Microsoft® TechNet

100% technológia ▪ 0% marketing

A tűzfal logja

V./2. szám
2004. június

ISSN 15845185 02
9 771586 518005

Tippek és trükkök –
Fókuszban az SQL Services 2000
MVP csúcstalálkozó
Fejlesztésmenedzsment módszertanok
Microsoft vizsgák és minősítések

Legyen Ön is a NAGYok között!

Windows Server 2003 alapú hivatalos Microsoft mérnök képzések

Hivatalos Microsoft tanfolyamokra épülő, rugalmas beosztású, kedvezményes konstrukciójú és árú képzéssorozatok. Különböző segédanyagokat, vizsgafelkészítő anyagokat tartalmazó oktatási konstrukciók és csomagok. Kedvezményes lehetőségek további tanfolyamokra, vizsgákra és felkészítő anyagokra.

Válassza ki az Önnek megfelelő minősítést és képzési konstrukciót!

Microsoft rendszeradminisztrátor (MCSA) képzés

A kisebb Windows Server 2003 alapú informatikai rendszereket és hálózatokat üzemeltetni kívánó szakemberek számára ajánljuk.

Két tanfolyam (80 óra) – 249.000 Ft+ÁFÁ-tól

Microsoft rendszermérnök (MCSE) képzés

Nagy, összetett IT rendszereket és hálózatokat üzemeltető szakemberek képzése, akiknek feladatuk lesz Windows Server 2003 alapú hálózatok teljes körű adminisztrálása és támogatása, informatikai infrastruktúrák és folyamatok tervezése.

Négy + 1 tanfolyam (160 óra) – 499.000 Ft+ÁFÁ-tól

Kezdési időpont: június 21.

A tanfolyamok mindegyikére beváltható a Microsoft frissítési garancia tanfolyambón (SA voucher) is.

Valamennyi tanfolyam szakképzési hozzájárulásból elszámolható!

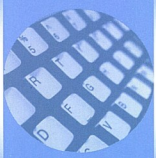
Jelentkezzen, amíg a szabad helyek tartanak!

A tanfolyamokkal és akciókkal kapcsolatos további tudnivalók weboldalunkon találhatók, vagy kérjük, keresse szervezőnket!

Microsoft CERTIFIED Professional

Microsoft GOLD CERTIFIED Partner

Microsoft CERTIFIED Technical Education Center



SZÁMALK

www.szamalk.hu/tisza

Tel.: 203-0304/3050 (Simon Ferenc)
1115 Budapest, Etele út 68.

Microsoft TechNet

TechNet Magazin
V. évfolyam, 2. szám
2004. június

Szerkesztőség és kiadó:
Microsoft Magyarország Kft.
1031 Budapest, Graphisoft park 3.

Felelős kiadó:
Székely Tamás marketingigazgató

Szerkesztő:
Takács Gitta (Epszilon Press)

Szaklektor:
Fóti Marcell (Netacademia)

Lapmenedzser:
Kolma Kornél (Microsoft Magyarország)

Lapterv és nyomdai előkészítés:
Dobák Ildikó (Ars Luna Bt.)

Nyomda:
AduPrint Kiadó és Nyomda Kft.
1033 Budapest Csikós utca 8.
Felelős vezető: Tóth Béláné

webcím:
www.microsoft.com/hun/technet/
E-mail:
technetmagazin@microsoft.hu

ISSN 1586-5185

A Microsoft megnyitotta Tudásbázisát az MVP-k előtt

Ezerötszáz MVP (Most Valuable Professional), azaz Microsoft technológiában jártas szakértő részére tartott csúcstalálkozót idén áprilisban a Microsoft az Egyesült Államokban, melynek keretében több fontos bejelentésre is sor került. A Microsoft vezetői bemutatták az egyelőre próbaüzemben működő Community Solutions Content Program for Microsoft Knowledge Base nevű programot, amely lehetővé teszi az MVP-k számára, hogy a Microsoft termékekkel és technológiákkal foglalkozó Microsoft Tudásbázis (Microsoft Knowledge Base) részeként működő Community Solutions webhely számára tartalmat készítsenek. Új információk hangzottak el a 2003 októberében beindított Microsoft MVP Forráskód-licencprogramról is. A jogosult országokból eddig több mint 200 MVP szerzett licencet a Megosztott Forráskód Kezdeményezés keretében, akik ezáltal hozzáférhetnek a Microsoft vagyonának egyik legértékesebb darabjához, a Windows forráskódjához.

A csúcstalálkozó keretében olyan új segítő és közösségi eszközöket is bemutattak, amelyek új visszajelzési csatornát biztosítanak az MVP-k számára. Az ügyfelek ezeken keresztül nyújthatnak be a termékekkel és azok funkcióival kapcsolatos javaslatokat és megvitathatják a felmerült ötleteket még a termékfejlesztés szakaszában. A visszajelzések közvetlenül a Microsoft adott technológiáért felelős csoportjához futnak be, mely ezután reagál a legtöbb szavazatot összegyűjtő javaslatokra.

A Microsoft a kilencvenes évek elején indította el a Most Valuable Professional (MVP) Award Programot, azzal a céllal, hogy elismerje a nagyközönség azon tagjainak munkáját, akik idejüket és informatikai szakértelmüket a többi felhasználó megsegítésére fordítják. Az indítás óta eltelt idő alatt folyamatosan nőtt a Microsoft MVP Award Program tagjainak száma. A kezdeményezés mára több mint 2500 tagot tudhat sorai között, akik 63 országot és 20 nyelvterületet képviselnek, ismereteik pedig több mint 70 Microsoft-technológiára terjednek ki. Az MVP-k mélyreható és szerteágazó ismeretekkel rendelkeznek legalább egy Microsoft-termékről, és ismereteiket szívesen osztják meg másokkal. A legtöbb MVP több mint 15 éves technikai tapasztalattal rendelkezik saját szakterületén, férfiak és nők egyaránt vannak köztük, a legfiatalabb 13 éves, a legidősebbek pedig hatvanas éveik közepén járnak. Vannak köztük publikáló írók, webhelykezelők, ismeretterjesztő előadók, oktatók és professzionális fejlesztők. Ha MVP nyújt technikai segítséget egy online vagy offline közösségben, a többiek és az ügyfelek azonnal tudják, hogy egy tapasztalt és hozzáértő személytől kapnak kiváló tanácsot.

A díjakat a Microsoft technikai közösségének legkiemelkedőbb tagjai kapják, cserébe azért, hogy hozzájárultak több száz online és offline technikai közösség ismereteinek növeléséhez Microsoft nyilvános hírcsoportokban, külső működtetésű webes hirdetőtáblákon, webes naplókban és felhasználócsoportokban, amelyek mind nagy népszerűségnek örvendenek a Microsoft termékeivel, technológiáival és szolgáltatásaival kapcsolatban egymással kommunikáló felhasználók körében.

Magyarország tavaly csatlakozott a nemzetközi MVP Programhoz. Jelenleg nyolc főből áll a hazai MVP-k köre, akik közül hatan vettek részt az amerikai csúcstalálkozón.

http://www.microsoft.com/mvp/
http://www.microsoft.com/communities/

Tartalom

MVP-találkozó Amerikában

Újdonságok a .NET Framework 2.0-ban

A tavaszi amerikai MVP-találkozón számos újdonságot hallhattunk a jövő technológiáiról. Cikkünkben elsősorban a Visual Studio 2005 és a .NET Framework 2.0 (Whidbey) újdonságairól számolunk be.

Tippek & trükkök

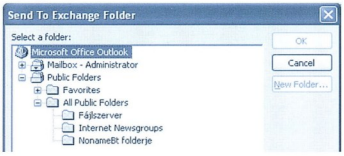
Fókuszban az SQL Server 2000

Egy-egy trükk megismerése sokat segíthet a mindennapi munkában. Cikkünkben az SQL Server 2000 használatához adunk tippeket.

Ami a hivatalos Microsoft tanfolyamokból kimaradt...

Exchange 2000/ Exchange Server 2003 II. rész

Az előző számunkban közölt cikk folytatásaként újabb Exchange-érdekességeket, tanulságokat ismertetünk.



Vizsgák és minősítések

Microsoft oktatási tájékoztató

A Microsoft kínálatában több mint negyven féle technikai vizsga van, amelyeket a hivatalos Microsoft oktatóközpontokban lehet letenni. Összeállításunkban részletesen ismertetjük a Microsoft vizsgák és minősítések rendszerét.

Dr. Watson

Ne higgy a lognak!

Mármint a tűzfalak logjának. Ebben a cikkben alaposan utánajárunk annak a bosszantó jelenségnek, amikor egy tűzfalgazda kiszúrja, hogy egy távoli IP-címről bizonyos TCP-portokat „feszegetnek"; vagy netalán portscannelést hajtottak végre. Azt hihetnénk, a támadó személyének vagy helyének felkutatására a legjobb információforrás a tűzfal naplófájlja. Az ottani IP-cím alapján rövid kutatást végezve kiderül, hogy a www.érdektelen.hu webszerver a támadó, ami gazdátlanul lóg a neten, láthatóan ép és egészséges, nincs feltörve. Elkövető tehát nincs. Ez hogy lehet?

Automatikus memóriagazdálkodás a .NET-ben 1. rész

Objektumok születése és halála

A .NET szemétgyűjtője, a Garbage Collector az esetek nagy többségében észrevétlenül és hibátlanul teszi a dolgát. Előfordul azonban, hogy ismernünk kell a GC belső működésének részleteit is, hogy alkalmazásunk hatékonyan tudjon együttműködni vele.

Az alkalmazások korrekt erőforrás-kezelésének megvalósítása nem egyszerű feladat, komoly odafigyelést és gyakorlatot igényel. Az erőforrások (és a hozzájuk tartozó memóriaterület) lefoglalásával és felszabadításával kapcsolatban elkövetett hibák nehezen azonosítható, rejtélyes hibákat okozhatnak az alkalmazás használata során.
A C++ nyelven fejlesztett programok esetében például a lefoglalt memória felszabadítását maga az alkalmazás kezdeményezi, az objektum számára new operátorral lefoglalt memóriát a megfelelő pillanatban a delete operátor segítségével fel kell szabadítani. A módszer természetesen tökéletes, egészen addig, amíg a megvalósítás hibátlan. Egy nagyméretű, több szálon futó alkalmazás esetében azonban egyáltalán nem könnyű a pontos kivitelezés, a memóriában „felejtett" referencia nélküli objektumok memóriahézagokat (memory leak), a már felszabadított memóriaterületre való hivatkozás pedig futási hibát okoz.

A COM komponensek a hivatkozások számlálásával olják meg az általuk használt erőforrások felszabadítását. Az új hivatkozások megnövelik, a megszűnő referenciák pedig csökkentik egy számláló értékét. Ha a számláló értéke nulla, a komponensre már nincs szükség, az lezárja az erőforrásokat, és felszabadítja a memóriát. A módszer gyenge pontja az, hogy a számláló beállítását a komponenshez csatlakozó ügyfeleknek kell(ene) kezdeményezniük. Ha ezt csak egy is elmulasztja, a komponens akár az idők végezetéig is fölöslegesen a memóriában maradhat.
Általánosságban is elmondható, hogy nem lehet tökéletes az a megoldás, ahol egy szűkös erőforrással (esetünkben a memóriaterülettel) való gazdálkodást maguk az erőforrás felhasználói irányítják.

Minden objektum (és az általa megvalósított erőforrás) eléréséhez (használatához) a következő lépéseket kell elvégezni:

- Memóriafoglalás (memory allocation)
- A lefoglalt memória kezdeti értékének beállítása (memory inicialization)
- A létrehozott objektum használata (memory use)
- Az objektum lezárása (memory tear-down)
- A memóriaterület felszabadítása (memory freeing)

A .NET által biztosított Garbage Collector (GC) teljes mértékben magára vállalja a hivatkozás nélkül maradt objektumok felkutatását és az általuk lefoglalt memóriaterület felszabadítását. Ha azonban az objektum a CLR hatáskörén kívül álló erőforrásokra is hivatkozik (unmanaged resources), azokat a GC nem tudja megfelelően lezárni. Ilyen erőforrások jellemzően az operációs rendszer által biztosított objektumok: megnyitott fájlok, hálózati vagy adatbázis-kapcsolatok stb. Ilyen esetben a lezárást elvégző kódot a programozónak kell megírnia az objektum Close, Dispose, vagy Finalize metódusában, a Finalize metódus megfelelő pillanatban való meghívásról viszont már a GC gondoskodik.

Memóriafoglalás

A .NET CLR-je minden erőforrást az adott folyamat számára létrehozott menedzselt heap-en helyez el. A menedzselt heap hasonló a hagyományos programnyelvek által használt heap-hez, de az itt létrehozott objektumokat nem a program szünteti meg, a memória felszabadítása automatikusan történik, ha az adott objektumra már nincs többé szükség. Hogy ezt megtehesse, a GC-nek természetesen tudnia kell, hogy mely objektumokra nincsen már szüksége a folyamatnak. A szükségtelen objektumok felderítésére használt algoritmust később részletesen meg fogjuk tárgyalni.
Előbb azonban tekintsük át, hogyan is zajlik a memóriafoglalás a CLR felügyelete alatt. Amikor új folyamat indul el, a CLR folyamatos memóriaterületet (address space) foglal le a számára. Ez a memóriaterület a folyamat menedzselt

heap-je. A heap-hez egy mutató is tartozik, ami a következő létrehozható objektum kezdő címére mutat, kezdetben tehát az adott memóriaterület kezdő címére (base address). Ez az állapot természetesen sohasem látható, mivel már az alkalmazást magát reprezentáló objektum is a heap-en foglal helyet. A folyamat objektumai által elfoglalt memóriaterületet (bájtokban) a GC osztály statikus metódusával kérdezhetjük le:

```
System.GC.GetTotalMemory(false);
```

A false paraméter azt jelenti, hogy a GC nem fog begyűjtést végezni a szabad memória meghatározása előtt, tehát a kijelzett értékben benne lesz a már használhatatlan objektumok által elfoglalt terület is.

Memóriafoglalás a menedzselt heap-en

A new operátor segítségével a heap-re helyezett objektumok mindig a mutató által meghatározott címre fognak kerülni (természetesen ilyenkor a mutató is módosul, hogy továbbra is a szabad hely kezdetére mutasson). Nincsen tehát szükség a hagyományos heap-nél megszokott helykeresésre, így a memóriafoglalás lényegesen egyszerűbb és gyorsabb. Azonban hogy a memóriafoglalás ilyen módon működhessen, azt kell feltételeznünk, hogy a mutató által meghatározott cím után mindig lesz elegendő szabad hely a létrehozandó objektum számára. Ennek biztosítása a GC feladata.
Amikor az alkalmazás a new operátor használatával létre akar hozni egy objektumot, természetesen előfordulhat, hogy az objektum már nem fér el az adott címterületen. Ha a szabad terület kezdetét jelző mutató és a létrehozandó objektum méretének összege már a címtéren kívül esik, a heap megtelt. Ilyenkor lép akcióba a GC, felfüggeszti az összes szál futását, és eltávolítja a hivatkozás nélküli objektumokat. Ezután a megmaradt, még használható objektumokkal folyamatosan feltölti a címteret (elvégzi a hivatkozások megfelelő módosítását is), tehát a címtér felső részen folytonos szabad címtartomány keletkezik, ahová az új objektum már elhelyezhető.
Az **[1]** címről letölthető mintaprogramot fogjuk használni a leírtak gyakorlati demonstrálására. A program konzol alapú, menüszerkezete a következő lesz:

```
Parancssor - gc

D:\>gc
==========================================
0 - Lefoglalt memória
1 - A fő szál hash kódja
2 - Szemetelés
3 - Szemétgyűjtés
4 - Finalization queue
5 - Újraéleszthető objektum
6 - Objektum generációk
7 - Gyenge referencia
8 - Dispose használata
9 - Kilépés
```

A mintaprogram menüszerkezete

Van egy AlapObjektum osztályunk, amelyből majd más osztályokat fogunk örökíteni. Kódja a következő:

```
class AlapObjektum:Object {
   protected String nev;

   public AlapObjektum(string s)
{     Console.WriteLine("Az alap objektum
↵     osztályában megadott konstruktor fut.");

   ~AlapObjektum() {
      Console.WriteLine("Az alap objektum
↵     osztályában megadott Finalize metódus
↵     fut.");
   }
}
```

A nev mezőt az utódok használják, minden objektumnak saját neve lesz, hogy létrejöttüket és megszűnésüket nyomon követhessük.
Örökítünk belőle egy Szemet osztályt, ilyen objektumokat fogunk majd létrehozni és a GC segítségével megszüntetni. A Szemet osztály kódja a következő:

```
class Szemet:AlapObjektum {
      public Szemet(string s):base(s) {
         this.nev=s+". Szemét objektum";
         Console.WriteLine("A(z) "+nev+"
↵     konstruktora fut.");
      }

      ~Szemet() {
         Console.WriteLine("A(z) "+nev+"
↵        Finalize metódusa fut.");
         Console.WriteLine("A Finalize szál hash
         kódja: " +Thread.CurrentThread..
↵        GetHashCode());
      }
}
```

Az objektumokat létrehozó kód a *Szemetelés* menüpont segítségével indítható:

```
static void Szemetel(){
      Szemet sz;
         for (int i=0;i<100;i++)
         sz=new Szemet(i.ToString());
}
```

Minden objektum paraméterként egy sorszámot kap, ez lesz a neve, ez fog szerepelni a konstruktorok és destruktorok üzeneteiben.

Az új objektumok létrejöttét a konstruktorok képernyőre írt üzeneteinek alapján követhetjük nyomon, és a fenti kódból az is látható, hogy az objektumok létrehozásuk után azonnal referencia nélkül maradnak. Ha a folyamat közben a heap megtelik, az új objektumok létrehozása átmenetileg leáll, és a GC automatikusan nekikezd a hivatkozás nélküli objektumok felszámolásának (az utoljára létrehozott objektum kivételével valamennyi ilyen). Ekkor a destruktorok üzeneteit láthatjuk a képernyőn. A takarítás végeztével folytatódik tovább az objektumok létrehozása.

A *Szemétgyűjtés* menüpont segítségével magunk is bármikor elindíthatjuk a GC-t, és a destruktorok üzeneteinek segítségével megfigyelhetjük a „takarítás" folyamatát. A menüpont a következő metódust indítja:

```
static void Collect() {
   Console.WriteLine("A szemétgyűjtés elindult");
   GC.Collect();
}
```

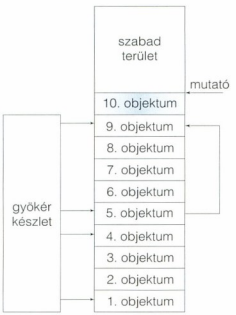
A *Lefoglalt memória* menüpont segítségével ellenőrizhetjük a heap foglaltságát.

A szemétgyűjtés algoritmusa

Feladata elvégzéséhez a GC-nek képesnek kell lennie arra, hogy az alkalmazás által már nem használható objektumokat azonosítsa. Ha ilyen objektumokat talál, az általuk használt memóriaterületet felszabadítja. A következőkben áttekintjük, hogyan történik a referencia nélküli objektumok azonosítása.

Minden alkalmazás rendelkezik egy úgynevezett gyökérkészlettel (set of roots). A gyökerek gyakorlatilag olyan mutatók, amelyek a heap-en lévő objektumokat azonosítanak, értékük vagy egy objektum címe, vagy null. Az alkalmazás gyökérkészletéhez tartoznak az objektumokra mutató globális és statikus adattagok, az adott szál veremterületén elhelyezkedő lokális változókban tárolt referenciák és a heap objektumaira vonatkozó hivatkozásokat tartalmazó CPU-regiszterek is. A gyökérlistát a JIT (just-in-time) fordító és a CLR tartja karban, a GC pedig bármikor hozzáférhet annak tartalmához.

Amikor a GC elindul, azt feltételezi, hogy a heap-en lévő összes objektum szemét, tehát egyetlen gyökér sem tartalmaz élő hivatkozást. Ezután a GC sorra végiglátogatja a gyökérkészlet elemeit, végigköveti az adott gyökérből kiinduló hivatkozási láncot, és gráfot épít az elérhető objektumokból.

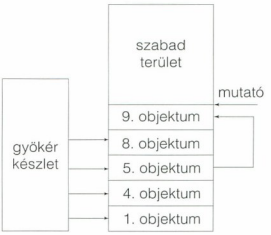


Objektumhivatkozások

Az ábrán látható 1., 4., 5., 8. objektumokra közvetlen gyökérhivatkozás található, így azok felkerülnek a gráfra. Az 5. objektum hozzáadásakor a GC megtalálja az objektumban lévő hivatkozást a 9. objektumra, és így azt is a gráf megfelelő ágához csatolja. Ez azért lehetséges, mert a CLR a heap-en lévő minden objektumról „tudja" annak valódi típusát, a referenciához tartozó metaadatokból pedig meghatározható, hogy az objektum mely tagjai hivatkoznak más objektumokra. A GC az összes elérhető objektumot rekurzívan végiglátogatja.

Ha a GC olyan objektumot talál, amit már hozzáadott a gráfhoz, az adott ágon megállítja a keresést, mivel ebben az esetben az onnan elérhető objektumoknak is már a gráfon kell lenniük. Ez egyrészt jelentősen csökkentheti a keresésre fordított időt, másrészt így elkerülhetők a végtelen ciklusok, amelyeket a körkörös hivatkozások okozhatnak.

