a mappahierarchiát is.
Forrás: NetAcademia Exchange 2000 lista

K: *Hogyan lehet az IIS „default virtual SMTP server" könyvtárát megváltoztatni? Ha egy további „virtual SMTP Servert" veszek fel, ott meg lehet adni, de a default egyből, kérdés nélkül az Inetpub alá telepszik. A teljes IIS-t kéne újratelepítenem, hogy rákérdezzen a target directory-ra?*
V: Nem. Ezek az információk az úgynevezett Metabase-ben vannak, ami hasonló a regisztrációs adatbázishoz, s az IIS itt tárol egy csomó információt önmagáról. *(Lásd Metabse cikkünket ugyanebben a lapszámban.)* Gyárilag a Metabase a winnt\system32\inetsrv\adminsamples könyvtárban lévő scriptekkel módosítható, ami elég kellemetlen, de szerencsére van grafikus eszköz is, a Metabase Editor! Letöltés az [1] címről. Aki ezt hibátlanul begépeli, megérdemli az eszközt!):



Forrás: NetAcademia Windows 2000 lista

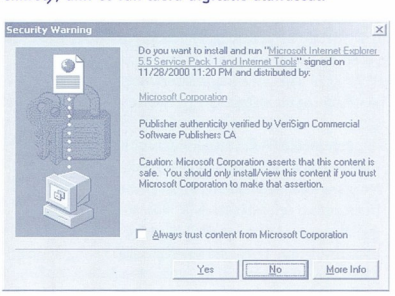
K: *Nagyon tetszik a Terminal Services, használjuk is rendesen, különösen hasznos a helyi erőforrások „felmeppelése" és a helyi és távoli felület közös vágólapja. Már csak egy dolog hiányzik a boldogságomhoz: fájlokat nem lehetne átvinni a két rendszer között?*
V: A Windows 2000 Resource Kit része az RDPCLIP.EXE, kiegészíti a Terminal Services beépített RDP protokollját, hogy az képes legyen fájlokat átvinni. Míg az eredeti kiadásban ha az egyik felületen *(pl. a távolin)* megkíséreltünk lemásolni egy fájlt *(CTRL+C)*, a másikon semmit sem illetszt be a rendszer, addig e kiegészítő telepítése után a vágólap komplett fájlokat is át fog vinni. CTRL+C itt, CTRL+V ott és kész. A [2] címről letölthető az ingyenes változat – amely az én kísérleteim során csökönyösen kéregette a ResKit CD-t, tehát mégsem ingyenes :-O
Forrás: NetAcademia Windows 2000 lista

K: *Hogyan lehetne TSQL programból egyértelműen megállapítani a bejelentkezett felhasználó nevét?*
V: Nincs olyan, hogy egyértelmű, ugyanis az SQL Server felhasználói eleve kétfélék lehetnek: Windows fiókok és csoportok valamint SQL felhasználók. Továbbá egy bejelentkezett felhasználó más és más néven szerepelhet az adatbázisokban, emlékezzünk csak a sa=dbo aliasra. Van egy csomó rendszerfüggvény a felhasználó azonosságának megállapítására, de mind másra való. A megszokott SUSER_NAME() például már semmire, SQL 2000-nél mindig NULL-t ad vissza, ne használjuk. Windows hitelesítéssel *(egyébként sysadminként)* bejelentkezve a következő válaszokat kapjuk:

```
select user_name()
  dbo --az aktuális adatbázis USER-e
select suser_sname()
  NETACADEMIA\fm -- A loginnév
```

Forrás: NetAcademia SQL 2000 lista

K: *A Security Bulletin által javasolt gyorsjavítások nincsenek digitálisan aláírva. Ergo pont a biztonsági patchek hamisíthatók. Ez kicsit fura nemde?*
V: Dede. De a Microsoft javuló tendenciát mutat. Egyik listatagunk hívta fel arra a figyelmünket, hogy van már olyan hotfix *(IE 5.5 SP1 utáni hotfix, letölthető a* [3] *címről)*, ami el van látva digitális aláírással.