Miután a GC a keresést az összes gyökérből kiindulva elvégezte, a gráfnak tartalmaznia kell minden olyan objektumot, amelyek valamilyen módon elérhetőek az alkalmazásból. A gráfon nem szereplő objektumok az alkalmazás számára már nem használhatók, tehát a GC fel fogja szabadítani az általuk elfoglalt memóriaterületet. A memória felszabadítása a legmagasabb címen kezdődik, és az alapcím felé halad. Az eltávolított objektumok helyén ezután lyukak maradnak a címtérben, ezért a megmaradt objektumokat a GC az alapcím közelébe mozgatja, hogy az általuk elfoglalt címtér folytonos legyen. Az objektumok mozgatásával természetesen együtt jár a hivatkozások módosítása is. A GC feladata a gyökérkészletben és az objektumokban lévő referenciák aktualizálása is.



A heap a GC futása után

Ezután a GC beállítja a szabad terület kezdetét jelző mutatót, és folytatódhat az a felfüggesztett művelet (új objektum létrehozása), ami a begyűjtést kiváltotta.

Látható, hogy a referencia nélküli objektumok megkeresése és eltávolítása idő- és erőforrás-igényes művelet, ezt az árat kell fizetnünk a gyors és egyszerű memóriafoglalásért és a memóriakezeléssel kapcsolatos hibalehetőségek kiküszöböléséért. A begyűjtés azonban viszonylag ritkán következik be (csak akkor, ha a heap megtelt), és az objektumok generációkba sorolása (az objektumgenerációkról a második részben részletesen szólunk) is jelentősen csökkenti az egyes begyűjtések időigényét.

A Finalize metódusok

A Finalize metódusok azt teszik lehetővé, hogy az objektumok megfelelően lezárják az általuk használt nem menedzselt erőforrásokat, mielőtt a GC áldozatául esnének. Kis-

sé leegyszerűsítve az eseményeket, azt mondhatjuk, hogy miután a GC memóriaszemétnek minősít egy adott objektumot, meghívja annak Finalize metódusát (ha létezik ilyen), mielőtt felszabadítaná az általa elfoglalt memóriát. A Finalize az Object osztály metódusa, felülírásának szintaktikája a következő (lenne):

```
public class AlapObjektum {
      public AlapObjektum() {
      }
      protected override void Finalize() {
         //nem menedzselt erőforrások lezárása
      }
}
```

A C# fordító azonban nem hajlandó ennek lefordítására, mert nem engedi a Finalize metódus felülírását (a VB.NET fordító igen).
Amikor a fordító egy osztály konstruktorának kódját generálja, automatikusan beilleszti az ősosztályok konstruktorait is. A C++ fordítók ugyanezt teszik a destruktorok esetében is. A Finalize metódusok esetében nem ez a helyzet, ha ilyen viselkedést szeretnénk, a metódus végén explicit módon meg kell hívnunk az ősosztály Finalize metódusát. A C# programnyelvben a C++ destruktorokhoz hasonló szintaktikával helyettesíthetjük a Finalize metódus felülírását és az ős Finalize metódus explicit meghívását:

```
class AlapObjektum {
      ~AlapObjektum() {
         ...
      }
}
```

Nézzük meg a korábban említett és használt Szemet osztály ilyen szintaktikával megadott metódusából generált IL kódot:

```
Szemet::Finalize : void()
    IL_001f:  call       class [mscorlib]System.Threading.Threac
    IL_0024:  callvirt   instance int32 [mscorlib]System.Object:
    IL_0029:  box        [mscorlib]System.Int32
    IL_002e:  call       string [mscorlib]System.String::Concat(

    IL_0033:  call       void [mscorlib]System.Console::WriteLir
    IL_0038:  leave.s    IL_0041
  }  // end .try
  finally
  {
    IL_003a:  ldarg.0
    IL_003b:  call       instance void AlapObjektum::Finalize()
    IL_0040:  endfinally
  }  // end handler
  IL_0041:  ret
} // end of method Szemet::Finalize
```

Az ősosztály Finalize metódusának meghívása

A metódus az IL kódban már Finalize() néven szerepel, és láthatjuk, hogy a fordító a metódus végére beillesztette az AlapObjektum osztály Finalize metódusának meghívását is.
A szintaktikai hasonlóság ellenére azonban a fenti kód nem „igazi" destruktor, nincsen arra lehetőség, hogy a kódból meghívjuk. Osztályaink megtervezésekor, ha csak lehet, kerüljük el a Finalize metódus használatát. Ennek okai a következők:

- A Finalize metódussal rendelkező objektumok (és az általuk hivatkozott összes objektum) memóriaterületének felszabadítása csak két lépésben (a GC kétszeri futásával) lehetséges (ennek okát a második részben részletesen meg fogjuk vizsgálni).
- A Finalize metódussal rendelkező objektumok memóriafoglalása hosszabb időt vesz igénybe.
- Nagyszámú objektum esetén a Finalize metódusok meghívása sok időt vehet igénybe.
- Nem tudhatjuk, hogy mikor fog lefutni az objektum Finalize metódusa, így az alkalmazás hosszú ideig foglalva tarthat olyan külső erőforrásokat (például adatbázis-kapcsolat), amelyek mielőbbi felszabadítása fontos lenne.
- Semmilyen módon nem befolyásolhatjuk a Finalize metódusok meghívásának sorrendjét, ami egymásra hivatkozó objektumok esetén véletlenszerűen jelentkező futási hibákhoz vezethet.

Ha mégis szükséges a Finalize metódus használata, biztosítanunk kell, hogy a kód a lehető leggyorsabban, minden felesleges várakozás (például szálak közötti szinkronizáció) nélkül végrehajtható legyen. A Finalize metódusok helyett a második részben bemutatott Dispose vagy Close metódusok használata javasolt.

A következő részben tovább folytatjuk a Garbage Collector belső működésével való ismerkedést. Szó lesz majd az objektumok „újraélesztéséről", a gyenge referenciákról (weak references) és az objektumok generációkba sorolásáról, amelynek segítségével a GC futása kevesebb erőforrást és időt igényel. Bemutatjuk továbbá a Dispose és Close metódusok helyes megtervezésének módját, valamint megvizsgáljuk a Garbage Collector működésének valós idejű monitorozására szolgáló eszközöket is.

Szerényi László
szerenyi.l@met.hu

A cikkben szereplő URL-ek:

[1] http://store.netacademia.net/mshu/OTHER/technet_code/gc.zip

Fejlesztésmenedzsment módszertanok

Szempontok és nézőpontok

Az informatikai cégek különbözőképpen reagálnak, ha fejlesztésmenedzsment módszertanokról kérdezzük őket. Egy részük azonnal rávágja az alkalmazott metodika nevét és fő jellemzőit, míg a másik véglet némi bizonytalanság után közli, hogy náluk ilyesmire nincs szükség, hiszen mindenki érti a dolgát.

Eddigi pályafutásom során több informatikai projektben vettem részt: ezek között volt olyan, amelyben egy 60-70 fős csapat egyik fogaskereke lehettem, máskor 3 fős fejlesztői gárda egyik tagja voltam, és az is előfordult, hogy a szakmai tervezés lett a feladatom. Így nyugodt szívvel mondhatom, hogy elég sokféle szemszögből láttam már egy-egy szoftver fejlődését.

Megközelítések

Ha megkérdezzük az IT cégeket, milyen módszertant alkalmaznak munkájuk során, gyakorlatilag mindenki másképp válaszol. Nem mintha ilyen kifinomult lenne a vezetés mindenhol, egyszerűen arról van szó, hogy a megvalósításban emberek vesznek részt, akik különböző egyéniséggel, különböző tapasztalatokkal és tudással rendelkeznek, s mindezeket akarva-akaratlanul hozzák magukkal. Ráadásul a csapattagok közötti együttműködésre soha nem tekinthetünk úgy, mint valamilyen gépies folyamatra, amelyben gondolkodás nélkül követik a kiadott instrukciókat, hanem ahhoz mindenki hozzáteszi saját gondolatait, tudását, sőt, az emberek egymás közötti kapcsolata is meghatározó lehet.

Mindezek kezelését gyakorlatilag minden vállalat másképp oldja meg, saját igényeihez, lehetőségeihez és korlátaihoz igazítva. Ezek a megközelítések azonban helyenként hasonlíthatnak egymásra, így véleményem szerint három nagy csoportba sorolhatók:

1. a cégnek nincs kialakult koncepciója,
2. a cég egy az egyben átvesz valamely nagy és közismert módszertant, és azt alkalmazza,
3. a cég saját metodikát dolgoz ki informatikai projektjeinek irányítására és lebonyolítására.

Koncepció nélkül

A koncepció nélküli cégek többnyire kicsi vagy újonnan alapított vállalkozások, bár bőven akad ellenpélda is. Sajnos régóta működő, egyre növekvő vállalatoknál is gyakran megfigyelhetjük, hogy nincs kialakult koncepciójuk, a fejlesztési vezetőkre van bízva, mit és hogyan csinálnak. Természetesen ez egyfajta szabadságot biztosít, ami kreativitásra motiválhatja az embereket.

Mindennek azonban rengeteg hátulütője is lehet: először is, ha nincsenek lerögzítve legalább az alapjai annak, hogy hogyan és milyen munkát várunk, az félreértésekhez vezethet. Ha valaki már évek óta nálunk dolgozik, felgyülemlett már annyi tapasztalata, hogy tudja, mi az, amit elvárunk tőle, és mi az, amit nem szeretünk látni. Ha azonban valakit újonnan vettünk fel, rengeteg konfliktus adódhat abból, ha ő hozza a saját tapasztalatait és elképzeléseit, és azok nem egyeznek a mieinkkel. Ráadásul ha kérdezi, mit és hogyan szeretnénk, gyakran csak néhány ködös mondatban tudjuk azt elmondani neki, ami meglehetősen kevés ahhoz, hogy meg is értse, pontosan hogyan képzeljük el az együttműködést.

Sőt az is előfordulhat, hogy különböző kollégák különböző szemszögből látják a folyamatokat, így elmondásaikból meglehetősen nehéz összerakni egy egésszé, pontosan mi is a globális elképzelés.

A másik problémás kérdés ebben az esetben, ha valaki másik csapatba kerül a cégen belül. Ennek számos oka lehet, az átszervezéstől kezdve az előléptetésig, illetve a projektszervezésű vállalatoknál mindez igen gyakori. Ha ugyanis folyamatosan ugyanabban a környezetben, ugyanazzal a társasággal dolgozunk, akarva-akaratlanul is kialakul egyfajta munkamenet, a csapat idővel összeszokik. Ezeket a megszokásokat azonban nem könnyű levetkőzni, így ha az ember új környezetbe kerül, ösztönösen próbálja eddigi tapasztalatait felhasználni. Az átállás sokkal egyszerűbb, ha konkrétan rögzítve van, mi az, amire át kell szokni, amit meg kell tanulni és elsajátítani. Ha csak azt éreztetjük az emberekkel, hogy amit csinálnak, az úgy nem jó, viszont nem tudjuk elmondani, megmutatni, mi az, amit konkrétan elvárunk, könnyen ellenállásba és érdektelenségbe ütközhetünk.

A koncepció nélküli vállalat természetesen lehet működőképes, azonban versenyképesnek maradnia nem egyszerű. A piaci helyzet ugyanis megkívánja, hogy a határidőket, költségeket és egyéb korlátokat tartsuk. Ha nem alkalmazunk semmiféle koncepciót, már ezek becslése is nehéz, nem is

beszélve a betartásukról. Előbb-utóbb érezni fogjuk a kísértést: valami hiányzik...

Egy cégnél a következő helyzettel találkoztam: a kitűzött cél egy nyilvántartó rendszer kifejlesztése volt egy olyan platformon, amelyen korábban nem dolgoztak. A megrendelő a cégvezetés volt, bízva abban, hogy az elkészülő rendszerre sikerül majd vevőket találni. A fejlesztéshez vettek fel új embereket is. A szoftverről azonban csak egy ködös elképzelés állt rendelkezésre, és a technológia is új volt, így senki nem tudta megbecsülni, mennyi idő alatt és mekkora költségvetéssel készülhet el. A specifikációk folyton változtak, hiszen nem szorított a határidő. Egy idő után már mindenki kezdett belefásulni, hiszen mint egy rágógumi, nyúlt a dolog, ám kézzelfogható eredmény nem volt.

A rendszer mind a mai napig nem készült el. Fejlődése során szervezeti és technológiai átalakításokat egyaránt megélt, azonban a folyamat vége nem látszik. Érthető módon a vezetőség türelme egyre fogy, a fejlesztők pedig – bár szeretik munkájukat – egyre kevésbé motiváltak.

Ha nincsenek határidők, nehéz tartani azokat. Ha nincsenek költségkorlátok, nagyon könnyen abba a hibába esünk, hogy minden felmerülő ötletet igyekszünk megvalósítani. Így azonban a rendszer még később lesz kész – és a kör bezárult. Ha viszont nem tudjuk, mikor leszünk készen, mekkora befektetéssel, és mit fog tudni pontosan a rendszer, eladni sem lesz egyszerű azt.

A koncepció nélküli vállalat természetesen lehet működőképes, azonban versenyképesnek maradnia nem egyszerű.

Sokkal célszerűbb azt mondani: rendben, az első verzió ezt és ezt a funkciócsoportot tudja majd, a menet közben felmerülő ötletek később kerülnek megvalósításra. Ezzel elkerülhetjük a „rétestészta-effektust", és az apró részekre bontott, konkrét célokat kitűző rendszer megvalósítása is egyszerűbb, átláthatóbb, ezáltal az emberek motivációja is növelhető. Ha látják, mit kell elérni, és sikerélmények tarkítják az utat, sokkal lelkesebben és odaadóbban végzik majd munkájukat.

Ehhez azonban fel kell ismerni a változás szükségességét, és határozott lépéseket kell tennünk azok megvalósításáért.

Jól bevált módszertan alkalmazása

Ha eljutottunk abba a fázisba, hogy látjuk: reformokra van szükség ahhoz, hogy vállalatunk hatékonyan és eredményesen működhessen tovább, az elsőként kínálkozó megoldás: vezessünk be olyan ajánlásokat, amelyek máshol már bizonyítottak.

Az ötlet alapvetően nem rossz, azonban van néhány dolog, amire érdemes odafigyelni ebben az esetben is. Először is: mindenképp figyelembe kell vennünk, hogy cégünk, projektünk és embereink mind-mind saját egyéniséggel rendelkeznek. Mint ahogy nincs két egyforma ember, ugyanúgy különböznek a vállalatok, a csapatok és a folyamatok is. Ráadásul nem szabad megfeledkeznünk a különféle külső hatásoktól, amelyek előre soha nem kiszámíthatók.

Így ha arra a döntésre jutunk, hogy mások által kidolgozott módszertant kívánunk bevezetni, vegyük figyelembe azon sajátosságokat, amelyek eltérőek lehetnek.

Először is, nem biztos, hogy az adott módszertan valamennyi aspektusát alkalmaznunk kell. Ha a projekt mérete, keretei vagy a csapat összetétele úgy kívánja, bátran hagyjuk el azon részleteket, amelyeket feleslegesnek ítélünk. Például a metodikák nagy része jelentős dokumentációigénnyel rendelkezik, ezek többsége azonban szükségtelennek bizonyulhat rövid terjedelmű, kis projektek esetén. Természetesen nem az a cél, hogy semmit ne dokumentáljunk! Sőt! Ne arra törekedjünk, hogy minél kevesebb adminisztrációval járjon a munkánk, hanem a minél teljesebb megoldásra, amely azonban mentes a felesleges szócsépléstől.

Másik kritikus tényező a különböző megbeszélések gyakorisága és rendje. A projektvezetők gyakran abba a hibába esnek, hogy túl gyakran hívják össze az embereket, akik így a kommunikációs lehetőség helyett munkájuk akadályoztatását érzik elsősorban. Fokozottan jelentkezik ez, ha nemcsak azokat hívjuk meg, akik feltétlenül szükségesek az adott témával, döntéssel kapcsolatban, hanem sok olyan személyt is, akinek a munkája meglehetősen távol áll a területtől. Alapvetően arra kell figyelni, hogy a döntések előkészítésekor és meghozatalakor is csak azok legyenek jelen, akik kompetensek a témában. Például egy részletekbe menő technológiai megbeszélésre ugyanúgy felesleges odarendelni az üzleti oldalt, mint ahogy a fejlesztőket sem célszerű egy felső szintű üzleti megbeszélésre invitálni.

A megrendelő és a fejlesztők között mindig legyen egy olyan interfészember (vagy csoport), akinek feladata az egyeztetés, közvetítés. Ez a személy lehetőleg értsen a feladat szakmai és üzleti részéhez egyaránt, hiszen munkáját csak így végezheti hatékonyan.

A konkrét, külső módszertanokkal kapcsolatban két céget mutatnék be: mindkét helyen ugyanazt a módszertant alkalmazzák. Az első vállalat mereven ragaszkodik az előírásokhoz, mindenben a kiválasztott metodika lépéseit követi. Az így keletkező többletmunka azonban gyakran akkora terhelést jelent, hogy az embereknek alig jut idejük a munka érdemi részére. Így egyrészt a határidők mindig csúsznak, a költségkeret folyamatos bővítésre szorul, ráadásul az emberek nemcsak a feladatukat, hanem a módszertan által megkövetelt lépéseket sem tudják precízen végezni.

A másik vállalatnál nem ennyire merevek: ők úgy gondolják, ragaszkodnak az adott módszertanhoz, viszont ahol szükségesnek találják, testreszabják azt. Így egyrészt van egy konkrét, kialakult, ledokumentált rendszer, amely alapján munkájukat végzik, másrészt ezt kellően rugalmasan kezelik ahhoz, hogy ne akadályozza, hanem segítse őket. A cég ily módon képes a határidők betartására, a költségek kézben tartására, és a megvalósítandó funkciók megfelelő kezelésére. Véleményem szerint azonban ennek a megoldásnak is lehetnek buktatói. Egy előre „gyártott", kész módszertan mindig úgy születik, hogy az adott cég ledokumentálja saját folyamatait, majd ebből különböző általánosítások alapján egy olyan metodikát publikál, amely kellően átfogó és általános ahhoz, hogy mindenkinek nyújthasson praktikus ötleteket. Ugyanakkor ennek az összefoglaló, mindenre

kiterjedő jellegnek köszönhetően helyenként túl sokat is követelhet, és túl keveset is adhat, a konkrét projekt természetétől függően. Ha pedig mindezt testre akarjuk szabni, felmerül a kérdés: vállalati vagy projekt szinten kívánjuk mindezt? Ha felső, vállalati egységes kialakításban gondolkodunk, az a probléma, hogy nemcsak a cégek, hanem a projektek is különbözőek, így elképzelhető, hogy ugyanazon szervezeten belül más vezetése alá tartozó, más projektek más módszertant igényelnek. Ekkor pedig nem sokat értünk a testreszabással.
Nyitott maradt a kérdés: mi a helyzet azzal a szituációval, amikor ragaszkodunk valamely módszertanhoz, azonban azt minden alkalommal, minden új projektre testreszabjuk.
A vállalatok többségénél egyszerre több fejlesztés is folyik. Ha ezek mindegyikénél egyedileg alakítjuk ki az alkalmazott stratégiát, egy idő után követhetetlenné válhat, hol hogyan dolgozunk, és ki milyen szempontok alapján alakítja ki a projektkörnyezetet. Így a cégen belül is kaotikus, áttekinthetetlen lehet a helyzet, és az emberek átmozgatása egyik projektből a másikba igencsak nehézkessé válhat.
Természetesen mindez megfelelő adminisztrációval, többletmunkával elkerülhető, azonban adódik egy harmadik, ígéretes, ám nem egyszerű megoldás is.

Saját módszertan kidolgozása

A „nagyok" módszertanaiból rengeteget lehet tanulni, azonban mint láthattuk, alkalmazásuk nem mindig egyszerű. A levont tanulságok, megszerzett ismeretek jól alkalmazhatók egy flexibilis környezetben, ahol nem a módszertanhoz való merev ragaszkodás a cél, hanem a hatékony, eredményes munka.
Ha saját, testreszabott metodikát szeretnénk kidolgozni, érdemes odafigyelni néhány problémára: az első és legfontosabb kritérium, hogy ne essünk ugyanabba a hibába, mint a másoktól átvett szabályozások alapján, vagyis új módszertanunk ne legyen túl merev, könnyedén el lehessen hagyni egyes részeit, vagy szükség szerint kiegészíthessük újabb komponensekkel. Ez azonban ismét azt a kérdést veti fel, hogy hogyan kerülhetjük el a követhetetlen, mindig másképp levezényelt projekteket, hiszen a módosítás szabadsága rengeteg félreértéshez vezethet.
A megoldást az jelentheti, ha nemcsak a konkrét, vállalati szintű módszertant határozzuk meg, hanem arra is adunk ajánlást, hogy ezt hogyan, milyen irányban lehet módosítani. Ha azt is ledokumentáljuk, melyik projekttípushoz mit javaslunk, a szabadságból semmit nem veszünk el, azonban cégünket mégis az egységesítés felé tereljük, és ezáltal mind a minőségbiztosítási, mind a szervezeti kritériumoknak megfelelhetünk.

Hogyan álljunk tehát neki a munkának? Előre leszögezem: a feladat nem egyszerű.
Először is fel kell mérnünk, mik azok a dolgok, amelyek jelenleg jól működnek. Ha a cégünk még életben van, akkor jó eséllyel találunk ilyeneket, bármilyen elkeserítőnek látjuk is a helyzetet. Járjunk annak is a végére, hogy hogyan, mitől működnek ezek a dolgok, s vegyünk számításba mindent: lehet, hogy egy-egy rátermett ember a kulcstényező. Lehet, hogy többé-kevésbé tudatosan módszeresen dolgoznak az emberek, és persze az is előfordulhat, hogy a siker csupán a véletlen műve. Ez az esély azonban hosszú távon egyre inkább a nullához közelít…
Természetesen mindezek felmérése során teljesen objektívnek kell lennünk, hiszen e nélkül szinte az egész erőfeszítés felesleges. Hasznos lehet külső szakértő bevonása is, aki hozzáértő módon, független szempontból vizsgálja át cégünk működését.
Ha már úgy érezzük, hogy mindent tudunk a vállalatról, illetve csoportjainkról, elkezdhetjük összeszedni azokat a dolgokat, amelyek már most is működnek, és egybevágnak céljainkkal, illetve azokat is,a melyek szöges ellentétben állnak azokkal.
Tegyük fel például, hogy ügyfeleink elégedettségén szeretnénk javítani, ám korábban egyetlen projektben sem vontuk be őket a tervezésbe. A probléma leírása után megszakítottuk velük a kapcsolatot, és legközelebb az átadásnál találkoztunk. Érezhető, hogy a megoldás első lépése rögtön az, hogy ezen a hozzáálláson változtassunk. Képzeljük el, mit éreznénk, ha a házunkat úgy építenék fel, hogy mi (jó esetben) látjuk a terveket, majd legközelebb a kulcsátadásra engednek oda bennünket. Ugyanígy a megrendelőket, ügyfeleket sem csak az előkészítéskor, hanem a projekt teljes életciklusa során be kell vonni a munkába, hogy az állandó kapcsolat eredményeként szoros és hatékony együttműködés jöhessen létre.
Ha tehát már látjuk, mit csinálunk jelenleg jól, és mi az, amin változtatni kell, összeáll egy átfogó kép, amelyet összevethetünk különböző, korábban megismert módszertanokkal.: melyik az, amely leginkább megfelel a jelenlegi állapotnak, melyik vezet leggyorsabban céljainkhoz? Melyik az, amelynek bevezetése a legkevesebb „fájdalommal" járna?
És itt jön a lényeg: ebben a lépésben nem szükséges egyetlen módszertan mellett elkötelezni magunkat. Sőt! Úgy gondolom, ha különböző forrásokból összegyűjtjük a lehetőségeket, és azokból egy saját, testreszabott, jól paraméterezhető metodikát dolgozunk ki, az lehet a leghatékonyabb. Így akár arra is tekintettel lehetünk, hogy hogyan tudjuk fokozatosan, lépésenként „forradalmasítani" munkánkat, melyek azok a fázisok, amelyeket akár azonnal meg tudunk valósítani, és melyek azok, amelyek alaposabb előkészítést igényelnek.

Az újítások bevezetése

Természetesen nem elég, ha mindez a fejünkben összeáll, vagy papíron leírjuk és közzétesszük. Cégünknél be is kell vezetni az újításokat, hogy azok érjenek valamit. Egyik napról a másikra azonban nem lehet megváltani a vállalatot. Hiába vagyunk jó vezetők, ha minden előzmény nélkül kiállunk dolgozóink elé, és azt mondjuk: mától ezt és ezt a módszertant fogjuk alkalmazni – semmit nem érünk el, csupán kemény ellenállásba ütközünk. Fokozatosan, lépésről lépésre sokkal messzebbre juthatunk. Ráadásul ha a bevezetendő módszertanunkat jól dolgoztuk ki, és bevezetését jól készítettük elő (belső marketing, a dolgozók folyamatos tájékoztatása, stb.), senki nem fogja erőltetettnek vagy feleslegesnek érezni azt.
Így pedig határozottan könnyebb, kellemesebb és kényelmesebb, sőt hatékonyabb lesz a munka. Kell ennél több?

Molnár Ágnes
rendszertervező
molnar.agnes@cleware.hu

Üzletiintelligencia-szoftverek

MS SQL 2000 Reporting Services II. rész

Az üzleti intelligencia minden vállalat életében nélkülözhetetlen, segítségével megalapozott döntéseket hozhatunk, amelyek sikeressé tesznek minket a piacon. Az üzleti intelligencia nem feltétlenül igényel egy teljes OLAP-ot vagy adatbányászatot, mint amilyenek az SQL Server Analysis Services-re épülő megoldások, de minden üzletiintelligencia-megoldásnak van egy közös eleme: a jelentések (report-ok).

Jelentéskészítés VS.NET Report Designer-rel

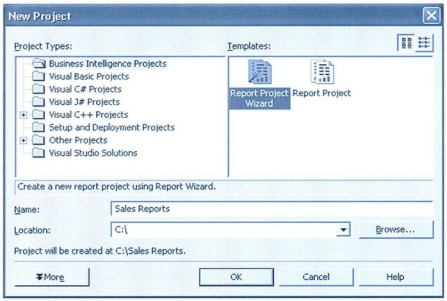
A korábbi részben bemutatott kimutatáskészítés életciklus-diagramja alapján el fogunk készíteni egy jelentést, majd publikáljuk a *Report Server*-be. Először is el kell készítenünk magát a kimutatást. Erre két lehetőségünk van. A *Report Definition Language (RDL)* ismeretében saját magunk összerakhatjuk azt az XML fájlt, ami teljes egészében leírja a kimutatás vizuális komponenseit, vezérlőit, a paramétereit, az adatforrást, a kimeneti formátumot és még sok mást. A *Reporting Services Books Online*-ban *(RSBOL)* megtalálhatóak az adott XML állományban használható elemek **[1]**, segítségünkre lehet még egy XML diagram is **[2]**, amely bemutatja a főbb elemek sémabeli elhelyezkedését, valamint az elkészült RDL állományunk ellenőrzésére felhasználható egy XSD dokumentum **[3]**. Ez a módszer jó lehet, ha egy olyan alkalmazást kell készítenünk, amelynek a feladata magának a kimutatásnak az összeállítása dinamikusan, gyakran az ügyfél gépén. Amennyiben előre elkészített jelentéseket akarunk gyorsan elkészíteni, segítségünkre siet a Visual Studio 2003 *(VS.NET)* egy új projekttípussal.

A kimutatás varázsló

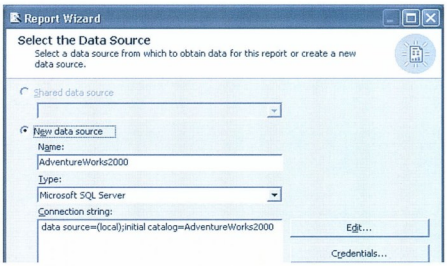
Indítsuk el a VS-t, és válasszuk ki a *Business Intelligence Projects*-ek közül a *Report Project Wizard*-ot, majd nevezzük el a projectet például *Sales Reports*-nak *(1. ábra)*.

Majd a varázsló üdvözlő képernyőjét átlépve állítsunk be egy adatforrást a *Reporting Services*-zel együtt települt *AdventureWorks2000* példa adatbázisára *(2. ábra)*.

Ezt az adatbázis kapcsolatot megosztottá is tudnánk tenni, ezáltal a kapcsolat nem a kimutatásban tárolódna el, hanem a projektünkben, ami egy különálló állományt alkotna a jelentés publikálásakor. Akkor van igazán előnye ennek, ha a kimutatásprojektünk sok-sok kimutatásból áll, de ugyanazon adatforrásból táplálkozik, ilyenkor központilag tudjuk a kapcsolatot megváltoztatni. A következő lapon a lekérdezést tervezhetjük meg, amelyhez jár az *Access*-ben, az



1. ábra **Új projekt készítése**

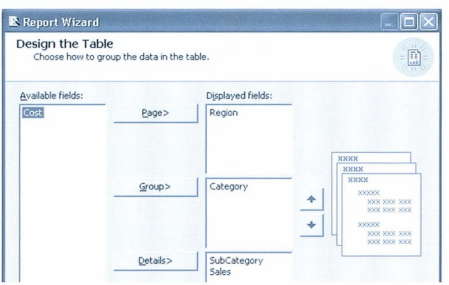


2. ábra **Adatforrás kiválasztása**

SQL Server Enterprise Manager-ben már jól megszokott *Query Builder*. Adjuk meg a lejjebb látható lekérdezést,

```
SELECT st..[Name] AS Region, pc..[Name] AS
Category,
      psc..[Name] AS SubCategory,
      SUM(p.ListPrice * sod.OrderQty) AS Sales,
      SUM(p.StandardCost * sod.OrderQty) AS Cost
FROM SalesOrderHeader soh
INNER JOIN SalesOrderDetail sod
      ON soh.SalesOrderID = sod.SalesOrderID
INNER JOIN Product p
      ON sod.ProductID = p.ProductID
INNER JOIN ProductSubCategory psc
      ON p.ProductSubCategoryID =
      psc.ProductSubCategoryID
INNER JOIN ProductCategory pc
      ON pc.ProductCategoryID =
      psc.ProductCategoryID
INNER JOIN SalesPerson sp
      ON sp.SalesPersonID = soh.SalesPersonID
INNER JOIN SalesTerritory st
      ON sp.TerritoryID = st.TerritoryID
INNER JOIN Customer c
      ON soh.CustomerID = c.CustomerID
WHERE st.[Group] = 'North America' AND
      c.CustomerType = 'S'
GROUP BY st.[Name], pc.[Name], psc.[Name]
```

amely leválogatja az *AdventureWorks2000* példa adatbázisból a dél-amerikai csoporthoz tartozó értékesítési területek kiadásait, bevételeit területenkénti, kategóriánkénti, alkategóriánkénti bontásban. A *Next* gomb után válasszuk ki a táblázatos *(tabular)* kimutatástípust. Az ezt követő táblázattervező lapon pedig állítsuk be a kimutatás csoportosítását.



3. ábra **A táblázat megtervezése**

A régiók laponként legyenek csoportosítva, ezen belül minden lapon legyen kategóriánként csoportosítva, a csoport részletezésében pedig az alkategóriánkénti eladások szerepeljenek (3. ábra). A további lapon válasszuk ki a táblázatelrendezésekből a lépcsőzetes *(stepped)* elrendezést a részösszegek megjelenítésével *(include subtotals)*. Majd az újabb lapon beállíthatjuk a táblázatunk stílusát, hasonlóan egy Excel-táblázathoz, csak lényegesen kevesebb előre elkészített stílussal. Legyen például *Corporate* a stílusa. Az utolsó előtti lapon beállíthatjuk a majdani *Report Server* elérési útvonalát és az azon belüli telepítési könyvtárat. A végső oldalon pedig elnevezhetjük a kimutatásunkat, legyen például *Sales by Region*. A *Finish* gombra kattintva el is készül a kimutatásunk, amelyet design módban meg is tekinthetünk a *Layout* fül alatt. Ha átváltunk a *Preview* fülre, egy röpke fordítás után meg is jelenik a kimutatás futásidőben, láthatóvá válik a 6 oldalon a 6 különböző régió. Ez a mód csak egy lokálisan futtatott mód, nem tárolódik ekkor még a jelentés a *Report Server*-ben, viszont alkalmas arra, hogy a jelentésünk működését teszteljük. Ugyanezen a *designer*-lapon található a Data fül, ahol a varázslóban már megjelent *Query Builder*-t találjuk meg, lehetőséget adva a kimutatás alapjául szolgáló lekérdezés módosítására. Ha elkészültünk, nincs más hátra, mint publikálni a jelentésprojektünket a *Report Server*-be, amelyhez a VS.NET támogatást ad. A *Solution Explorer*-ben a projekt tulajdonságaiban módosítható a *Report Server* elérése *(TargetServer URL)*, illetve a telepítési könyvtár *(TargetFolder)*, amennyiben a varázslóban elrontottuk. Majd a *Build* → *Deploy Solution* menüponttal publikálhatjuk az elkészült kimutatásunkat, így elérhetővé válik a hálózat minden egyes kliense számára webböngésző használatával a következő

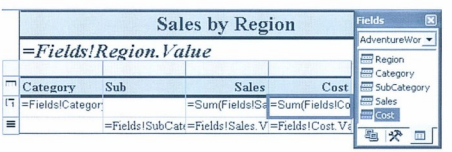
```
http://localhost/Reports/Pages/Folder.aspx
```

URL-ről elindulva. Mivel a projektünk neve Sales Reports volt, ezért a végleges útvonal, ahonnan elérhetjük:

```
http://localhost/Reports/Pages/Report.aspx?
ItemPath=/Sales Reports/Sales by Region
```

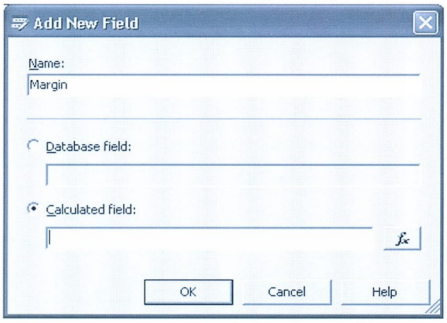
Az elkészült kimutatás módosítása

Adjunk egy új oszlopot a már meglévő táblázatunkhoz. Gondolom, mindenkinek feltűnt, hogy a SELECT listában szereplő *Cost* alias nevű oszlopot nem használtuk fel. Kattintsunk a táblázatunk tetszőleges cellájára, majd a szerkesztési módba került táblázaton a Sales oszlop fejlécre jobb egérrel kattintva válasszuk az *Insert Column to the Right* menüt. Ekkor kiegészül a 3 oszlop egy újabb negyedikkel. Ezután a bal oldali eszköztárban lévő *Fields* ablakból húzzuk rá a negyedik oszlop legalsó cellájába *(Detail)* a *Cost* mezőt. Látható, hogy az oszlop fejléce felveszi a mező nevét. Majd ugyanezt a műveletet ismételjük meg, de most a középső üres *Cost* cellába dobjuk bele a *Fields* ablakból a *Cost* mezőt.

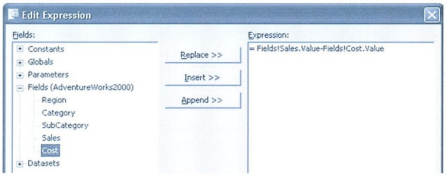


4. ábra **Új oszlop hozzáadása**

Mivel ez egy aggregált oszlop, a *Designer* automatikusan kiegészíti a *Sum()* függvénnyel. Így lesz az értéket képviselő kifejezés: *=Sum(Fields!Cost.Value)*. Az egyenlőségjel az *Excel*-ből már ismerős lehet, az oszlopra pedig a *Fields!* jelöléssel hivatkozhatunk. Adjunk a táblázatunkhoz egy számított oszlopot is, amelyben megjelenítjük a bevétel és a kiadás közötti árrést. Ehhez a *Fields* ablakban jobb egérrel válasszuk az *Add* menüt. Nevezzük el például *Margin*-nak (árrés) az oszlopot, és kiválasztva a *Calculated field*-et adjuk meg a kifejezést (segít az fx gomb).

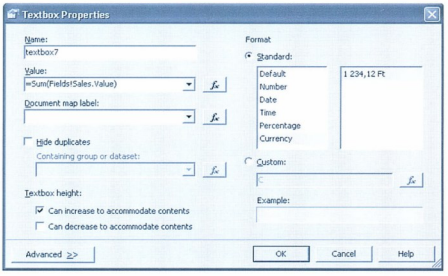


5. ábra **Számított oszlop hozzáadása**



6. ábra **Általános kifejezés szerkesztő ablak**

Majd az előző *Cost* oszlophoz hasonlóan eljárva egészítsük ki egy ötödik oszloppal a táblázatunkat, és a két üres cellára húzzuk rá a számolt *Margin* oszlopunkat. Ezeket a pénznemeket illik megformázni, hogy az aktuális pénznemben íródjanak ki az értékek. Egyszerre több cellát is kijelölve (a Shift nyomva tartása mellett), majd az F4-gyel megjelenítve a tulajdonság ablakot a *Format*-hoz adjunk meg egy nagy C-t *(Currency)*, a *Language*-hez pedig a *Hungarian*-t. A másik módszer, hogy egyenként jobb egérrel kiválasztjuk a tulajdonság lapot,



7. ábra **A táblázat egy szövegmezőjének formázása**

illetve az *Advanced* gombra kattintva a *Format* fülön is sok egyéb beállításra nyílik lehetőség. Ha rákattintunk a táblázat bármelyik cellájára, úgynevezett sor- és oszlopkiválasztók jelennek meg a sor elején, illetve az oszlopok tetején. A *Formatting* eszköztár segítségével színezhetjük, formázhatjuk az egyes sorokat, oszlopokat, vagy akár cellákat is.

Például a kategóriánkénti összesítő sor hátterét (a *Detail* feletti sor), színezzük ki szürkére *(LightGray)*.

Paraméterezés

Beszéltünk már róla, hogy a *Reporting Services* másik nagy előnye, hogy a kimutatások interaktivitása a jelentésfájlokban lévő paraméterezési lehetőséggel van biztosítva. Azaz a jelentés együtt él, egységbe van zárva a jelentést befolyásoló paraméterekkel. Így a felhasználónak, amennyiben egy paraméterezett lekérdezést akar elindítani, előbb be kell állítania a szükséges paraméter értékeit (a lekérdezési dátum intervallumai, régiószűrések stb. értékei). Egy valós eset, hogy a példánkban használt adatbázisból az egyes területek értékesítési menedzserei más és más kimutatást szeretnének kinyerni, pontosabban őket csak a saját eladási adataik érdeklik majd. Azaz a fenti *SELECT* kifejezésbe fixen beépített *(Group = 'North America')* szűrést ki kell cserélni egy kívülről érkező és beállított paraméterre. Ehhez a *designer*-ben a *Data* nézetben meg kell keresni a lekérdezésünket leíró rácsban az = *'North America'* kifejezést, ezt a *Criteria* oszlopban találjuk, és le kell cserélni az = *@Territory* kifejezésre.

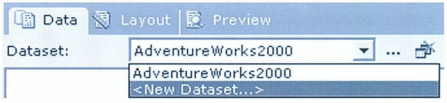
Column	Alias	Table	Output	Sort Type	Sort Order	Group By	Criteria
Name	SubCategor	psc	✓			Group By	
p.ListPrice * sor	Sales		✓			Sum	
p.StandardCost	Cost		✓			Sum	
[Group]		st				Where	= @Territory
CustomerType		c				Where	= 'S'

8. ábra **Paraméter megadása a lekérdezésben**

A *Report Designer* automatikusan észleli a paraméterünket, és felveszi a paraméterkollekcióba. Ha megnézzük a *Preview* nézetet, azt látjuk, hogy megjelenik egy *Territory* paramétert bekérő szövegdoboz, de nekünk kell kézzel beírni az értéket, méghozzá pontosan, különben a lekérdezés nem eredményez egy sort sem.

9. ábra **A paraméter első megjelenése a kimutatásban**

Ehelyett szebb lenne, ha feltöltődne egy legördülő lista a lehetséges értékekkel. Mi sem egyszerűbb! Egy kimutatásállományban több lekérdezés is szerepelhet, így elkészíthetünk egy olyat, ami visszaadja az összes területet az adatbázisból. A Data nézetben adjunk a meglévő DataSetünk mellé egy újabbat.

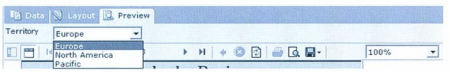


10. ábra **Több DataSet létrehozása egy jelentésben**

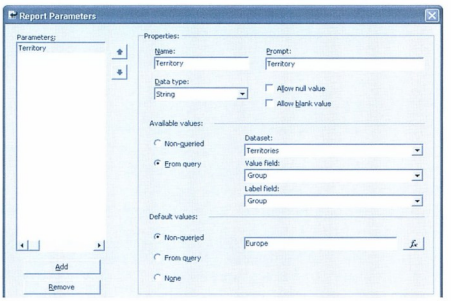
Adjuk meg a *DataSet* nevét (pl. *Territories*), a *DataSource* még mindig a kezdetben beállított *AdventureWorks2000*, a *Command type* pedig *Text*, majd írjuk be a következő lekérdezést, amelyet persze a *Query Builder* segítségével is össze lehetett volna rakni.

```
SELECT [Group] FROM
SalesTerritory
GROUP BY [Group]
```

A végeredményt ki is próbálhatjuk a felkiáltójel (!) megnyomásával. Amit kapunk: *Europe, North America* és *Pacific*. Ezek után a *Layout* nézetben nyissuk meg a *Report → Report Parameters* menüt. Majd a korábban létrehozott *Territory* paraméterhez rendeljük hozzá a most elkészített *DataSet*-et *(Territories)*. A *Value field* kerül behelyettesítésre a lekérdezésbe, míg a *Label field* lesz a megjelenített mező a listában. Ne engedjük meg a *NULL* értéket, és a *blank* üres sztring értéket se. Állítsunk be alapértelmezett *(Non queried)* értéknek *Europe*-ot..



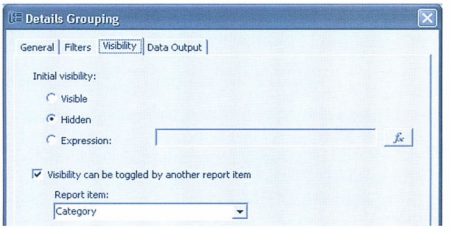
11. ábra **Az elkészült legördülő lista**



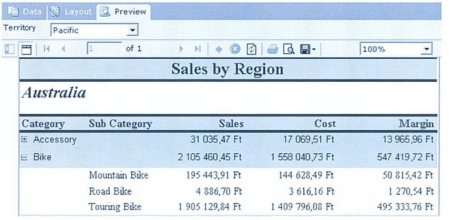
12. ábra **DataSet hozzárendelése a paraméterhez**

További interaktív jelentések

A fentiekben vázolt paraméteres kimutatások az interaktív jelentések egyik csoportját alkotják. Egy másik csoportot alkotnak az úgynevezett *drill-down*, valamint a kimutatás egyes részeinek elrejtését használó jelentések. Ha azt szeretnénk elérni, hogy például az egyes kategóriák alatti részletek rejtve legyenek mindaddig, amíg le nem nyitja a felhasználó egy-egy kategória esetén, akkor nem kell mást tenni, mint a *Layout* nézetben a táblázatban kijelölni a legalsó szintű *Detail* sort. Majd F4-gyel megnyitni a tulajdonság ablakot, és a *Visibility* csoportban beállítani a *Hidden* tulajdonságot *True*-ra (alapértelmezetten a csoportok össze lesznek csukva), valamint a *ToggleItem* tulajdonságot arra az értékre, amely az adott kibontandó táblázat cellát képviseli, jelen esetben a *Category* oszlop, itt lesznek majd láthatóak a +/- jelek. Ugyanezt megtehetjük úgy is, hogy a kijelölt soron jobb egérrel az *Edit Group* menüt választjuk ki, majd a *Visibility* fülnél állítjuk be az értékeket (13. ábra).