Bízzunk benne, hogy a jövőben ez a módszer válik általánossá!
Forrás: NetAcademia Security lista

K: *a gépemen lévő Outlook 2000 kicsit furcsán kezd viselkedni... Először úgy töltötte le a leveleim, hogy az összes levél subjectje üres volt, a feladás dátuma pedig az épp aktuális rendszeridő, a levél tartalma pedig a teljes e-mail (internet fejléccel, mindennel együtt). Ezután egyszerűen nem volt hajlandó megnyitni az üzeneteket, mondván, hogy sérült a pst file, lépjek ki és futtassak egy bizonyos Inbox Repair Toolt... Kérdésem az lenne, hogy mi ez az eszköz, és hol találom?*
V: SCANPST.EXE, és a Microsoft Office 2000 legelső CD-jén található. A PST fájl belső felépítése hasonló a JET adatbázisokhoz – pointerek mindenfelé. Ha ez elromlik, akkor az Outlook sem tehet mást, mint hogy zagyvalékot mutat. Az Inbox Repair Tool azonban nem csodaeszköz. Láttam már olyat, hogy a javítása annyiból állt, hogy üresre radírozta a PST fájlt. Használata előtt illik menteni!
Forrás: NetAcademia Exchange 2000 lista

K: *Hogyan lehetne weblapról tetszőleges címről/címre levelet írni? A lehető legegyszerűbb megoldás érdekelne!*
V: Egyszerű és rugalmas – ez az ASP! Az alábbi négysoros programka kerüljön bele egy KULDUNK.ASP nevű weblapba, s az meg egy olyan könyvtárba, ahol a scriptvégrehajtási jog engedélyezve van:

```
<html>
<%
  Set objNewMail=
    Server.CreateObject("CDONTS.NewMail")
  objNewMail.Send Request("feladó"),
    Request("cimzett"), Request("targy"),
    Request("level")
  Set objNewMail = Nothing
  ' canNOT reuse it for another message
%>
Kész.
</html>
```

Ezt azután a következőképpen kell meghívni: [4]. Természetesen a gyakorlottabbak felismerték, hogy itt a CDONTS könyvtár NewMail objektumát babráltuk. Ez pont arra való, hogy a lehető legkevesebb macerával lehessen programból levelet írni. ***(Legsikeresebben akkor használhatjuk, ha a webkiszolgálón fut egy SMTP szolgáltatás is, mert akkor az leveszi a levélküldés terhét a MAPI kliens válláról – erről a témáról az ASP suli című cikksorozatunk egy későbbi részében részletesebben is beszélünk majd. – [mICK])*** **Természetesen a módszer bonyolultabb levelek megírására is alkalmas, hisz az objektumnak nem csak ez a négy-öt paramétere van. Részletesebben az [5] címen olvashatunk róla.**
Forrás: NetAcademia MsDev lista

A cikkben található URL-ek:
[1] http://support.microsoft.com/support/kb/articles/Q232/0/68.ASP?LN=EN-US&SD=gn&FR=0
[2] http://www.microsoft.com/windows2000/library/resources/reskit/tools/hotfixes/rdpclip-o.asp
[3] http://www.microsoft.com/windows/ie/download/critical/q279328/default.asp
[4] KULDUNK.ASP?felado=akarki@aa.hu&cimzett=marcellf@netacademia.net&targy=akarmi&level=szoveg
[5] http://msdn.microsoft.com/library/psdk/cdo/_denali_newmail_object_cdonts_library_.htm

A haladás csak idő kérdése

A napi munkám egy része előrejelzések készítése, hogy ezek alapján tudjunk cselekedni. Jobb vagyok a jövőbeli trendek előrejelzésében, mint annak meghatározásában, hogy az adott technológiák mikor terjednek el széles körben. Az időzítés előrejelzése nehéz, ahogy azt pályám is mutatja.

Például az 1980-as évek elején megjósoltam, hogy a grafikus felhasználói felület *(melyet először a Xerox fejlesztett ki, és később az Apple Macintosh és a Microsoft Windows tette népszerűvé)* lesz a fő felület az ember és a számítógép kapcsolatában. A jóslat helyes volt, de több évet vett igénybe, mint gondoltam.

1986-ban úgy gondoltam, hogy a CD-ROM meghajtók nagyon rövid időn belül általános célúak lesznek. Igazam volt abban, hogy a nagy tárolókapacitású lemezek általánossá válnak, de túl optimista voltam azzal kapcsolatban, hogy ez mikor történik meg.