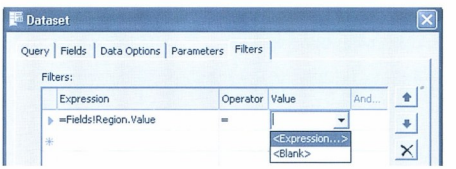


13. ábra **A láthatóság beállítása az „Edit Group"-on keresztül**



14. ábra **A drill-down kimutatás**

Egy másik lehetőség az interaktív kimutatásokra az adatforráshoz történő *„Filter"* hozzáadása. Ez akkor lehet hasznos, ha az a cél, hogy a teljes lekérdezés végeredménye leválogatódjon először, és csak azután történjen a szűrés valamilyen kritérium alapján, vagy olyankor, amikor az adott adatforrás nem támogatja a @-os paramétereket a lekérdezésben. A *„Filter"* megadásához úgyszintén a *Report Designer*-t hívjuk segítségül. A *Data* nézetben a kiválasztott *DataSet* listája mellett található gombot megnyomva, majd a *Filter* fület kiválasztva adhatjuk meg a szűrési feltételt.



További interaktivitást szolgálnak a kimutatásban elhelyezett *hyperlink*-ek és a dokumentumtérkép *(document map)*. A *hyperlink*-et *Layout* nézetben az adott táblázat egy szövegmezőjének kiválasztása után a jobb egérgombbal ki kell választani a tulajdonságok lapot, majd megnyomni az Advanced gombot. Itt a *Navigation* fülön tudjuk beállítani a szükséges *hyperlink*-et, könyvjelzőt stb.
A dokumentumtérkép beállítási lehetőségét a 7. ábrán találjuk meg.

A varázsló nem minden

Meg kell említeni még, hogy ugyan az elején a varázsló használatát vettük igénybe a gyorsabb haladás érdekében, de ugyanezt egy üres kimutatásprojektből kiindulva is el lehetett volna készíteni, persze korántsem ilyen gyorsasággal és hatékonysággal.

Kimutatás elérése *URL*-en keresztül

Az elkészült és publikált kimutatások egyik programozott elérési módja, amikor az URL-ben adjuk meg a megjelenéssel, paraméterekkel kapcsolatos információt. Például a következő URL a korábban elkészített jelentésünket a Territory=Pacific paraméterrel hívja meg, és ezzel egyidejűleg le is tiltja a paraméter beállítási lehetőségét (nem jelenik meg egyáltalán a képernyőn), valamint tiltja a Toolbar megjelenését, és a nagyítást 150%-ra állítja.

```
http://localhost/ReportServer?/Sales Reports/
↳ Sales by Region&Territory=Pacific&
↳ rc:Parameters=false&rc:Toolbar=false&
↳ rc:Zoom=150
```

A részletesebb URL-szintaktika itt található meg: **[4]**. A szintaktika kiterjed a kimutatások keresésére, a kimeneti formátum megadására *(pl. azonnal PDF-be exportálhatunk)*, az eszköztár, a dokumentumtérkép engedélyezésére/tiltására. Ránavigálhatunk a dokumentum egy adott részére, beállíthatunk kezdeti nagyítási értéket, adott sorszámú lapot jeleníthetünk meg.

Kimutatás elérése webszolgáltatáson keresztül

A példák *C#* szintaktikával fognak megjelenni *(de Visual Basic-re is átültethető analóg módon)*. A webszolgáltatást és ezzel együtt a korábban elkészített kimutatásunkat egy standard *Windows Form*s alkalmazásból fogjuk elérni. Ahhoz, hogy alkalmazásunkban használhassuk a *Reporting Services* által nyújtott webszolgáltatást, a következők szükségesek:

- Létre kell hoznunk egy proxy osztályt a webszolgáltatáshoz
- Authentikáltatni kell a webszolgáltatás klienst a Report Server-rel
- Meg kell hívni az adott webmetódust, amely a szükséges műveletet végrehajtja

A *proxy* osztályra azért van szükség, mert a kliens és a webszolgáltatás SOAP-üzeneteken keresztül kommunikál egymással, úgy, hogy az üzenetet, beleértve a paramétereket, XML-ben küldi át. A *proxy* osztály a paramétereket XML elemekké alakítja, majd ezt a SOAP-üzenetet küldi át a hálózaton. A SOAP szintjét a proxy elfedi, így ugyanúgy tudunk meghívni egy webmetódust, mintha az egy általunk megírt metódus lenne az alkalmazásunkban. *Proxy* osztály generálására két lehetőség is adódik. Az egyik a WSDL eszköz használata. A .NET Framework SDK tartalmaz egy WSDL *(Web Services Description Language)* eszközt, amely képes egy ilyen proxy osztály generálására. Minimálisan meg kell adnunk a webszolgáltatásunk elérési útvonalát:

```
wsdl.exe /language:CS
↳ http://myserver/reportserver/reportservice.asmx
```

A fenti utasítás az aktuális könyvtárban készít nekünk egy *C#* forrásfájlt *ReportingService.cs* névvel (ez a szolgáltatás neve). Bővebb paraméterezésről itt olvashat: **[5]**. Ezek után a proxy osztályt le kell fordítani assembly *DLL*-lé, és betenni az alkalmazásunkba. A továbbiakban ezt az osztályt fogjuk referenciaként használni. A második módszer a *proxy* osztály létrehozására a VS.NET fejlesztőeszköz maga. A különbség, hogy elvégzi automatikusan a proxy osztály projektbe való integrálását, felveszi referenciába a *System.Web.Services.dll*-t, amit az első módszernél nekünk kell majd elvégezni. Ekkor nincs más teendőnk, mint a *Project◊Add Web Reference* menüpontot elindítva megadjuk a korábban ismertetett webszolgáltatást leíró állomány *(reportservice.asmx)* útvonalát. Majd a *GO* megnyomása után már meg is kaptuk azt a kis leírást, amely felsorolja a használható 93 metódust a paraméterezésekkel, amit használhatunk majd. Elnevezhetjük a referenciánkat, ami nem lesz más, mint a *proxy* osztályunk leendő névtere. Az *Add Reference* hatására elkészül a proxy osztályunk, és hozzáadóik a *System.Web.Services.dll* is.

```
using myNamespace.myReferenceName;
...
ReportingService rs = new ReportingService();
```

Ezek után authentikáltatni kell a kliensünket. Erre vagy Windows-authentikációt, vagy Basic-authentikációt használhatunk, az IIS Report Server virtuális könyvtárának biztonsági beállításától függően.

```
// Windows authentikáció esetén
rs.Credentials =
   System.Net.CredentialCache.DefaultCredentials;

// Basic authentikáció esetén
rs.Credentials = new
   System.Net.NetworkCredential("username",
      "password", "domain");
```

Ha el akarjuk kerülni a HTTP: 401-es *Access Denied* hibaüzenetet, akkor mielőtt bármilyen webmetódust meghívnánk, állítsuk be a *Credentials* tulajdonságot. Ha van rá mód, használjunk Windows-authentikációt inkább, ha nincs, futásidőben kérjük be, mellőzük a *Credential*-ök fájlban való tárolását, ha mégis szükséges, akkor a Win32 Crypto API-t használjuk hozzá. Innentől kezdve csak a műveletek elvégzéséhez szükséges webmetódusokat kell hívogatni. Ilyen például a *Render()* metódus, amelynek a segítségével feldolgozhatunk egy elkészített kimutatást, majd adott formátumban visszanyerhetjük, ezt egy *byte[]* tömbben adja vissza a metódusunk. Bővebben a *Render()* metódusról itt olvashatunk: **[6]**. Első lépésként elő kell állítanunk a metódusunk argumentumait, a kimutatás útvonalát, a megjelenítési formátumot (célszerűen HTML 4.0-t választottam, így a *drill-down* kimutatásunk interaktív lesz). A *DeviceInfo* **[7]** paraméterben megadhatunk egy XML állományt, amely formátumspecifikus megjelenítési beállításokat tartalmaz. Jelen esetben így tiltjuk le a *Toolbar* megjelenítését, így nem lehet a *Territory* paramétert kézzel beállítani. Hasonló elvet alkalmaztunk az URL-es eléréskor is.

```
byte[] result = null;
string reportPath =
                  "/Sales Reports/Sales by Region";
string format = "HTML4.0";
string historyID = null;
string devInfo = @"<DeviceInfo><Toolbar>False" +
                  @"</Toolbar></DeviceInfo>";
```

A következő lépés a paraméterek megadása, amely egy *ParameterValue* osztályból képzett tömbbel történik. Mivel egy paraméterünk van csak *(Territory)*, ezért egy egyelemű tömböt adunk át.

```
ParameterValue[] parameters=new ParameterValue[1];
parameters[0] = new ParameterValue();
parameters[0].Name = "Territory";
parameters[0].Value = "Pacific";
```

A továbbiakban megadhatnánk az adatforráshoz tartozó *Credential*-t, illetve egy sor jelenleg nem lényeges paramétert (a feldolgozás alatt keletkezett figyelmeztetések, kódolási beállítást, *MIME*-típust, a kibontható jelentésrészeket, a kimutatástörténethez tartozó paramétereket, a feldolgozási állapot információit).

```
DataSourceCredentials[] credentials = null;
string showHideToggle = null;
string encoding;
string mimeType;
Warning[] warnings = null;
ParameterValue[] reportHistoryParameters = null;
string[] streamIDs = null;
rs.SessionHeaderValue = new SessionHeader();
```

Ezután meghívjuk a *Render()* metódust a korábban beállított paraméterekkel, és visszakapjuk az eredményt egy *byte[]* tömbben.

```
try
{
   result = rs.Render(reportPath, format,
      historyID, devInfo, parameters, credentials,
      showHideToggle, out encoding, out mimeType,
      out reportHistoryParameters, out warnings,
      out streamIDs);
}
catch (SoapException eSoap)
{
   MessageBox.Show(eSoap.Detail.OuterXml);
}
```

A keletkezett eredményt egy *FileStream*-en keresztül a példa kedvéért a *C:\Report.htm* állományba írhatjuk.

```
try
{
   FileStream stream =
      File.Create(@"C:\Report.htm", result.Length);
   stream.Write(result, 0, result.Length);
   stream.Close();
}
catch (Exception eFile)
{
   MessageBox.Show(eFile.Message);
}
```

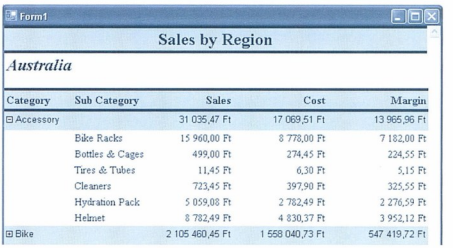
A végén pedig, csak a gyors és egyszerű teszt kedvéért, felrakhatunk a *Windows Forms* alkalmazásunk űrlapjára egy *COM*-os *WebBrowser* kontrollt, hogy megnézhessük a saját űrlapunkon az eredményt. Ehhez a vezérlőhöz elkészül egy *COM* csomagoló osztály *(axWebBrowser)*, amelynek a *Navigate()* metódusában megadjuk az elkészült HTML állomány elérési útvonalát (most fájlszinten).

```
object flags = Type.Missing;
object targetFrameName = Type.Missing;
object postData = Type.Missing;
object headers = Type.Missing;
axWebBrowser1.Navigate(@"C:\Report.htm",
   ref flags, ref targetFrameName, ref postData,
   ref headers);
```

Persze a fentieken túl bármit használhatunk a végső formátum megjelenítésére (a legkülönfélébb HTML-megjelenítő vezérlőktől az Excel munkafüzeteken át az Acrobat megjelenítőkig, széles a skála). A megfelelő működéshez a VS.NET *designer* által automatikusan importált névtereken felül a következőket kellett felvenni:

```
using AxSHDocVw; // WebBrowser kontrol csomagoló
using WindowsApplication1.RS; // webszolgáltatás
using System.Web.Services.Protocols;//SOAP kivétel
using System.IO; // FileStream műveletek
```

A *WindowsApplication1*-et és az RS-t (a webszolgáltatás referenciájának a neve) értelemszerűen a tesztprojektünknek megfelelően kell beállítani. A 15. ábrán jól látszik a *Windows Forms* projektünk végeredménye.



15. ábra **Saját Windows Forms alkalmazásunk kimutatásmegjelenítője webszolgáltatáson keresztül**

MÁTHÉ ZOLTÁN
mathez@bsi.hu
MCSD, SQL Server MVP

Felhasznált források:

Reporting Services Books Online
http://tinyurl.com/ys3e8

A cikkben szereplő URL-ek:

[1] http://tinyurl.com/2ftpn
[2] http://tinyurl.com/29h4u
[3] http://tinyurl.com/3a8fq
[4] http://tinyurl.com/3Bpj7
[5] http://tinyurl.com/3f3cn
[6] http://tinyurl.com/3f57x
[7] http://tinyurl.com/32rwu

Kiszolgálópark üzemeltetése 2. rész

A vas

A kiszolgálók telepítéséről, használatáról, hibajavításáról sok cikket olvastam, a kiszolgálók üzemeltetéséről szóló információkat ellenben meglehetősen nehéz összeszedni. Ez a cikksorozat ennek a hiánynak a pótlására született meg. Ebben a fejezetben azt elemzem, milyen legyen és mire jó a kiszolgálóhardver.
Egyáltalán: miért vegyünk kiszolgálóhardvert?

A hardver az amibe belerúghatsz. – szoktuk mondani. Nos, a cikksorozat mostani második része e rugdosódás célpontjával, a „vassal" foglalkozik. Szó esik majd arról, hogy milyen különböző elvárások vonatkoznak egy PC-re és egy kiszolgálóra. Milyen lépéseket tesznek a gyártók a kiszolgálóhardverek tervezésekor. Megvizsgáljuk, hogy mi határozza meg egy kiszolgáló teljesítményét. Szó esik a megbízhatóság növelésére tett lépésekről, valamint a gyártók különböző támogatási formáiról.

Minek vegyünk kiszolgálót?

Még ma is sok kisebb-nagyobb cégnél üzemelnek egyszerű PC-k kiszolgálói feladatkörben. Sem a cégvezetés, sőt sokszor az üzemeltetés sincs tisztában azzal, hogy milyen alapvető különbségek vannak egy végfelhasználói és egy kiszolgálói szerepre szánt PC között.
A leggyakoribb válasz erre az, hogy: *„Tulajdonképpen bármilyen PC-re telepíthetők a kiszolgálóalkalmazások, csak az install-CD-t kell betenni, és már megy is."* Ez a szemlélet egészen az első felbukkanó rejtélyes és váratlan problémáig tartható is lehet. Előbb-utóbb azonban váltani kell. A szemléletmódban és a hardverben is.
Miben különböznek a PC-vel és a kiszolgálóval szembeni elvárásaink? Az alábbi táblázatban a leggyakoribb használatbeli különbségeket foglalom össze (természetesen ezek általános használatra vonatkoznak, kivételek létezhetnek).

PC	Kiszolgáló
egy felhasználó használja	több felhasználó használja
a felhasználó a konzolon keresztül éri el	a felhasználók távolról (hálózaton keresztül) érik el
nem folyamatos üzemben használják	(365 x 24 órában folyamatosan üzemel
nem feltétlenül van hálózatra kötve	hálózatra van kötve
viszonylag csendes működés	viszonylag zajos működés

Természetesen e lista tetszőleges elemekkel tovább is bővíthető. Ám már a fenti kiragadott szempontok alapján is belátható, hogy a más jellegű használatra speciális hardvereszközök használatára lehet szükség.

Milyen szempontok alapján válasszunk hardvert?

Az egyik kiemelt szempont az ár.
Egy PC és egy kiszolgáló ára között a különbség akár háromszoros, nyolcszoros is lehet. Vajon megéri-e ezt a befektetést a kiszolgáló? Mi az a többletérték, amit a hardvergyártó ezért nyújt? Amit az ár vizsgálatakor célszerű figyelni, az a Total Cost of Ownership (TCO), azaz nemcsak a vételárat, hanem az eszköz élettartama során várható egyéb költségeket is magában foglaló „birtoklási költséget" is érdemes megvizsgálni.

Már van hol...

...de mivel is?

A következő szempont a teljesítmény.
Vajon tényleg tudjuk, hogy mit takarnak a különféle hangzatos számok? A teljesítmény komplex tulajdonság, több részegység együttműködése szükséges hozzá. Mérésére különféle tesztprogramok léteznek, melyek számos jellemzőt mérnek, ezek alapján lehet a különféle eszközöket összehasonlítani. A teljesítményt célszerű az egyes komponensek szintjén megvizsgálni, így az igényeknek jobban megfelelő eszközt tudjuk kiválasztani.
A megbízhatóság is kiemelt szempont kell(ene) hogy legyen egy kiszolgáló kiválasztásakor. Ezeket az eszközöket sokan használják, így az esetleges hardveres meghibásodások, illetve a bizonytalan működésből eredő szolgáltatászavarok kihatása igen nagy is lehet. Sok helyen a szolgáltatás kiesése konkrét anyagi veszteséget jelent, ezeken a helyeken könnyebben kimutatható, hogy mennyibe is kerül az állásidő.

A gyártói támogatás mint szempont.
Melyek és mire használhatók azok a támogatási formák, melyeket a hardvergyártó ad a különféle eszközökhöz.

Milyen szempontok alapján készítik fel a gyártók a kiszolgálóhardvereket a feladatkörök ellátására?

Tervezési szempontok

A kiszolgálóknál már a tervezőasztalon figyelembe veszik a kiszolgálókkal szemben támasztott követelményeket. Nemcsak szabványos alkatrészekből rakják össze a kész gépet, hanem erre a feladatra, környezetre tervezett hardverelemeket is felhasználnak. A különféle kiegészítők, perifériák illesztését is kipróbálják, ezeket egymáshoz is illesztik szükség szerint.

Teljesítmény (vagy másképp: performancia)

Lássuk a fő részegységeket és azoknak a teljesítményre gyakorolt hatásait:

Processzor

Sokan azt gondolják, hogy az egyik meghatározó mérőszám a sebesség. (Szerintük egy 1 GHz sebességű processzor nagyobb teljesítményű, mint egy 800 kHz-es. Ez nem feltétlenül van így!)

Egy processzor működését számos tényező határozza meg: a processzor működési elve, utasításkészlet, gyorsítási technológiák, hogyan működhet együtt több processzor.

Memória

Itt sem csak a méret a lényeg! Fontos még emellett a működési elv, az elérési idő, a sávszélesség, az I/O sávszélesség, a processzorsebesség.

Gyorsítótár (cache)

Típus, méret, sebesség, elérési idő a kulcsadatok.

Háttértárak

A háttértárak mérete, elérési sebessége, szervezése, technológiája az, amit figyelembe kell venni a kiválasztáskor.

Buszarchitektúra

A részegységek összekapcsolására különféle buszrendszerek léteznek. Ezek teljesítménye is alapjaiban meghatározza egy kiszolgáló teljesítményét. A busz sebességének növelése emeli a teljesítményt. Természetesen ekkor mind a processzornak, mind a busznak támogatnia kell a nagyobb sebességet.

Egy rendszer teljesítményének korlátja a leggyengébb részegységében kereshető, ez a szűk keresztmetszet az „üvegnyak". (Hiába van egy nagyon gyors processzorom, ha mellette nagyon lassú memória van, akkor a processzor akár idejének 90 százalékát is várakozással tölti, gyorsabb processzorral sem érhető el érezhető teljesítményjavulás.)

Tekintsük át, hogy a kiszolgálókat alkotó egyes részegységek miben térnek el a hagyományos PC-k elemeitől:

Processzor

Az Intel processzor esetén a sebességében nincs sok különbség a PC-kben és a kiszolgálókban található típusok között. A cache méretében azonban már jelentős eltéréseket találhatunk. Egy mai modern kiszolgálóprocesszorban 256–512 kb L2 és 1024–4096 kb L3 cache található.

Fontos tulajdonság a Hyper threading technológia használata. Ezzel a processzor virtuálisan több processzornak látszik az operációs rendszerben, ami körülbelül 30-40%-nyi teljesítménynövekedést hoz a konyhára, sőt például a Windows 2003 kiszolgálók ezt a technológiát operációs rendszer szintjén is támogatják, még optimálisabb kihasználást téve lehetővé. (Régebbi operációs rendszerek két fizikai processzort látnak, így osztják el a munkát közöttük, míg a Windows 2003 meg tudja különböztetni a virtuális és a fizikai processzorokat, így még jobban el tudja osztani feladatokat közöttük.)

Külön érdekességet jelentenek a Xeon DP és a Xeon MP processzorok, melyek speciálisan a többprocesszoros kiszolgálókba terveztek. Ezek segítségével valódi többprocesszoros kiszolgálókat építenek a gyártók, melyekkel további teljesítménynövekedést lehet elérni.

Memória

A memória általában az egyik tipikus szűk keresztmetszet szokott lenni. Ma már viszonylag nagy memóriák is elérhető áron megvehetők, így nem ritka a 2–4 GB memória kiépítettségű kiszolgáló sem. PC-ket is lehet ekkora memóriával megtölteni, ám ott többnyire a memóriakezelés, illetve a busz sebessége, szélessége hagy maga után kívánnivalót. A kiszolgálóknál ügyelnek arra, hogy a memória elérési sebessége összhangban álljon a processzor sebességével.

Háttértár

A PC-kben ma is többnyire az IDE/ATA felületű diszkek a legelterjedtebbek. Ez a technológia rengeteget fejlődött, olyannyira, hogy egyes, elsősorban kisebb teljesítményű kiszolgálókban is felhasználják ezeket. A kiszolgálókban általában SCSI, újabban a Serial ATA, illetve a réz alapú Fibre felületű HDD-k találhatóak meg. Ezek nagy teljesítményt nyújtanak, gyors elérési idővel rendelkeznek, az MTBF-értékük általában nagy.

Ezeket azután valamiféle hibatűrő szervezésben célszerű konfigurálni. A legelterjedtebb hibatűrő szervezési technológia a RAID, amit a legtöbb kiszolgálógyártó hardveresen is támogat, azaz a lemezvezérlő menedzseli a diszkeket, levéve a terhet az operációs rendszer és a CPU válláról, ezzel is növelve a rendszer teljesítményét, megbízhatóságát. (A firmware az operációs rendszer nélkül is képes például befejezni az adatok kiírását egy esetleges rendszerelszállás során.)

Buszarchitektúra

A kiszolgálókban sokféle buszrendszer megtalálható. Ezeknek a teljesítményre gyakorolt hatása talán a legfontosabb különbség a kiszolgáló és egy hagyományos PC között. A backside-busz a processzor és a cache közötti kapcsolatot valósítja meg, a frontside-busz pedig a processzort az egyéb részegységekkel köti össze. Egy jó kiszolgálóban ma már akár 533 MHz sebességű FSB is megtalálható. A memóriabusz nevéhez hűen a memória illesztését végzi. A perifériák az I/O buszrendszereken keresztül érhetőek el. A leggyakrabban támogatott I/O buszrendsze-

rek: ISA, EISA, PCI, PCI-X, PCI-X2.0, InfiniBand, PCI Express, AGP, USB, FireWire.
A nagy sebességű, illetve széles (64 bit, 66, 133 MHz) PCI-buszokat szinte kizárólag csak kiszolgálóhardverekbe építik be. A nagy sebességű PCI-X buszra 64 bites, 133 MHz órajelű eszköz is kerülhet, ami elméletileg akár 1067 MB/s átvitelre is képes, szemben a PC-kbe szerelt 32 bites, 33 MHz sebességű busz 133 MB/s elméleti maximum teljesítményével. Ide kerülhet például nagy sebességű háttértárvezérlő, amellyel sok diszk (28–36) is gyorsan elérhető. További, csak kiszolgálókra jellemző gyorsítási lehetőség a több PCI-vezérlő használata, így egy kiszolgálóba több kártyát helyezhetünk el. További előnye még ennek a megoldásnak, hogy a kártyáknak nem kell osztozkodniuk egy PCI-vezérlőn, ami szintén növeli a teljesítményt.

Ház, beépíthetőség

A folyamatos üzemre felkészített ház is sokat jelent. A benne elhelyezkedő részegységek hűtéséről is gondoskodni kell. Ezenkívül méret, elhelyezhetőség szerint többféle szempontot is figyelembe vesznek:

- Lehet rack-be szerelhető kivitelű, így könnyen, egyszerűen használható nagyvállalati környezetben.
- Lehet sok diszket befogadó ház, állománykiszolgálók befogadására.
- Lehet nagy hely a házban speciális, teljes hosszúságú kártyák befogadására.

Meg kell jegyezni, hogy általában valamely célra specializált házban lehet a kiszolgálót kapni, olyan univerzális ház, amely minden igényt egyszerre elégít ki, viszonylag ritkán fordul elő.

Megbízhatóság

Külön kiemelt fontosságú kérdés a kiszolgálók üzembiztossága. A kiszolgálókat 365 x 24 órás folyamatos üzemre tervezik, ezért különféle technológiákat alkalmaznak, úgymint:

Nagy megbízhatóság

A részegységekkel szemben magasabb minőségi igényeket támasztanak, mint egy átlagos PC-nél. A részegységeket nem csak külön-külön, hanem egymással összeépítve is tesztelik. Egyre több intelligens hardverfigyelő lehetőséget építenek be az eszközökbe, melyek üzem közben monitorozzák, tesztelik a rendszert, így képesek a hibák előrejelzésére is.

Redundancia

Az egyes kritikus elemeket duplikálják, így meghibásodás esetén sem áll le a kiszolgáló. Leggyakrabban a hűtőventilátorokat, tápegységeket, diszkeket kettőzik meg. A passzív eszközökből egyszerűen többet építenek be (ventilátor, tápegység), alkalmazhatók azonban intelligens megoldások is, például a ventilátorok sebességszabályozása a hűtési igénynek megfelelően stb.
A diszkek hardveres hibatűrő megoldásai is elterjedtek. Létezik egyszerű tükrözés vagy valamilyen magasabb szintű RAID-megoldás. Használnak még úgynevezett melegtartalék diszket is, ez a rendszerbe beépített +1 diszk, amely valamelyik másik diszk meghibásodásakor automatikusan átveszi annak a helyét.

A memóriamodulok meghibásodásának jelzésére és hibajavításra is többféle megoldást alkalmaznak. Legelterjedtebb a paritásgenerálás és -vizsgálat. Egy másik biztonsági tényező a melegtartalék. Itt a diszkekhez hasonlóan egy plusz memóriamodul van beépítve a rendszerbe, amely képes átvenni egy meghibásodott memóriamodul szerepét. A nagyvállalati kiszolgálókban létezik memóriatükrözés, illetve RAID-szervezésű memóriakezelés is, ezek a megoldások már igen jó hibatűréssel rendelkeznek.
A még biztonságosabb működés érdekében a hálózati adapterek is tükrözhetők, ilyenkor az egyik adapter (vagy hálózati szegmens, esetleg külön-külön aktív eszközre kötve az aktív eszköz) hibája esetén is elérhető marad a kiszolgáló. A diszkvezérlők is duplikálhatók, ez a megoldás még nagyobb rendelkezésre állást valósít meg.
Az alaplapi BIOS is megkettőzhető, ekkor nem csak a BIOS-hibákat, hanem a néha előforduló BIOS-flash-frissítési hibákat is kordában lehet tartani.

Gyors javíthatóság

Egyre nagyobb teret hódítanak az üzem közben cserélhető részegységek. Ezek, amint azt a nevük is mutatja, a kiszolgáló leállítása nélkül is cserélhetők (Hot Plug technológia), így még kevesebb leállásra van szükség az esetleges hardvermeghibásodások esetén. Egy ventilátor vagy egy tápegység cseréjekor a kiszolgálón futó operációs rendszer nem is feltétlenül szerez tudomást a javításról. Azonban például egy Hot Plug PCI-kártya cseréjéhez speciális eszközmeghajtók telepítése, illetve néhány szabály betartása is szükséges. A legújabb operációs rendszerek, mint például a Windows 2003, már tartalmaznak Hot Plug eszköztámogatást is. Egy nagyvállalati kiszolgálóban menet közben cserélhetők lehetnek a ventilátorok, tápegység, diszk, PCI-kártyák (pl. hálózati kártya, lemezvezérlő), memóriamodulok, sőt egyes modellekben maguk a processzorok is. Ilyen funkciókkal felszerelt kiszolgálók üzemeltetése esetén nagyon fontos a gyártó utasításainak betartása! A józan ész határain belül célszerű maradni, azaz nem célszerű például egyszerre az összes processzort vagy memóriamodult kivenni, akkor sem, ha melegen cserélhető…

Moduláris felépítés

Egy kiszolgálónál fontos szempont lehet a méretezhetőség. Költséghatékony lehet, ha csak azokat a részegységeket vesszük meg, amelyeket tényleg használni fogunk. Később viszont az igények változásával jól jöhet, ha a kiszolgálónk bővíthető, illetve egyes egységei cserélhetők, és nem kell az egészet újra váltani. Például egy dinamikusan fejlődő adatbázis-kiszolgálót kezdetben megvásárolhatunk két processzorral, majd később (ha vásárláskor gondoltunk erre) egyszerűen bővíthetjük négy processzorra, illetve növelhetjük a memóriáját is. Az állománykiszolgálónkhoz idővel újabb diszkeket köthetünk a felhasználói igényeknek megfelelően. A redundáns eszközöket is beszerezhetjük később, lehetőségeinkhez mérten (pl. redundáns tápegység, memória stb.)

Könnyű szerelhetőség és szervizelhetőség

A kiszolgálókat tapasztalatom szerint viszonylag ritkán kell szerelni, de akkor nagyon jól jönnek azok az ötletes megoldások, melyek a szervizmérnök életét könnyítik meg. Ezek a néha pofonegyszerű megoldások nemcsak a hibalehetőségeket csökkentik, hanem a szereléssel töltött időt (a potenciális állásidőt) is. Egy modern kiszolgálóhardver fizikai kialakítása átlátható, ábrákkal, színekkel jelölt lehet, az egyes részegységek könnyen, akár szerszámok nélkül is kivehetők, beépíthetők. Az újabb részegységek könnyen és helyesen beilleszthetők, a különféle kábelek és egyéb csatlakozók jól illeszthetők, megfelelő méretűek, könnyen és egyértelműen csatlakoztathatók, a csatlakozások könnyen bonthatók. Az eszköz hűtése, légáramlása megfelelő, jól átgondolt.

Biztonsági funkciók

Egyes kiszolgálók különféle biztonsági funkciókat is tartalmazhatnak:

- fizikai hozzáférés védelme: a ki-be kapcsolás, a ház nyitása, illetve a részegységek, diszkek kivétele ellen védhető általában valamilyen kulcsos megoldással,
- egyes speciális kártyákat, eszközöket speciális módon védhet, megakadályozva az adatok kinyerését (pl. PKI-hez kulcstároló kártyákat célszerű védeni).

Távoli menedzsment

Egyre több kiszolgáló tartalmaz beépítve valamilyen távmenedzselési szolgáltatást, melynek segítségével a távoli telephelyen-kiszolgálóhelyiségben elhelyezett kiszolgáló fizikai jelenlét nélkül is kezelhető. Ilyen lehet a konzol (billentyűzet, monitor, egér) távoli átvétele, floppy, CD távoli átvétele, különféle diagnosztikai funkciók futtatása-elindítása. (Létezik olyan, saját tápellátással rendelkező menedzsmenteszköz, mellyel a kikapcsolt, illetve meghibásodott kiszolgáló állapota is figyelemmel kísérhető, sőt beavatkozások is eszközölhetők.)

Szoftveres támogatás

A kiszolgálókhoz a gyártó különféle programokat mellékel. Ezek egy része jár a hardverhez, másokért külön licencdíjat kell fizetni.
Ezek lehetnek:

- eszközmeghajtók, frissítések (Driver, Firmware),
- a kiszolgáló telepítését, konfigurálását támogató, végző programcsomagok,
- a kiszolgáló távmenedzselését megvalósító programok,
- a kiszolgáló állapotáról különféle információkat nyújtó programok (monitorozás).

Támogatás

Szinte mindegyik kiszolgálógyártó különféle támogatási csomagokat ajánl és forgalmaz a kiszolgálóihoz.
Ezek lehetnek, a teljesség igénye nélkül:

- emelt szintű szerviz,
- terméktámogatás,
- termékkövetés,
- szoftverfrissítések,
- tanácsadás,
- telepítés, beépítés,
- segédanyagok, kellékek.

A kiszolgálók csoportosítása

A kiszolgálókat többféleképpen csoportosíthatjuk.

Általános célú kiszolgálók

Mindennapi kiszolgálónk lehet, ha nincsenek különleges igényeink.

Kisvállalkozások (Small Business)

Ez azoknak ajánlott, akik mindent egyben szeretnének látni. Itt általában a szerényebb teljesítmény, kedvezőbb ár is fontos szerepet játszik.

Gyors javíthatóság

Sokféle redundáns részegységet tartalmaz. Nagy megbízhatóságú környezetbe ajánlott.

Nagyvállalat (Enterprise)

Nagyágyú, nagy teljesítményt nyújt, jól skálázható, megbízható eszköz. Az ára is tükrözi a pozícióját.

Speciális feladatok

Speciális kivitellel rendelkező kiszolgálók különleges feladatokra.

- Mobil vagy speciális ipari környezetbe szánt, megerősített fizikai kivitel.
- Speciális csatlakozások, perifériák célfeladatok ellátására.

Moduláris kiszolgálók és pengék (blade szerverek)

Modul rendszerű, rackbe építhető kiszolgálók, elsősorban nagy mennyiségű kiszolgáló elhelyezésére használhatók.

Az I/O-modulok, tápegységek, diszkek, processzor vagy szerverkártyák, külön-külön igény szerint szerezhetőek be, konfigurálhatóak menedzselhetőek.

A kiszolgálók csoportosítása a rajtuk futó szolgáltatások alapján

Alkalmazáskiszolgáló

Kiszolgálóoldali alkalmazások futtatására használható. Általában a klasszikus kliens–szerver programok ilyenek.
Meghatározó tényező itt a memória- és processzorigény, jellemzőek a multiprocesszoros hardverek.

Adatbázis-kiszolgáló

Az adatok tárolására használható. Sok esetben az üzleti logikát a teljesítmény növelése érdekében külön alkalmazáskiszolgálóra teszik, az adatbázis-kezelő részére dedikálnak kiszolgálót.
Memória- és háttértárigénye meghatározó. A nagy tömegű adatok gyors eléréséhez szükség lehet nagy sebességű I/O-eszközökre is. Jellemző lehet a külső háttértár.

Alapinfrastruktúra

Alapvető kiszolgálófunkciók megvalósítása. Ezek általában nem igényelnek különösen sem memória-, sem processzor-, sőt általában diszkkapacitást sem, és a hálózati sávszélességük sem meghatározó, viszont elvárás a nagy rendelkezésre állás, hiszen alapvető szolgáltatásokat biztosítanak (pl. DNS, DHCP, azonosítás (DC) stb.).

Infrastruktúra

A hálózati infrastrukturális funkciók megvalósítására szolgálnak. Igényük hasonló lehet az előző kategóriához, de szóba jöhetnek speciális hardverelemek is, elsősorban I/O-eszközök (bridge, proxy, router stb.).

Felügyelet/menedzsment

A nagy tömegű kiszolgáló/kliens, illetve a magas rendelkezésre állás igénye teszi szükségessé ezeket az eszközöket. Ezek általában nem igényelnek különleges hardverkomponenseket, a kiszolgálók és a kliensek felügyeleti-menedzsment szoftverei, alkalmazásai futnak rajtuk.

Mentés

Ahogyan egyre nagyobb értékű adatok egyre komplexebb kiszolgálókörnyezetben kerülnek elhelyezésre, úgy nő, bonyolódik a mentési alrendszer is. Egyes helyeken kiemelt fontosságú a megfelelő mentési stratégia, itt a különféle mentési folyamatokra külön kiszolgálókat állítanak be. Hardverigényük az alkalmazott mentési megoldástól függ.

Állománykiszolgáló

Egy másik klasszikus alapfeladat az állományok tárolása. Itt a fő cél az állományok minél gyorsabb elérése. Ehhez nagy teljesítményű I/O alrendszerre, gyors, biztonságos diszk-alrendszerre van elsősorban szükség. Jellemző lehet a külső, intelligens háttértár használata. Egyre nagyobb teret hódítanak az egybecsomagolt kiszolgálómegoldások (appliance), ahol egy célhardver beágyazott operációs rendszerrel célfunkciót valósít meg, így még nagyobb teljesítmény érhető el.

Tűzfal

Feladata a hálózat védelme, elválasztása más hálózati részektől, hozzáférés elleni védelem. Ehhez általában több hálózati adaptert kell tartalmaznia. A VPN-, HTTPS-forgalomhoz megfelelő számítási teljesítményre van szüksége, szempont lehet a gyors I/O-rendszer is.

Levelezőkiszolgáló

Feladata az elektronikus levélfogalom kezelése.
Nagy teljesítményű I/O-rendszer, esetleg külső háttértár használata megfontolandó lehet.
Lehet a kliensek és a más kiszolgálókkal történő kapcsolattartásra dedikált kiszolgálókat is beállítani.

Kommunikáció

Egyéb elektronikus kommunikációs szolgáltatások (csevegés, azonnali üzenetküldés, konferencia, videokonferencia stb.) üzemeltetésére használható. Terheléstől és az adott szolgáltatástól függő hardveren futhat.

Nyomtatókiszolgáló

Nyomtatók, nyomtatási sorok, nyomtatási munkák kezelése a feladata.
Ma már viszonylag ritkán használnak közvetlenül a kiszolgálóra kötött nyomtatókat. Általában a hálózatra kötött nyomtatók kezelésére használják.
A gyors hálózati kapcsolat mellett fontos lehet a megfelelő méretű háttértár a nyomtatási sorok kiszolgálásához.

Webkiszolgáló

Webanyagok kezelésére használható.
A klasszikus statikus oldalak mellett egyre több, dinamikus tartalommal rendelkező webkiszolgálót üzemeltetnek, egyre több alkalmazást írnak úgynevezett vékonykliens-technológiával, amikor az alkalmazás a webkiszolgálón fut, és a kliensek az alkalmazás felületét webböngészőn keresztül érik el. A multimédiás vagy szórt (streaming) webalkalmazások is egyre inkább teret hódítanak.
Terheléstől és feladatkörtől függ a megfelelő hardver.
Speciális webkiszolgáló-farmok és az ehhez tervezett pengehardverek fedik le ezt a célterületet.

Záró gondolatok

A kiszolgálókon egyre több, egyre kritikusabb szolgáltatások futnak. Kiválasztásuk, üzemeltetésük egyre komplexebb feladat. Ehhez a feladathoz természetesen szakértői segítséget is igénybe lehet venni, sokszor azonban célszerű magunknak is a kiválasztás alapelveivel tisztában lennünk. Cikkemmel, remélem, sokakat közelebb juttathatok ehhez a célhoz.

Megyesi Barnabás
megyesi.barnabas@meb.hu
üzemeltetésvezető
MCSE, HP ASE

A cikkben szereplő URL-ek:

[1] http://www.hp.com/products/servers/platforms/k

Windows szolgáltatások 2. rész

Melyiket használjuk és melyiket ne?

A szolgáltatásokkal kapcsolatban számos alapfokú és haladóknak szóló információt nyújtottunk a sorozat első részében. Most megpróbáljuk egyesével sorra venni egy frissen telepített Windows Server 2003 tartományvezérlő alapértelmezett szolgáltatásait.

Használati utasítás ezen íráshoz

Lássuk melyek azok a szolgáltatások amelyek feltétlenül kellenek, amelyek nélkül összedől a rendszer és mi az ami leállítható (vagy éppen muszáj letiltani), esetleg el is távolíthatjuk végleg. Minden rendszer más és más, ha csak a rendszerrel szemben támasztott követelményekből indulok ki, már akkor is érvényes ez a tétel. De igaz ez a hardver és a hálózatban/tartományban betöltött szerepek vonatkozásában is. Semmilyen az operációs rendszeren kívüli kiszolgáló vagy egyéb szoftver jelenlétével nem számolunk, nincs Exchange, ISA, SQL Server és egyéb komponensek sem. Elsősorban alapszintű tájékozódásra javaslom forgatni ezeket az oldalakat, és egyúttal leszögezném, hogy a mentés nélküli, éles rendszeren ténykedők harakirit elkövető operációs rendszereivel kapcsolatban semmiféle kártérítésre nincs mód.

Egy kis statisztika

Számításom szerint egy friss Windows Server 2003 tartományvezérlőnél (a telepített komponensektől némiképp azért függően) közel 100 szolgáltatás van. Ennek körülbelül a fele automatikus indítású, a manuálisan indíthatók száma 32-35, a maradék (kb. 12-15) pedig le van tiltva alapértelmezés szerint. Ha a bejelentkezés fiókja szerint nézzük meg a listát, akkor a Local System az aranyérmes, hiszen mind a Network Service, mind a Local Service fiók alkalmazása külön-külön is csak kb. 9-10 szervizre jellemző, az összes többinek a helyben korlátlan hatalmú Local System fiók jut. Ha a mennyiségi fejlődést nézem az operációs rendszerekhez viszonyítva, akkor érdekes lehet az, hogy a Windows 2000-hez képest több mint negyven új szolgáltatás van az XP-ben és/vagy a Windows 2003 Serverben. Az utóbbiban egyébként tíznél több olyan új szerviz van, amely korábban még sohasem szerepelt egy Windows operációs rendszerben sem, íme a tömörített lista: ASP.NET State Service, HTTP SSL, Message Queue Triggers, Remote Administration Service, Remote Server Manager, Resultant Set of Policy Provider, Special Administration Console Helper, Terminal Services Session Directory, Virtual Disk Service, Web Element Manager, Windows Media Services, WinHTTP Web Proxy Auto Discovery Service, IAS Jet Database Access.

Az első 10

Ebben a részben összesen 10 szervizt veszünk górcső alá. Közös tulajdonságuk az, hogy mind az újonnan (XP, Windows Server 2003) bevezetett Local Service fiókkal futó szervizek, amelyekről az előző rész alapján tudnunk kell már egy-két dolgot, de azért foglaljuk össze röviden:

- gyengített jogosultsági körrel futnak, ami körülbelül megfelel egy átlagos felhasználói fiók lehetőségeinek,
- a hétköznapi értelemben vett erőforrás hozzáférés (objektumokhoz) nem megengedett,
- hálózati hitelesítés sem jár ehhez a fiókhoz,
- és végül: nincs jelszavuk sem.