A kilencvenes évek elején azt jósoltam, hogy az interaktív hálózatok fogják eljuttatni az információkat az emberek számítógépeire, televízióiba és egyéb információkezelő eszközeire. Ezt „kéznél levő információ"-ként emlegettem, de tévesen ítéltem meg ennek formáját, és gyorsan kellett reagálnom, amikor az Internet népszerűsége 1995-ben elérte a kritikus tömeget.

Nemrégiben, 1997. elején 15 előrejelzést mondtam arra az évre. Tíz igaz lett, például az is, hogy jól használható PC-k kaphatók lesznek 1000 USD alatti áron. Őt túl optimistának bizonyult, de még mindig bízom benne, hogy megvalósulnak – csak idő kérdése.

1998. elején két új előrejelzést tettem. Azt jósoltam, hogy az év második felében a DVD és a DSL fontossá válik. Ezek a technológiák fontossá váltak az elmúlt hónapokban, de bevallom, nem érték el azt a népszerűséget, amire számítottam.

A DVD Digital Versatile Disk-et *(általános célú digitális lemezt)* jelent, bár eredetileg Digital Video Disk-nek hívták. A DVD tulajdonképpen a CD-ROM utódja, és úgy is néz ki, mint egy CD-ROM. Még a CD-ROM-ok és audio CD-k is olvashatók a DVD meghajtóval.

A különbség az, hogy a DVD a CD-ROM-on tárolható információmennyiség többszörösét képes tárolni, vagyis egyetlen lemezen elfér egy VHS-nél jobb minőségű kétórás film. Egy DVD meghajtóval felszerelt PC segítségével a film lejátszható a gép saját képernyőjén, vagy akár a TV-n is, egy DVD-s laptop segítségével pedig akár a repülőn is nézhetünk filmet.

Tudtam, hogy 1998. nagy részében még kevés DVD-t fognak eladni, de azt hittem, mostanra nagy áttörés lesz ezen a téren. Túl optimista voltam.

A DVD meghajtók kezdenek feltűnni a csúcskategóriás PC-kben, és az új PC-k kiegészítőiként. Egy DVD meghajtó általában 50 és 200 dollár közötti többletköltséget jelent, és a CD-ROM meghajtót váltja ki. Én a DVD megvásárlását javaslom, bár egyelőre még kevés szoftver jött ki DVD-n, de vannak figyelemreméltó kivételek. Egyre több film vásárolható meg, néhány játékot és referenciamunkát kiadnak DVD-n. A National Geographic az elmúlt 108 év folyóiratait egy olyan DVD csomagban adta ki, mely elfér a kezemben. A DVD -ken a szövegen kívül több mint 180.000 fénykép van.

A másik rövidítés, a DSL Digital Subscriber Line-t *(digitális előfizetői vonal)* jelent. Ez egy olyan technológia, amely lehetővé teszi, hogy nagy mennyiségű digitális adat legyen átvihető a hagyományos telefonvonalakon.

„A DSL próbaüzemei már folyamatban vannak az Egyesült Államokban, de 1998. vége előtt nem lesz széles körben hozzáférhető a szolgáltatás" mondtam 1998 elején.

Ismét túl optimista voltam. A DSL üzembe-helyezése néhány helyen folyamatban van, és vannak rajongói, akik szeretik a gyors Internet elérést, de a telefontársaságok többet kérnek a szolgáltatásért, mint amire számítottam, és ez akadályozza a DSL elterjedését. Ugyanígy nehezíti a kábelmodemek terjedése, amelyek a telefonvonal helyett a kábeltelevízió hálózaton keresztül biztosítanak nagysebességű Internet elérést. Ennek eredményeként a DSL jelentősége sokkal kisebb, mint amire számítottam.

Szerencsére néhány találmány gyorsabban népszerűvé válik, mint ahogy gondolnám. Van erre egy remek példám, ami 1998-ban történt.

Év elején azt hittem, hogy a nyomtatott szöveggel versenyképes felbontású szöveg megjelenítésére képes sík képernyő megvalósítására még legalább hat évet kell várnom. Alig vártam azt a napot, amikor az emberek könnyű eszközökön lesznek képesek magukkal vinni számtalan újságot, könyvet, vagy akár filmet is.