Ha a Local Service fiók használata alapján csoportosítunk akkor a következő szolgáltatásokat lehet egy kalap alá venni:

- Alerter
- Application Layer Gateway Service
- Remote Registry
- Smart Card
- TCP/IP NetBIOS Helper
- Telnet
- Uninterruptible Power Supply
- WebClient
- Windows Image Acquisition
- WinHTTP Web Proxy Auto-Discovery Service

A részletezés előtt röviden szólnék az elnevezésekről:
A szerviz neve után zárójelben a Windows Server 2003 magyar nyelvű súgójában szereplő nevet jegyeztem be. A szerviz rövid neve az a név amely parancssorból például a „net start" parancs után következik. Annál a szerviznél, ahol nincs külön futtatható állomány, a processzt tartalmazó .dll szerepel, plusz az svchost.exe is, mint futattó alkalmazás. Ez utóbbi módszerről a sorozat előző cikkében volt szó.

Alerter (Riasztás)

A szerviz rövid neve: Alerter
Az alkalmazás neve: alrsvc.dll (svchost.exe)
Függés: Workstation szerviz
Függesztés: –

Porthasználat: TCP: 2869, dinamikus; UDP: 1900
Alapértelmezett indítás: letiltva

Az Alerter szolgáltatással az adminisztratív üzenetek küldése történik. A felhasználók/számítógépek részére biztonsággal, hozzáféréssel, replikációval, nyomtatással vagy pl. a felhasználói munkamenetekkel kapcsolatos riasztásokat küldhetnek a különböző alkalmazások (pl. a szünetmentes tápegységek szoftvereinek bevett szokása ez). Mindennapi használatban lehet akkor is, ha a Performance Monitor „figyeli" a rendszerünket például hibakeresés céljából. Ekkor az általunk kiszemelt objektumok nem rendeltetésszerű működéséről az Alerter szerviz tudósíthat bennünket.

Ahhoz hogy a feladó géptől a felhasználóhoz eljusson ez az üzenet, a Messenger (Üzenetküldő) szolgáltatásnak is futnia kell a felhasználó gépén. Ha az Alerter szolgáltatás le van állítva vagy tiltva, akkor az üzenet csak helyben kézbesítődik.

Application Layer Gateway Service
(Alkalmazási réteg átjárószolgáltatása)

A szerviz rövid neve: Alg
Az alkalmazás neve: Alg.exe
Függés: –
Függesztés: Internet Connection Firewall; Internet Connection Sharing
Porthasználat: TCP: 21, dinamikus
Alapértelmezett indítás: kézi, leállítva

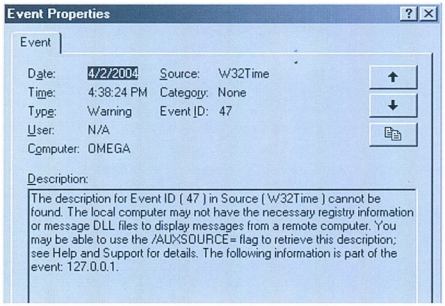
Az ALG új szerviz, érthető okokból a Windows 2000-nél még nem találkozhattunk vele, hiszen tulajdonképpen a Windows 2003/XP beépített tűzfala és az internet elérés megosztását végző alkalmazás kiegészítője ez a szerviz. Konkrétan a külső szoftverek protokollbővítményei számára nyújt opciókat, azzal, hogy lehetővé teszi, hogy a beépített tűzfalon átjuthassanak és az ICS mögött is működhessenek. Ezek a bővítmények így rendeltetésszerűen nyithatnak és változtathatnak portokat illetve IP címeket. Maga a Windows Server 2003 Standard es Enterprise változata egyetlen ilyen bővítménnyel rendelkezik, ez pedig az FTP protokollhoz tartozik. Amikor a szerviz kifelé menő FTP forgalmat tapasztal, kicsípi a csomagból azt a portszámot, amelyen a visszafelé forgalmazás történne, és ezt felhasználva egy dinamikus port átirányítást hajt végre az adatcsatorna részére.

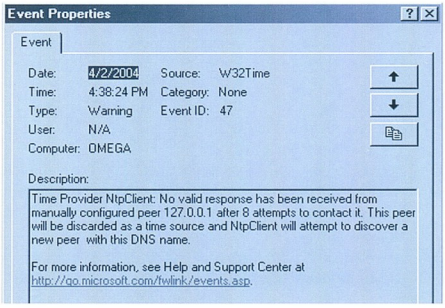
Ha leállítjuk vagy letiltjuk ezt a szolgáltatást, akkor a tűzfal és az internet megosztás nem fog működni, valamint a külső gyártók tűzfal szoftvereivel is hasonló gondjaink lesznek. Kézi indítású a szerviz alapállapota, így elvileg csak akkor indul el, ha valóban szükség van rá, de ha a fentiek alapján még sincs rá szükségünk, akkor nyugodtan letilthatjuk.

Remote Registry
(Távoli rendszerleíró adatbázis)

A szerviz rövid neve: RemoteRegistry
Az alkalmazás neve: regsvc.dll (svchost.exe)
Függés: Remote Procedure Call szerviz
Függesztés: –
Porthasználat: –
Alapértelmezett indítás: automatikus

Ez a szolgáltatás a távoli felhasználók számára nyújt hozzáférést az adott gép regisztrációs adatbázisához. Egy tartományban elsősorban az üzemeltető használja a kliens gépek felügyelete/kezelése során. Ezenkívül van még egy érdekes szerepe: a Windows Server 2003/XP valamelyik változatát futattó távoli gépen az Eseménynapló egy-egy eseményének a tulajdonságlapján a Description illetve a Category mezőjének leolvasásához is szükséges. Ha ez nincs meg, akkor adja azt a levelezési listákon már többször felmerült hibaüzenetet az Eseménynapló, amelyet a felső képen is lehet látni a Description mezőben (az alsó képnél a különbséget az okozza, hogy elindítottam a szervizt).





Az Eseménynaplónak nem mindegy, hogy működik-e a Remote Registry szolgáltatás

Persze, ha már a távoli eseménynapló elindításakor sem fut a szerviz, akkor már a naplóállományokat sem lehet kezelni. Ha viszont közben újraindítjuk a szervizt, akkor újra csatlakozni is kell az adott géphez.

Ha letiltjuk ezt a szolgáltatást, akkor már csak a helyi gépről érhető el a regisztrációs adatbázis, arról viszont továbbra is korlátozások nélkül. Biztonsági szempontból önálló gépen ennek mindenképpen van értelme, persze például egy tartományban a praktikum erősebb érv lehet a letiltással szemben.

Smart Card (Intelligens kártya)

A szerviz rövid neve: SCardSvr
Az alkalmazás neve: SCardSvr.exe
Függés: Plug and Play szerviz
Függesztés: –
Porthasználat: –
Alapértelmezett indítás: kézi, leállítva

Értelemszerűen az intelligens kártya használatot támogató szolgáltatásról van szó, amely akkor indul, ha egy kártyát gyömöszölünk az olvasóba.

Ha leállítjuk vagy letiltjuk akkor nem történik semmi extra fennakadás, csak annyi, hogy ez a támogatás nem áll rendelkezésre.

TCP/IP NetBIOS Helper (TCP/IP NetBIOS támogató)

A szerviz rövid neve: LMHosts
Az alkalmazás neve: lmhsvc.dll (svchost.exe)
Függés: AFD Networking Support Environment, NetBIOS Over TCP/IP, TCP/IP Protocol Driver, IPSec Driver
Függesztés: –
Porthasználat: TCP: 139; UDP: 137, 138
Alapértelmezett indítás: automatikus

Ez a szolgáltatás fontos, sőt a legtöbb esetben még ma is nélkülözhetetlen: a TCP/IP feletti NetBIOS forgalomhoz, valamint a NetBIOS névfeloldási rendszer használatához szükséges. Ennek megfelelően például az állomány és nyomtató megosztás vagy a hálózati belépés, esetlegesen a DNS szolgáltatás sem működik megfelelően ha leállítjuk, vagy esetleg letiltjuk. Ez a szerviz gyakorlatilag a NetBT kiterjesztése, ezért a NetBT-vel kommunikáló kliensek pl. a Redirector, Server, Netlogon vagy a Messenger szervizek sem „élnek" meg a hiányában, de pl. a tartomány alapú házirendekkel kapcsolatban is számíthatunk néhány problémára (pl. szoftvertelepítés).

Telnet (Telnet)

A szerviz rövid neve: TlntSvr
Az alkalmazás neve: tlntsvr.exe
Függés: Remote Procedure Call, NT LM Security Support Provider szervizek, TCP/IP Protocol Driver, IPSec Driver
Függesztés: –
Porthasználat: TCP 23
Alapértelmezett indítás: letiltva

A komoly hagyományokkal rendelkező és valószínűleg sokak számára jól ismert Telnet szolgáltatás a TCP/IP protokollkör egyik tagja. A kiszolgálón egy távoli munkamenet létrehozását teszi lehetővé, parancssorból. Hagyományai ellenére (vagy talán azért?) a Telnet protokoll bevallottan nagyon alacsony szintű adatvédelmet biztosít. A kliens és a kiszolgáló között minden adat, beleértve a jelszavakat is, egyszerű szövegként továbbítódik. Ugyan a Windows 2000 Server óta a Telnet munkamenet használhat NTLM hitelesítést, így a helyzet valamivel jobb mint korábban, de ez még mindig édeskevés. A legjobb megoldás ha meghagyjuk a szervizt a gyári induló értéken, azaz letiltva (paranoiások pedig el is távolíthatják, pl. a SrvInstw.exevel **[1]**).

Uninterruptible Power Supply (Szünetmentes áramforrás)

A szerviz rövid neve: UPS
Az alkalmazás neve: ups.exe
Függés: -
Függesztés: –
Porthasználat: –
Alapértelmezett indítás: kézi, leállítva

Csak a teljesség kedvéért említjük meg, mert szintén egyértelmű a szerepe: a csatlakoztatott szünetmentes tápegységek és az operációs rendszer kommunikációját valósítja meg. Összesen egy megjegyezni valóm van csak hozzá: amennyiben USB csatlakozású a szünetmentes tápegység, akkor bátran letiltható, ugyanis ez a szerviz csak a soros portokon képes dolgozni. Ha viszont az operációs rendszer egy soros porton működő UPS-t észlel, azonnal elindítja és automatikussá teszi az indító állapotát is.

WebClient (WebClient)

A szerviz rövid neve: WebClient
Az alkalmazás neve: davclnt.dll (svchost.exe)
Függés: WebDav Client Redirector szerviz
Függesztés: –
Porthasználat: TCP 80
Alapértelmezett indítás: letiltva

A WebClient szolgáltatás szintén a Windows Server 2003/XP operációs rendszerekben jelent meg először. A Win32 alkalmazások számára megengedhetővé teszi internetes és intranetes dokumentumok készítését, olvasását és írását. A szervizt a WebDAV (Web Distributed Authoring and Versioning) protokoll használja, ami pedig egy olyan állománykezelő szolgáltatás, amely a HTTP (1.1) protokoll hátán utazik, és gyakorlatilag teljes körű állománykezelési lehetőségeket nyújt (beleértve a Drag and Drop-tól kezdve az állomány megosztásig mindent). A WebDav Client Redirector szerviz segítségével pedig akár tűzfalakon és útválasztókon keresztül is működik (persze engedélyezni/publikálni azért kell). Ha ki szeretnénk aknázni a WebDav előnyeit (bevett szokás szerint pl. az intraneten), vagy bizonyos szolgáltatások esetén az interneten is, akkor a klienseken feltétlenül futni kell ennek a szerviznek (pl. Csoportházirendből is kötelezővé tehetjük, lásd előző szám), ellenkező esetben viszont inkább hagyjuk meg letiltva, mert igencsak érzékeny területen dolgozik.

Windows Image Acquisition (Windows Image Acquisition)

A szerviz rövid neve: Stisvc
Az alkalmazás neve: wiaservc.dll (svchost.exe)
Függés: Remote Procedure Call, Shell Hardware Detection szervizek
Függesztés: –
Porthasználat: -
Alapértelmezett indítás: letiltva

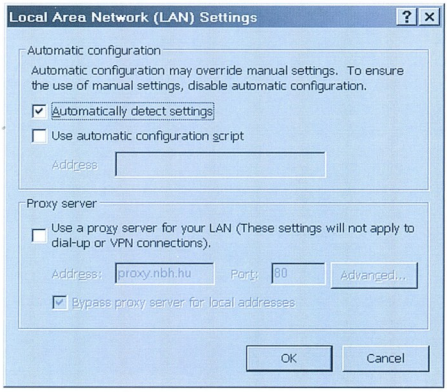
Üzemeltetők számára valószínűleg nem túl érdekes szolgáltatás a WIA, amely már a Windows Millenniumban is jelen volt (a Windows 2000-ben viszont nem). Kompakt megoldás, mert támogatja a SCSI, IEEE 1394 és az USB rendszerű képolvasókról és digitális fényképezőgépekről történő képolvasást. Gyakorlatilag elmondható, hogy alkalmazás szolgáltatási szinten a WIA váltja fel a TWAIN rendszert. A WIA a Windows Driver Model architektúra része és egyaránt jelent egy API-t az alkalmazások, valamint egy eszközvezérlő csatolófelületet az eszközök illesztőprogramjai részére.

Igazából nem nagyon értem, hogy egy kiszolgáló számítógépen mi haszna lehet ennek a szolgáltatásnak, mindenesetre annak ellenére, hogy valószínűleg biztonsági rizikót nem rejt a használata, hagyjuk meg az alapértelmezett tiltást.

WinHTTP Web Proxy Auto-Discovery Service (WinHTTP automatikus webproxy-kereső szolgáltatás)

A szerviz rövid neve: WinHttpAutoSvc
Az alkalmazás neve: winhttp.dll (svchost.exe)
Függés: DHCP Client szerviz, AFD Networking Support Environment, TCP/IP Protocol Driver, IPSec Driver
Függesztés: –
Porthasználat: TCP 80
Alapértelmezett indítás: kézi, elindítva

Ez a szolgáltatás érdekesebb mint az átlag, így érdemes neki egy kicsit nagyobb teret szentelni. A feladata a hálózatban általában oly fontos proxy beállítások keresése és átadása a HTTP kliensek (például a böngészők) részére.



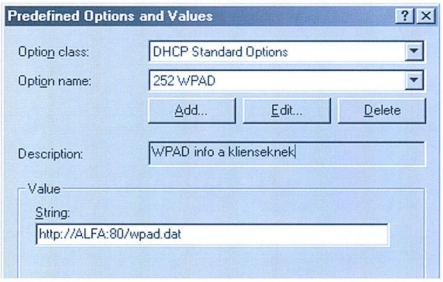
Az automatikus proxy detektálás

Ha pl. az Internet Explorerben bekattintjuk (vagy központilag a Csoportházirendben az egész tartományra kötelezően előírjuk) az ábrán is látható opciót („Automatically detect settings"), akkor a szerviz elkezdi keresni a hálózaton a proxy információkat.

De hogyan fogja megtalálni? Megmondja a DNS szerver! Ugyanis a DNS zónánkban manuálisan felvehetünk egy WPAD rekordot, CNAME típusként (pl. wpad.tjszki.hu), ami a proxy szerverre mutat, így egyszerűen kideríthető, hogy hova kell kapcsolódnia.

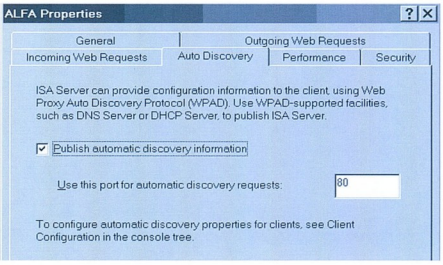
Sőt, erre a feladatra a DHCP szerver is alkalmas, ti. a 252-es számú opció beállítása ugyanezt az eredményt hozhatja. Ehhez először kattintsunk a DHCP konzolon a szerver nevére a jobb gombbal és válasszuk a „Set Predefined Option" menüpontot. Ezután az Add... gombbal adjuk hozzá a paramétereket a következő képen látható végeredménnyel. Két lépésben történik meg majd mindez, hiszen először magát a 252 számú bejegyzést kell felvennünk, majd azután azt az útvonalat megadni, amelyen a wpad.dat állomány jelen van.

Így vagy úgy, de végül is lekerül a proxy információ a kliensre és ettől kezdve pl. a böngésző az adott proxy kiszolgálóhoz kapcsolódva mehet ki a netre (vagy nem).



A WPAD info hozzáadása a DHCP-ben

Ennek a folyamatnak a sikerességéhez még az is szükséges, hogy a proxy szerverünkön (jelen esetben az ISA Serveren) bepipáljuk a „Publish automatic discovery information" opciót.



Az ISA-ban is engedjük meg...

Ezen lehetőségek azonban csak a WinHTTP 5.1-es verziójának használata esetén állnak a rendelkezésünkre (a Windows 2000 SP3, Windows XP SP1 ill. a Windows Server 2003 is tartalmazza).

De mi ennek az egész folyamatnak, így a szerviz használatának is az előnye? Egy darab proxy szervernél az, hogy teljesen automatikus a proxy beállítás, ráadásul dinamikusan változhatnak a szerver adatai. Egy proxy tömb esetén (ha az ún. PAC azaz Proxy Auto-Configuration szkriptet elkészítjük és a WPAD protokollal le is töltjük) azt is megtehetjük, hogy URL-től függően más és más proxy szerver felé irányítjuk a kérést.

Cikkünk következő részében

Terveink szerint a Network Service fiók nevében futó szolgáltatásokat mutatjuk be.

GÁL TAMÁS
MCSE, MCSA, MVP
gtamas@tjszki.hu

A cikkben szereplő URL-ek:

[1] http://tinyurl.com/2zs3e
[2] http://tinyurl.com/342tr

MVP-találkozó Amerikában

Újdonságok a .NET Framework 2.0-ban

A tavaszi amerikai MVP-találkozón számos újdonságot hallhattunk a jövő technológiáiról. Cikkünkben elsősorban a Visual Studio 2005 és a .NET Framework 2.0 (Whidbey) újdonságairól számolunk be.

Kik is az MVP-k?

Az MVP a Most Valuable Professional (legértékesebb szakértő) rövidítése. Kik is vagyunk mi? Ugyanolyan átlagemberek, mint Te, kedves olvasó. Azért kaptuk ezt a címet, mert évek óta sok embernek segítünk a levelezőlistákon és egyéb fórumokon abban, hogy eligazodjanak, és sikeresen használják a Microsoft-technológiákat. Mi nem vagyunk a Microsoft alkalmazottai. Ha azok lennénk, nem is kaphatnánk meg az MVP címet. Ez a „titulus" nem jelenti azt, hogy mi „szupermenek" vagyunk, akik minden kérdésre képesek válaszolni, és még az arcunk se lett ettől (sokkal ☺) nagyobb. Viszont a cím hozományaként sok olyan erőforráshoz jutunk és fogunk még hozzájutni, ami segít minket abban, hogy még jobban megértsük a Microsoft-technológiákat, és még több embernek tudjunk segíteni.

Ilyen erőforrások többek között:

- Belső Microsoft kapcsolattartó, akin keresztül a levelezőlistákon felmerülő, számunkra megoldhatatlan problémához kérhetünk belső segítséget. Ez különösen akkor jön jól, ha például valami bug-gyanús, mert így nem futunk felesleges köröket egy ismert probléma miatt.
- Nem nyilvános MVP hírcsoportok. Ezek csak az MVP-k számára hozzáférhető csoportok, amelyeken mind a többi MVP-vel, mind a Microsoft dolgozóival tudunk kommunikálni. Mivel az MVP-k zöme nagyon jó a saját szakterületén, itt nagyon értékes infókat tudunk összeszedni, amelyek egy részét a nyilvánossággal is megosztjuk (hacsak nem titoktartási szerződéssel védett tartalomról van szó).
- Knowledge Base (tudásbázis) cikkek, amelyek még nem jelentek meg, szerkesztés alatt állnak.
- A magyar MVP-k közül én például hozzáférhetek, illetve publikálhatok a Whidbey buglistájához/ba (Visual Studio 2005 és .NET Framework 2.0). Azaz ha valami gyanús a Whidbey-jel kapcsolatban, akkor én leszek a gateway a bug adatbázis és a nyilvánosság között. Mellesleg az adatbázis előbb-utóbb nyilvános lesz, csak előbb rajtunk, MVP-ken tesztelik.
- Hozzáférhetünk, és „debugolhatjuk" egyes termékek forráskódját, például VS.NET, Windows stb.

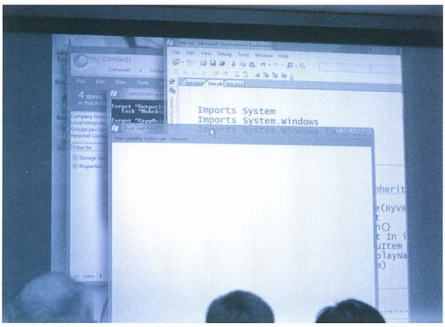
A lényeg tehát, hogy mi kívülálló emberek vagyunk, de hozzáférünk egyes belső Microsoft-információkhoz annak érdekében, hogy hatékonyabban segíthessünk másoknak.

Az MVP Summitról

Az MVP Summit egy, a világ különböző pontjairól érkezett MVP-k számára az Egyesült Államokban tartott szakmai rendezvény volt idén márciusban. A helyszín Seattle és Redmond. A rendezvény célja az MVP-k közötti nemzetközi szintű kapcsolatépítés, valamint közvetlen visszacsatolás a termékekről a Microsoft fejlesztőihez.

A redmondi konferencia-központ

A rendezvény gerincét természetesen az előadások alkották, ahol első kézből kaphattunk szakmai információkat. Ezekből szemezgetek a továbbiakban. Jövőbe néző infókat, amelyek a saját szűrőmön jöttek át, ennek megfelelően kéretik érdeklődve, de kritikusan olvasni.



Longhorn Glass Effect

ADO.NET a Whidbeyben (2005)

A DataSet binárisan is képes lesz szerializálni magát, nem csak XML formátumban. Ez nemcsak azért érdekes, mert kevesebb sávszélességet fogyaszt az adatátvitel, hanem azért is, mert így sokkal kevesebb memóriát használnak mind generáláskor, mind feldolgozáskor, hisz nem kell terjedelmes XML-stringeket összeállítani. Az árnyoldal viszont az, hogy csak remoting esetén használhatjuk ki a bináris átvitelt, webszolgáltatások esetén nem.

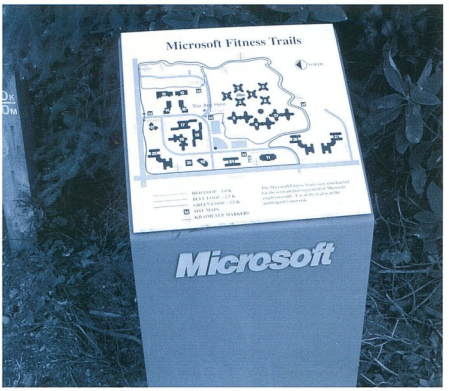
Seattle alulról

A DataSet indexelését átírták, így sokkal gyorsabb lett. Miért, használ a DataSet indexet? Hogyan lehet rajta létrehozni? Direktben sehogy, de amikor például Primary Keyt hozunk létre egy oszlopon, akkor keletkezik index az oszlop adatain, hogy például a Find() metódus gyorsan tudjon keresni kulcs alapján. Vagy a DataView-k is használják az indexeket, automatikusan például a szűrt oszlopon.
Gondolom, ismerős az a helyzet, hogy egyetlen táblát kellett volna kezelni, mégis DataSetet kellett használni, mert a DataTable nem tud sok dolgot, amit a DataSet igen. Ezt korrigálják a 2.0-ban, a DataTable mindent fog tudni, amit a DataSet, kivéve persze a több táblával kapcsolatos szolgáltatásokat.
Sok adat DataTable-be töltésére lesz egy Load metódus, ami valamilyen IDbDataReader tartalmát képes nagyon gyorsan betölteni a táblába.

A DataView tartalmát lehet majd materializálni egy új DataView.ToTable() metódussal. Hasznos lehet, ha szűrt kimenetet kell például hálózaton keresztülzavarni.
Ami új és nekem meglepő volt, hogy az aszinkron SQL parancsvégrehajtás (ez még nincs az 1.1-ben) nem használ ThreadPool szálakat. Azaz ha egyszerre 500 async parancs is fut éppen akkor, nem fog 500 szálat allokálni az ADO.NET, így nem is fogja kimeríteni a ThreadPoolt. Az előadó szerint átírták a Managed Data Provider hálózatkezelő részét, hogy ez így működhessen.
E tudás ismeretében mutatott egy nagyon ravasz megoldást, amivel extrém nagy terheltségű szerver esetén nagy áteresztőképességet lehet elérni.

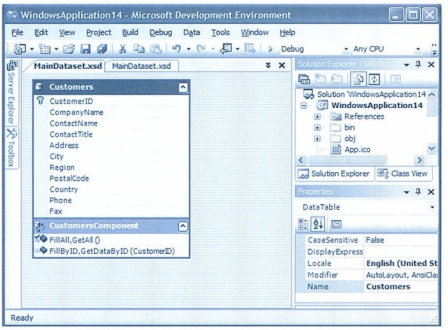
Látkép az egyik folyosó sarkában

Az ötlet a következő. Írjunk egy aszinkron HttpHandlert. A handlerben az induló eseménynél elindítjuk az aszinkron SQL parancsvégrehajtást. Mivel az nem blokkol, az aszinkron handler visszatér, így nem fogunk le ASP.NET-szálat sem a ThreadPoolból (ez a ThreadPool egy olyan állatfaj, aki nagyon hamar ki tud merülni). Mivel az ADO.NET se használ felesleges szálakat, a szerver minimális erőforrásokkal vár a parancsvégrehajtásra.
Amikor az aszinkron SQL parancsvégrehajtás megtörténik, egy callback metódus visszaszól, ahol befejezzük a HTTP-válasz generálását. Nagyon szellemes megoldás, bár várom magyarországi körülmények között azt a volumenű webes terhelést, ami miatt be kell vetni ezt a trükköt.
Ami érdekes, hogy az SqlConnection.Open()-nek viszont nem lesz aszinkron megfelelője, pedig még korábban így tervezték. Az ok az, hogy az SqlConnection megnyitásakor lehet, hogy a kapcsolatot fel kell venni egy elosztott COM+ tranzakcióba. Ebben az esetben viszont nem tudják garantálni, hogy a BeginOpen() tényleg azonnal visszatérjen, azaz az aszinkron BeginOpen() néha szinkron lenne. Ezért inkább nem lesz aszinkron. Érdekes dolog ez, hogy egyes problémákat még „házon belül" se egyszerű megoldani.



Ha egy redmondi kollégának tele van a feje a SOA-val, akkor mehet futni egy kört ☺

A Strongly Typed DataAdapter szellemes lesz. Már a 2003. őszi PDC Buildben láttam, hogy a DataSet tervezőben nem csak a táblák vannak benne, hanem a táblákhoz kapcsolódó SQL parancsok is fel vannak véve.



Típusos DataSet és a hozzá kapcsolódó adapterek a VS.NET 2005-ben

A Strongly Typed DataAdapter valami olyasmi lesz, hogy egy ST DataSetben a táblák szerkezete mellé a táblákat tápláló DataAdaptereket is felvehetjük a megfelelő SQL-parancsokkal vagy tárolt eljárásokkal együtt. Az adaptereket aztán némi kóddal meghajtják [Fill() hívások], így a végén például STDataSet.GetAllCustomers() hívással fel is töltöttük a DataSetünk egy vagy több tábláját.
Tetszőleges számú DataAdaptert hozhatunk így létre, például GetCustomerById() stb. néven.
Egy kis kódszösszenet az előbbi DataSet használatához:

```
CustomersComponent c = new CustomersComponent();
//Összes vásárló lekérése
MainDataSet.CustomersDataTable customers =
      c.GetAllCustomers();
//ID alapján egy vásárló lekérése
MainDataSet.CustomersDataTable onCust =
      c.GetDataByID("ALFKI");
```

Azaz a Designer megírja az adatelérő réteg nagy részét (a CustomersComponentet is az generálta)! Kényelmesen lehet majd dolgozni a Table Modul Patternnel **[2]**.
De mint tudjuk, az adatbázis alapú alkalmazások fejlesztésére van egy másik út is, amely elsősorban a Java-világban terjedt el, ez az, amit Martin Fowler Domain Modellnek **[3]** hív. Ebben az esetben a klasszikus OOP tervezési elvek alapján üzleti objektumokat hozunk létre, amelyek általában közönséges class-ok (osztályok). Például egy dolgozót modellezve lenne egy Employee osztályunk, amelynek egy példánya egy dolgozót reprezentál. Az adatokat, mint név, cím stb. tagváltozók tárolják. Az adott entitáson értelmezett műveleteket szokásos OOP módon az objektum metódusaival implementáljuk. A modell előnye, hogy az adatok és a rajtuk értelmezett üzleti logika jól egybe van zárva, így könnyen kezelhető, módosítható modellt kapunk. Sokan nem tudnak róla, de már az 1.0-ban is lehetséges volt üzleti objektumokat databindolni WinForms vagy WebForms vezérlőkhöz. Például az előbbi Employee példányokat egy ArrayListben tárolva simán meg lehet jeleníteni azokat egy DataGridben. Az oszlopok a publikus property-k tartalmát mutatják meg.
A Whidbey VS.NET 2005-ben Design Time is működni fog a binding. Azaz a Data Access Layer által visszaadott Employee objektumot a Property Browserben hozzá lehet bindolni egy vezérlőhöz, és a GUI már mutatja is, milyen adatok vannak az entitásban. Úgy kell ezt elképzelni, ahogy most a grid működik Stronly Typed DataSetekkel, azaz már Design Time látszik, hogy fog kinézni a vezérlő.

Zárszó

Amiről itt most beszámoltam, az csak két előadás volt a tizenegynéhányból, amit láttam. A jövőben a TechNet hasábjain írok majd a 2.0-s .NET Framework várható újdonságairól, a mostani cikknél sokkal mélyebb tartalommal.
Érdekes volt a találkozó, és izgalmas volt megfigyelni, hogyan éli mindennapi életét egy másik kultúra. De hadd említsek meg a sok szakmai élmény mellett egy igazán személyes örömöt is: egy hónapos kisfiam legjobb barátja az a rénszarvas lett, amit az MS Company Store-ban vettem. **[4]**

Soczó Zsolt
zsolt.soczo@netacademia.net
ASP.NET MVP, MCSE, MCSD, MCSD.NET, MCDBA, MCT

A cikkben szereplő URL-ek:

[1]: http://www.technetklub.hu
[2]: http://www.martinfowler.com/eaaCatalog/tableModule.html
[3]: http://www.martinfowler.com/eaaCatalog/domainModel.html
[4]: http://www.netacademia.net/staff/soci/balintka/

A magyar MVP-k névsora

- Balássy György, SharePoint Portal Server
- Gál Tamás, Windows Server Management
- Máthé Zoltán, SQL Server
- Moldova György, Office Systems
- Pintér Szabolcs, NT Server
- Soczó Zsolt, ASP.NET
- Subicz Péter, Windows Server Directory Services
- Sztanya Ferenc, Terminal Server

Elérhetőségeik **[1]**.

Tippek & trükkök

Fókuszban az SQL Server 2000

Egy-egy trükk megismerése sokat segíthet a mindennapi munkában. Cikkünkben az SQL Server 2000 használatához adunk tippeket.

Tábla sorainak egyetlen karaktersorozattá fűzése

Gyakran van szükség például oszloplisták különböző *SELECT*-ekben történő előállítására, begépelésére, esetleg dinamikus összeállítására, vagy éppen *INSERT...SELECT* kifejezésekben való megadására. Az ilyen jellegű feladat nem igényli kurzorok felhasználását.

```
CREATE FUNCTION [dbo].[ListStru] (@table SYSNAME)
RETURNS VARCHAR(8000)
AS
BEGIN
   DECLARE @stru VARCHAR(8000)
   SET @stru = ''
   SELECT @stru = @stru + QUOTENAME(COLUMN_NAME) +
                  ', '
   FROM INFORMATION_SCHEMA.COLUMNS
   WHERE TABLE_NAME =
                  OBJECT_NAME(OBJECT_ID(@table))
   ORDER BY ORDINAL_POSITION
   IF @stru <> ''
      SET @stru = LEFT(@stru, DATALENGTH(@stru) - 2)
   RETURN @stru
END
GO
PRINT [dbo].ListStru('Shippers')

Eredmény: [ShipperID], [CompanyName], [Phone]
```

A fenti függvény *(UDF – User Defined Function)* kódja az *INFORMATION_SCHEMA.COLUMNS* view-ból válogat le sorokat (preferálni kell ezeket a view-kat a direkt rendszertábla-lekérdezésekkel szemben, mert ezek dokumentált struktúrája verzióról verzióra állandó). Minden sor esetén a *COLUMN_NAME* oszlopot a *@stru* változóhoz adja hozzá (kezdő értékkel kell ellátni, hogy ne legyen benne *NULL* érték az összeadás miatt, valamint, hogyha a *SELECT* nem ad vissza egyetlen sort sem, akkor nem történik értékadás). A *QUOTENAME()* függvény használatával (a második paraméter elhagyása mellett) az oszlopneveket szögletes zárójelbe teszi, segítve ezzel a *TSQL* fenntartott szavakból összeállított oszlopnevekre való hivatkozást. Az *OBJECT_NAME(OBJECT_ID())* hívás során az aktuális adatbázis minősített táblanevei esetén is jól működik a függvényünk, amely a végén az összeállított, vesszővel elválasztott karakteres listával tér vissza.

Ne használjuk az 'sp_' prefixet saját objektumneveinkhez

Többször találkozom olyan saját készítésű objektumnevekkel (tárolt eljárások esetében), amelyek az SQL Server rendszerszintű tárolt eljárásainak *(SP – Stored Procedure)* elnevezési konvencióit használják. Ez nem csak azért célszerűtlen, mert nem különülnek el a rendszer-SP-k, a saját készítésű SP-ktől, hanem azért is, mert az SQL Server motor az ilyen *prefix*-szel ellátott SP indításakor először a *master* adatbázisban keresi az eljárást. Teszi ezt azért is, mert ezek a rendszer-SP-k általában bármely saját adatbázisból meghívhatóak, hiszen mindig a *master*-ben keresi először a rendszer. Ha *Profiler*-rel nézzük a végrehajtást, jól látható, hogy gerjesztünk ezzel egy *SP:CacheMiss* eseményt is. Amennyiben elkövetjük ezt a prefix-elési hibát, könnyen készíthetünk olyan tárolt eljárást is, amelynek neve megegyezik egy *master* adatbázisban lévővel, ilyenkor a sajátunk soha nem fog meghívódni, ami persze hamar kiderül. Tehát használjunk *prefixet* bátran a saját SP-inkhez, de ne 'sp_' legyen.

Használjunk tulajdonossal (owner) minősített objektumneveket

Felesleges folyamatokat takarítunk meg azzal, ha az objektumneveinket minősítjük a tulajdonosával. Bizonyára ismert a TSQL objektumok hierarchikus elrendezése. Azaz egy tábla teljesen minősített neve *(FQN – Fully Qualified Name)* a következő:

```
ServerName.DatabaseName.OwnerName.TableName
Pl.: ServerA.Northwind.dbo.Customers
```

Az egyes minősítők közé pontot teszünk. A táblanév objektumunk 3 minősítéssel rendelkezik, ezt a négyest nevezzük teljesen minősített névnek. Sőt, ugyan nincs feltüntetve, de a hierarchia tovább folytatható, például a táblához tartozó oszlopnévre újabb ponttal kötve tudunk hivatkozni. A minősítés egyes szintjei (az objektum szintjéig) balról kiindulva teljesen elhagyhatók, illetve köztes szintek is kihagyhatók, ilyenkor két pont követi egymást. Amennyiben nincs megadva szervernév, az aktuális szerver neve kerül behelyettesítésre, ha nincs adatbázis, az aktuális adatbázis neve. Ha a táblánk nevére tulajdonos nélkül hivatkozunk, a szerver először az aktuálisan bejelentkezett felhasználó által tulajdonolt objektumot keresi. Ha nem talál ilyet, akkor tovább keres *dbo*-val. Egy másik kiskapu lehet, ha az adott felhasználó nevében elkészül egy objektum, akkor nem az általunk feltételezett *dbo* tulajdonosú

objektumon dolgozik majd a rendszer, hanem az adott felhasználóval létrehozott objektummal. Tehát létrehozható végtelen számú azonos nevű és típusú objektum, ha különböző a tulajdonos. A másik indok, azok számára hasznos, akik gyakran választják az *EXEC()* helyett az *sp_executesql*-t (nagyon helyesen) dinamikus lekérdezéseik végrehajtására. Mint ahogy a *Books Online*-ban (BOL) is olvasható, csak akkor kerül ismételt felhasználásra a végrehajtási terv, ha az objektumnevek teljesen minősítettek (a *Profiler*-ben megfigyelhető az *SP:ExecContextHit* esemény). A harmadik indok, amikor tárolt eljáráson belül nem használunk tulajdonost, nem az aktuális SP-t futtató felhasználó nevében fog futni a kódunk, hanem az SP-t létrehozó kerül behelyettesítésre az objektumokhoz a tulajdonos minősítéséhez. Végezetül mindezek még fontosabbak az objektumok létrehozásakor, hiszen az általunk minősítetlen tulajdonosú *script*-elt objektumok, amennyiben nem a megfelelő szerepű felhasználó futtatja, könnyen előfordulhat, hogy a *dbo.TableName* helyett egy *UserName.TableName* nevű jön majd létre. Az elkészített objektum struktúra fejlesztési környezetből az ügyfél környezetbe juttatásakor ezekre nagyon figyelni kell. A minősített nevek ezt a procedúrát egyértelművé teszik.

Kód végrehajtása a szerver összes adatbázisára, illetve adott adatbázis összes táblájára

Létezik a *master* adatbázisban két nem dokumentált eljárás rövid TSQL utasítások az összes adatbázison, valamint adott adatbázis összes tábláján történő végrehajtásra. Rendszert építeni ezen eljárásokra nem szabad, de a jelenlegi *SQL2000*-es verzióban karbantartási célokra jó szolgálatot tehet olyan parancsok esetében, amelyek adott adatbázisra futnak csak le *(DBCC CHECKDB)*, vagy adott adatbázis adott táblájára *(DBCC DBREINDEX)*, vagy akár *SELECT* utasítások is makrózhatók általa.

```
EXEC sp_MSforeachdb 'SELECT ''?'' AS DBName,
↳ * FROM [?].dbo.sysusers'
GO
EXEC sp_MSforeachdb 'PRINT ''?'' DBCC CHECKDB
↳ (''?'')'
```

```
EXEC sp_MSforeachtable 'SELECT ''?'' AS TABLENAME
↳ EXEC sp_helpindex ''?'''
GO
EXEC sp_MSforeachtable 'PRINT ''?''
↳ DBCC DBREINDEX(''?'')'
```

Dátum műveletek

Érdemes az igen költséges karakteres konverziók *(CONVERT(VARCHAR,..., style)* helyett kihasználni a sokkal gyorsabb dátumkonverziós függvényeket:

```
DECLARE @d DATETIME
SET @d = '2004.02.10 10:15:30.100'
```

1. példa: a fenti dátumhoz tartozó hónap első napjának meghatározása:

```
SELECT DATEADD(month, DATEDIFF(month, 0, @d), 0)

Eredmény: 2004-02-01 00:00:00.000
```

2. példa: a fenti dátumhoz tartozó hónap napjainak száma (szökőévekben is jól működik):

```
SELECT DATEPART(DAY, DATEADD(DAY, -DATEPART(DAY,
↳ DATEADD(MONTH, 1, @d)), DATEADD(MONTH, 1, @d)))

Eredmény: 29
```

3. példa: a fenti dátumérték időkomponensének nullázása:

```
SELECT DATEADD(day, DATEDIFF(day, 0, @d), 0)
Eredmény: 2004-02-10 00:00:00.000
```

Rekordhalmazzal visszatérő tárolt eljárás adatainak további feldolgozása

Gyakori eset, hogy információt szeretnénk kinyerni rendszerszintű, illetve saját tárolt eljárásainkból, amelyek egy számunkra túl részletes vagy éppen nem elég részletes adathalmazzal *(SELECT)* térnek vissza. Talán a legismertebb szintaktika az *INSERT...EXEC*. Ilyenkor elkészítünk egy, a visszakapott rekordhalmaz struktúrájával megegyező üres táblastruktúrát. Majd ebbe bele *INSERT*-áljuk az *SP* kimenetét.

```
CREATE TABLE #Temp (spid SMALLINT, ecid SMALLINT,
      status NCHAR(30), loginname NVARCHAR(128),
      hostname NCHAR(128), blk CHAR(5),
      dbname NVARCHAR(128), cmd NCHAR(16))
GO
INSERT INTO #Temp EXEC sp_who
```

Igen ám, de elég körülményes kitalálni, vajon milyen adattípusú oszlopok képezik a visszakapott rekordhalmazt. Erre vagy van dokumentáció *(BOL)*, vagy az *SP* olvasható tartalommal rendelkezik. Ennél egyszerűbb módszer az *OPENQUERY(), OPENROWSET()* függvények használata, amelyek kimenetét egy táblába irányítjuk, majd az *INFORMATION_SCHEMA.COLUMNS view*-ból akár le is kérdezhetjük az oszlopok típusait, méreteit:

```
SELECT * INTO Test
FROM OPENQUERY(ServerName, 'EXEC sp_who ') AS a
--FROM OPENROWSET('SQLOLEDB','ServerName';
--↳ 'UserName';'Password','EXEC sp_who ') AS a
GO
SELECT COLUMN_NAME, DATA_TYPE,
     CHARACTER_MAXIMUM_LENGT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Test'
```

MÁTHÉ ZOLTÁN
mathez@bsi.hu
MCSD, SQL Server MVP

Ami a hivatalos Microsoft tanfolyamokból kimaradt...

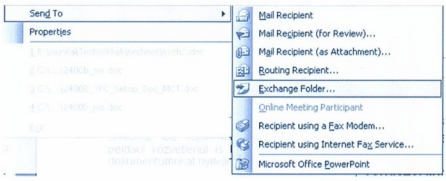
Exchange 2000/Exchange Server 2003 – II. rész

Az előző számunkban közölt cikk folytatásaként újabb Exchange-érdekességeket, tanulságokat ismertetünk.

Az itt leírt témák – csakúgy, mint előző számunkban – most sem kezdőknek szólnak! Akik nem értenek meg mindent, azoknak javasoljuk a 2400-as kódjelű „Implementing and Managing Microsoft Exchange Server 2003" tanfolyam elvégzését. Bátorítok minden kedves olvasót, hogy jelezzék e-mailben, ha valamiről további információkat szeretnének, vagy csak egyszerűen írják meg véleményüket, ötleteiket, javaslataikat.

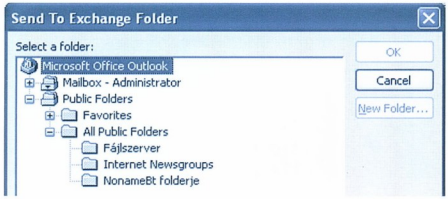
Exchange fájlkiszolgálóként?