Most, nagy meglepetésemre úgy tűnik, hogy csak két évet kell várnom! A Microsoft kutatói ugyanis néhány hónapja bemutattak egy ClearType-nak nevezett technológiát, amely lehetővé teszi a szöveg LCD képernyőn való igen jó minőségű megjelenítését. Ránéztem a képernyőre, és azt mondtam magamban, hogy ezt képes lennék egész nap olvasni!

A ClearType a gyengém, mert arra számítottam, hogy hardverfejlesztések szükségesek ahhoz, hogy az LCD kijelzők képesek legyenek a szöveg valóban éles megjelenítésére. De a ClearType tisztán szoftveres megoldás, amely szükségtelenné teszi új hardver kifejlesztését. A meglevő laptopok Cleartype-ban fogják megjeleníteni a szöveget, amint a szoftver az operációs rendszerbe integrálva lesz.

Ahogy új év kezdődik és lenézek az útra, kicsinek és lelkesnek érzem magam. Kicsinek, mert képtelen vagyok megjósolni az új technológiák *(például DVD és DSL)* elterjedésének ütemét. Lelkesnek az iparág teljes fejlődési üteme miatt, amely megállíthatatlanul halad előre, megváltoztatva életmódunkat és munkahelyünket.

Az idén óvatos leszek, és csak egyvalamit jósolok, de ez biztos: A jó szoftverek, melyeket nagyon nehéz létrehozni, egyre fontosabbak lesznek a jövőben. A részletek homályosak lehetnek, de a teljes kép olyan éles, mint a ClearType.


Development Platform
msdn

Hallott már Ön selejtmentes gyártásról? Rendszer-felügyeletmentes hálózatról (ZAK, ZAW)? Feltörhetetlen hálózatról (C2)?

Elérhetetlen, de megközelíthető célok. Az első kettő elérésében sajnos nem segíthetünk. De a harmadik cél megközelítésében sokat tehetünk Önért. Világszínvonalú, egyedi Security tanfolyamainkon megtudhatja, mitől döglik a hacker!

1. Biztonsági technológiák

rendszergazdáknak, 3 nap

- Titkosítási algoritmusok *(DES, RSA, MD5 stb.)* elmélete és gyakorlata
- Az authentikáció biztonsága *(Kerberos, NTLM, NTLMv2)*
- A nyílt kulcsú titkosítási technológia nagyvállalati, gyakorlati alkalmazása *(Certificate Server, SmartCard, biometrikus azonosítás)*
- Biztonságot növelő technológiák bevezetése *(IPSec, VPN, L2TP, Encrypting File System stb.)*

2. Biztonságos levelezőrendszer kialakítása MS Exchange alapokon

gyakorlott Exchange endszergazdák számára, 3 nap

- Titkosítás és digitális aláírás az elektronikus levelezésben (S/MIME)
- Az Exchange Server védelme, adminisztratív szerepek, jogosultságok, mentés, visszaállítás, journaling
- tűzfal vs. Exchange Server, SMTP relay beállítások, spam védelem, SSL, TSL, nyílt kulcsú titkosítás felhasználása, MAPI, POP3, IMAP, OWA, LDAP protokollok veszélyei.

3. Hálózatunk védelme a támadásokkal szemben

biztonsággal foglalkozó szakembereknek, 3 nap

- Kaloz-eszközkészlet *(Denial Of Service, Buffer Overrun, hálózatelemzés, trójai falovak, vírusok, jelszófeltörés, hátsó ajtók)*
- A Windows 2000 biztonsági elemei *(fájlrendszerek, regisztrációs adatbázis, hálózati adatforgalom)*
- Védekezés: tűzfalak, proxyk
- Aktív védekezés: Intrusion Detection

4. Programozzunk biztonságosan!

programozók számára, 3 nap

- Elmélet: védett mód, kernel, memóriakezelés
- Programozási gyakorlatok, rosszul és jól megírt kódok
- A Buffer overrun hatásai user és *(app es service)*+kernel módban
- RFC implementation errors
- JAVA, ActriveX, homokozó, scriptek
- Worm vírusok elemzése *(Iloveyou)*
- CryptoAPI, Smart Card API

Biztonsága érdekében keressen minket!

www.netacademia.net

A legjobbakat tanítjuk.

1105 Budapest, Ihász utca 13. • Tel.:263-2732

NETACADEMIA