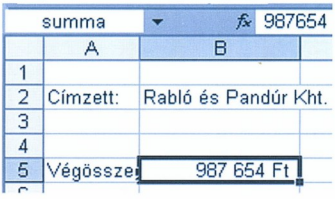
Az alcímben feltett kérdés természetesen csak költői, nem javasolom, hogy az Exchange-et fájlkiszolgálóként használja bárki is, de van néhány érdekes lehetőség, amit ki tudunk használni fájlok nyilvános mappákban történő tárolásakor. A nyilvános mappák egyik gyakori felhasználási területe, hogy olyan Word- és Excel-állományokat tartalmazó üzeneteket tárolunk ott, amelyeket sok felhasználó olvashat. De vajon mindenki tudja-e, hogy közvetlenül is lehet publikálni Office-dokumentumokat nyilvános mappákba?



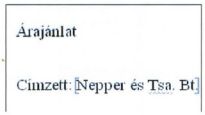
A fenti ábrán a Word 2003 *File* menüjéből elérhető *Send To → Exchange Folder...* parancsa látható, de ugyanez megvan az Excel-ben is. Ha ide kattintunk, akkor kiválaszthatjuk a megfelelő nyilvános mappát, amibe publikálni szeretnénk: A példa kedvéért hívjuk most ezt a mappát „Fájlszerver"-nek. Én mindjárt egy Word-dokumentumot és egy Excel-táblázatot is publikáltam ilyen módon ebbe a nyilvános mappába. Ha Outlookból megnézzük ennek az eredményét, a következőket láthatjuk:

A publikáláskor sajnos nincs lehetőségünk megadni új állományok esetében azok nevét, így a fenti ábrán az alaphelyzet szerinti nevek láthatók. Az azért biztató, hogy az Exchange „tudja", hogy ezek nem közönséges, „post" típusú bejegyzések, hanem egy Excel- és egy Word-állomány, amit az ikonokból is láthatunk. Hogyan lehetne más, ennél több információt látni ezekről az állományokról? Ha megnézzük az Excel-állományomat, akkor kiderül, hogy ez egy sematizált árajánlat, az egyszerűség kedvéért két fontos információval: a címzett és a végösszeg egy-egy cellába van beírva. Nevezzük el ezeket a cellákat: az összeget tartalmazót *summa*, a címzettet tartalmazót *címzett* névvel illettem.

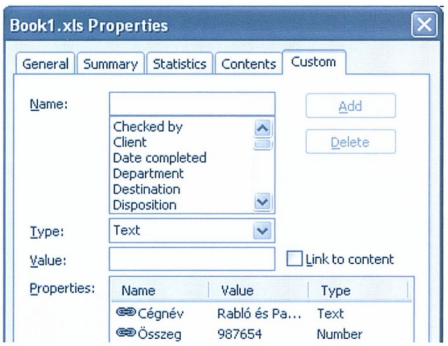




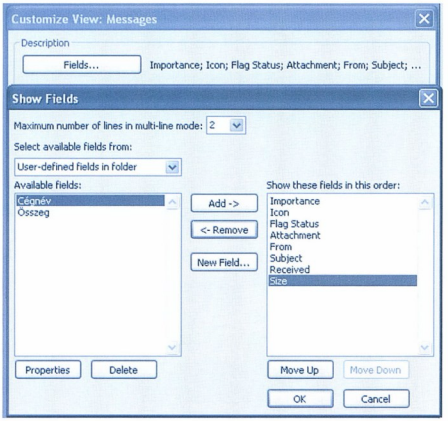
Hasonlóan a Word-állományom is egy árajánlat, ahol szintén megtalálhatók ezek az információk, itt ezekhez *summa* és *címzett* nevű könyvjelzőket rendeltem:



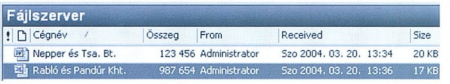
Ezek után mindkét alkalmazásban felveszek egy *Cégnév* és egy *Összeg* nevű tartalomfüggő *(Link to content)* egyedi *(Custom)* tulajdonságot a *File/Properties* menüben, melyek rendre a *címzett* és a *summa* cellához/könyvjelzőhöz vannak kötve:



Ezek az egyedi tulajdonságok – ugyanúgy, mint a SharePoint Portal Servernél – az úgynevezett „property promotion" folyamat során Exchange Store-beli, felhasználó által definiált (User-defined field) tulajdonsággá válnak, amelyeket meg lehet jeleníteni a mappa nézetében:



Így a *Fájlszerver* nevű mappánk végleges Outlook-beli nézetét akár a következő formátumúvá is át tudjuk alakítani:



Azaz ez egy olyan nézet, amelyben az állományokban található információkat anélkül láthatom, hogy megnyitnám azokat: látom a cég nevét, amelynek az ajánlatot küldtem, illetve látom az ajánlat végösszegét is. Ha megnyitom az állományokat, és módosítom ezeket az adatokat, akkor mentés után természetesen ezek frissülnek az Outlook-nézetben is.

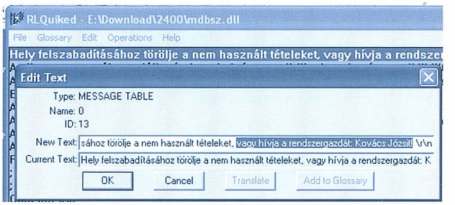
Rendszerüzenetek átírása

Az Exchange kiszolgálók sokfajta üzenetet küldenek a felhasználóknak. Azt, hogy hogyan lehet ezekre az üzenetekre válaszolni, megnéztük az előző számban. Ezek az üzenetek a belső felhasználóknak olyan nyelven érkeznek, amilyen nyelvű a levelesládájuk. A nyelv olyan lesz, amilyen az először rácsatlakozó kliensgép területi beállítása. Előfordulhat, hogy nem tetszik nekünk a felhasználóknak küldött üzenetek megfogalmazása. Például a levelesláda méretének túllépésekor be lehetne tenni a figyelmeztető üzenetbe a megfelelő Help Desk-munkatársunk nevét és telefonszámát, mert nem mindenki ismeri az eredeti üzenetben emlegetett .pst állományok rejtelmeit, és önállóan nem tudna ilyet létrehozni.

A Microsoft hivatalosan nem támogatja a rendszerüzenetek átírását, de megfelelő eszközzel és teszteléssel azért neki lehet veselkedni a feladatnak. Szerencsére az összes üzenet egy MDBSZ.DLL nevű erőforrás-dll-ben van összegyűjtve, így csak ezt kell módosítani. Ehhez eszköz található a

```
ftp://ftp.microsoft.com/Softlib/MSLFILES/
RLTOOLS.EXE
```

helyen. Ez egy önkicsomagoló állomány, ebben a számunkra érdekes segédprogram a RLQUIKED.EXE, mellyel nyelvenként lehet módosítani az üzeneteket. Számunkra a magyar és az angol nyelvű változat érdekes általában.



Mivel mondatonként szerepelnek a bejegyzések, és egy rendszerüzenet több mondatból is állhat, ezért nagyon alaposan le kell tesztelni, hogy mit is írunk át, és az pontosan melyik rendszerüzenetben szerepel. Érdemes az eredeti dll-t elmenteni, hogy hiba esetén visszaállítható legyen. Az átírt üzenetek küldése az Exchange szolgáltatásainak újraindítása után kezdődik.

Ha javítócsomagot teszünk a kiszolgálónkra, akkor ez felülíródhat, így a módosításokat érdemes dokumentálni, hogy újra elvégezhetők legyenek a változtatásaink.

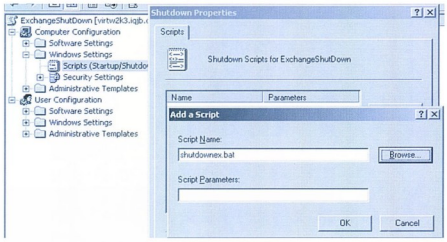
Az Exchange kiszolgáló lassan áll le

Ha az Exchange kiszolgálót tartományvezérlőre telepítjük – ami egyébként nem ajánlott, de például a Small Business Server esetében adottság –, akkor a szerver leállításánál tapasztalhatjuk, hogy az akár 10-15 percig is eltarthat. Ennek oka, hogy a szolgáltatások leállásának sorrendje nem ideális, azaz a Local Security Authority Subsystem szolgáltatás (ezen belül fut az AD) viszonylag gyorsan leáll, míg az Exchange különböző szolgáltatásai lassabban, és mire némelyik, AD-t intenzíven használó Exchange-szolgáltatás leállna, addigra time-out várakozási állapotba kerülnek az AD elérhetetlensége miatt.

Megoldás, hogy készítünk egy leállító parancsfájlt, például:

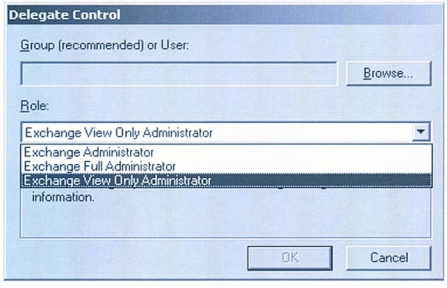
```
net stop MSExchangeMGMT /yes
net stop RESvc /yes
net stop POP3Svc /yes
net stop IMAP4Svc /yes
net stop NntpSvc /yes
net stop SMTPSVC /yes
net stop MSExchangeES /yes
net stop MSExchangeIS /yes
net stop MSExchangeMTA /yes
net stop MSExchangeSA /yes
```

Természetesen a ténylegesen futó szolgáltatásainkat figyelembe véve kell ezt kialakítani. A legjobb, ha ezt a fájlt betesszük az Excange/DC gépünk Shut down scriptjébe, így biztos, hogy nem fogjuk elfelejteni futtatni leállítás előtt. Erre a Group Policy-t tudjuk felhasználni:



Exchange rendszergazda-jogosultságok finomhangolása

Az *Exchange* „beépítetten" háromfajta rendszergazdai szerepet kínál fel az *Exchange System Manager Delegation Wizard*-jával:

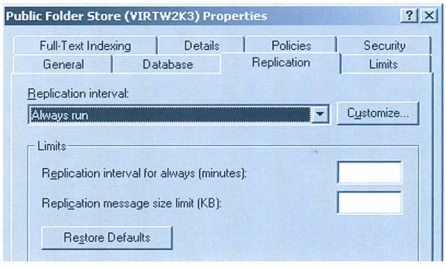


Ez erős leegyszerűsítése a világnak, önmagában csak ezekkel nem nagyon tudjuk megadni például annak a rendszergazdának a jogosultságait, aki csak a nyilvános mappákért felel, vagy azét sem, aki csak a Routing Group-ok kialakításáért és a connector objektumok konfigurálásáért felel. Elvileg belemászhatnánk az AD-be, és adhatnánk jogosultságokat csak bizonyos objektumtípusokra is, de ez elég munkaigényes, és nehezen átlátható eredményhez vezetne. Az egyszerűbb megoldás, hogy kihasználjuk az Administrative Group-ok lehetőségeit, és minden Exchange objektumot, amit csak lehet, szétpakolunk külön AG-kbe. Az alábbi ábra egy ilyen szétcincált állapotot mutat:

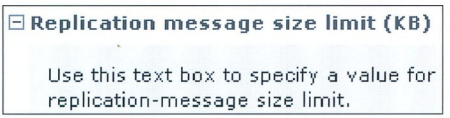
Létrehoztam az egyes AG-objektumokat, majd bennük az üres Folders, Routing Groups, System Policies konténereket. De vajon hogyan kerülnek át ide az eredeti helyükről a mappák, routing goup-ok stb.? Nagyon egyszerű: Cut-Copy eljárással! Az Exchange System Manager rendelkezik ezzel a szövegszerkesztőkben megszokott képességekkel, ki tudjuk vágni az egyik objektumhierarchiát, és be tudjuk illeszteni az új helyére. A régi, üres tároló objektumokat ki tudjuk törölni. Ezek után már az új AG-kre kell a megfelelő rendszergazdai szerepeket kiosztani a szegényes kínálatból, de ez pont az eredetileg elgondolt, objektumtípusra szelektált jogosultságot fogja eredményezni. Természetesen ezt csak natív módú Exchange szervezetben tudjuk megtenni.

„Apróságok"

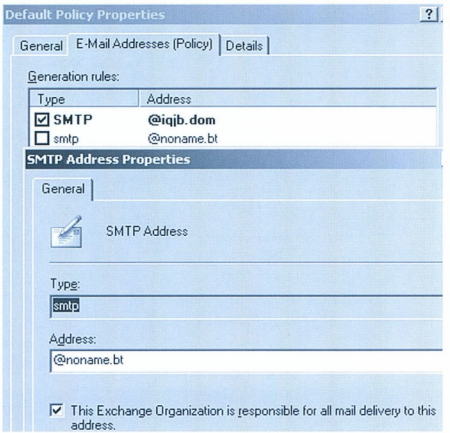
Úgy tapasztaltam, hogy néhány Exchange-beállítási lehetőség értelmezése sok rendszergazdának gondot okoz, mert nem teljesen egyértelmű a beállítás melletti felirat, és a súgó sem. Például a nyilvános mappák adatbázisainak *Replication* tulajdonságlapján van egy *„Replication message size limit (KB)"* nevű attribútum:



Az ember azt hinné, hogy ezzel be lehet állítani, hogy mekkora legyen az a maximális méretű üzenet, amelyet a rendszerünk még hajlandó lesz replikálni.
A súgó sem ad igazán jó eligazítást, idézem:



A valóságban ezzel a paraméterrel azt lehet szabályozni, hogy lehetőleg mekkora legyen a legkisebb replikációs üzenet. Azaz, ha több pici üzenetet kellene replikálni egy replikációs ciklus során, akkor az Exchange inkább összefogja ezeket egy nagyobb üzenetbe, és úgy küldi el a többi kiszolgálónknak. Ez az érték alaphelyzetben 300 KB, de itt módosítani lehet ezt.
A másik félreértésre okot adó beállítási lehetőség a *Recipient Policy*-knál található:



Az előbbi képen a hátrébb található ablakba felvettem egy *„@noname.bt"* címzési sémát a generálási szabályok közé, de nem tettem pipát az előtte található kis négyzetbe. Ekkor, ha megnyitjuk ennek a címnek a tulajdonságlapját, akkor alul a pipa a „This Exchange…." felirat mellett határozottan ott van. Azt hihetnénk, hogy attól még, hogy automatikusan nem generáltatjuk le a mailboxaink számára a „noname.bt" végű e-mail címeket, az Excange-szervezetünk be fogja fogadni a noname.bt-nek küldött leveleket. Az igazság az, hogy sajnos ennek ellenére az Exchange-ünk relaynek fogja értelmezni ezeket a próbálkozásokat. Azaz csak akkor érzi sajátjának a noname.bt-nek küldött leveleket, ha mindkét pipa be van rakva.
Mit tegyünk akkor, ha nem akarjuk automatikusan generáltatni egy új e-mail címet a felhasználóknak, mert manuálisan akarjuk ezeket kialakítani, de mégis szeretnénk, ha az Exchange elfogadná az ilyen címekre küldött leveleket? Ennek egyik módja, hogy külön Recipient Policy-ban definiáljuk ezt az új e-mail címet, és olyan szűrőfeltételt adunk meg, amely garantáltan üres halmazt eredményez.
Végül nézzük a globális beállítások között található „Sender Filtering" szűrőfeltétel egyik beállítását:

Archiválásról beszél, de vajon hova archivál? A súgó szintén nem sokat segít. A válasz az, hogy az archiválás helye az SMTP virtuális szerverek munkakönyvtáraiban létrejövő „filter" könyvtár lesz:

```
<exchange telepítési
partíció>\exchsrvr\mailroot\<vsi#>\filter
```


Soós Tibor
soost@iqjb.hu
IQSOFT - John Bryce Oktatóközpont

Vizsgák és minősítések

Microsoft oktatási tájékoztató

A Microsoft kínálatában több mint negyven féle technikai vizsga van, amelyeket a hivatalos Microsoft oktatóközpontokban lehet letenni. Összeállításunkban részletesen ismertetjük a Microsoft vizsgák és minősítések rendszerét.

A minősítések megszerzéséhez a Microsoft által előírt úgynevezett kötelező (vagy fő, angolul core) és szabadon választható (vagy kiegészítő, angolul elective) vizsgákat kell teljesíteni. Az egyes minősítések többféle vizsgaösszeállítással is megszerezhetőek. A Windows Server 2003 és a Visual Studio .NET megjelenésével a vizsgák száma, valamint az adott minősítések követelményrendszere is megváltozott illetve bővült.

Microsoft termékspecialista (MCP)

Követelmény: a cím elnyeréséhez egy vizsgát kell letenni bármely aktuális Microsoft vizsga közül. Ez tulajdonképpen a bevezető minősítés, erre építve szerezhet magasabb fokozatokat további vizsgák letételével.

Microsoft Windows 2000 rendszer-adminisztrátor (MCSA on Windows 2000)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 szabadon választható vizsgát kell letenni.
Kötelező vizsgák: 070-210, vagy 070-270, 070-215, 070-218
Választható vizsgák: 070-028, 070-086, 070-214, 070-216, 070-224, 070-227, 070-228, 070-244. Választható vizsgák lehetnek még a CompTIA A+ és Network+ vizsgái is.

Microsoft Windows 2000 rendszer-adminisztrátor + biztonság (MCSA + Security)

Követelmény: a cím elnyeréséhez 3 kötelező és 2 speciális, biztonság témájú vizsgát kell letenni.
Kötelező vizsgák: 070-210, vagy 070-270, 070-215, 070-218
Speciális vizsgák: a 070-214 és a 070-227, vagy a CompTIA Security+ vizsgája.

Microsoft Windows 2000 rendszer-adminisztrátor + Messaging (MCSA + Messaging)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 speciális vizsgát kell letenni.
Kötelező vizsgák: 070-210 vagy 070-270, 070-215, 070-218
Speciális vizsgák: 070-224 vagy 070-284

Microsoft Windows 2003 rendszer-adminisztrátor (MCSA on Windows 2003)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 szabadon választható vizsgát kell letenni. A korábbi Windows 2000 alapú MCSA minősítéssel rendelkezők a 070-292-es kódú vizsga letételével frissíthetik minősítésüket MCSA 2003-ra.
Kötelező vizsgák: 070-210, vagy 070-270, 070-290, 070-291
Választható vizsgák: 070-086, 070-227, 070-228, 070-284, 070-299. Választható vizsgák lehetnek még a CompTIA A+, Network+ vagy Server+ vizsgái is.

Microsoft Windows 2000 rendszer-adminisztrátor + Messaging (MCSA + Messaging)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 speciális vizsgát kell letenni.
Kötelező vizsgák: 070-210 vagy 070-270, 070-290, 070-291
Speciális vizsgák: 070-284

Microsoft Windows 2000 rendszermérnök (MCSE on Windows 2000)

Követelmény: a cím elnyeréséhez 5 kötelező és 2 szabadon választható vizsgát kell letenni.
Figyelem! A 070-019, 070-028, 070-029 vizsgák 2004. júniusától megszűnnek!
Kötelező vizsgák: 070-210, vagy 070-270, 070-215, 070-216, 070-217 és a 070-219, 070-220, 70-221, 070-226 közül az egyik.
Választható vizsgák: 070-019, 070-028, 070-029, 070-086, 070-214, 070-218, 070-219, 070-220, 070-221, 070-224, 070-225, 070-226, 070-227, 070-228, 070-229, 070-230, 070-231, 070-232, 070-234, 070-244, amennyiben az nem szerepelt a kötelezőek között. Az azonos témakörű, de különböző verziójú termékhez kapcsolódó vizsgákat egyként tekinti a Microsoft.

Microsoft Windows 2000 rendszermérnök + Messaging (MCSE + Messaging)

Követelmény: a cím elnyeréséhez 5 kötelező és 2 speciális vizsgát kell letenni.
Kötelező vizsgák: 070-210 vagy 070-270, 070-215, 070-216, 070-217 és a 070-219, -220, -221, -226 közül az egyik.
Speciális vizsgák: a 070-224 és a 070-225

Microsoft Windows 2003 rendszermérnök (MCSE on Windows 2003)

Követelmény: a cím elnyeréséhez 6 kötelező és 1 szabadon választható vizsgát kell letenni. A korábbi Windows 2000 alapú MCSE minősítéssel rendelkezők a 070-292-es és 070-296-os kódú vizsga letételével frissíthetik minősítésüket MCSE 2003-ra, mellyel automatikusan a MCSA 2003 címet is elnyerik.

Kötelező vizsgák: 070-210, vagy 070-270, 070-290, 070-291, 070-293, 070-294, és a 070-297 és 070-298 közül az egyik.

Választható vizsgák: 070-086, 070-227, 070-228, 070-229, 070-232, 070-284, 070-297, 070-298, 070-299 közül az egyik, amennyiben az nem szerepelt a kötelezőek között.

Microsoft Windows 2003 rendszermérnök + Messaging (MCSE 2003+Messaging)

Követelmény: a cím elnyeréséhez 6 kötelező és 2 speciális vizsgát kell letenni.

Kötelező vizsgák: 070-210 vagy 070-270, 070-290, 070-291, 070-293 és a 070-297, -298 közül az egyik.

Speciális vizsgák: 070-284 és a 070-285

Microsoft adatbázis-adminisztrátor (MCDBA)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 szabadon választható vizsgát kell letenni.

Figyelem! A 070-015, 070-019, 070-175 vizsgák 2004 júniusától megszűnnek!

Kötelező vizsgák: 070-215, 070-228, 070-229

Választható vizsgák: 070-216, 070-015, 070-019, 070-175, 070-305, 070-306, 070-315, 070-316, 070-310, 070-320.

Microsoft alkalmazásfejlesztő (MCAD)

Követelmény: a cím elnyeréséhez 2 kötelező és 1 szabadon választható vizsgát kell letenni.

Kötelező vizsgák: a 070-305, 070-306, 070-315 és 070-316 közül az egyik és a 070-310, vagy 070-320 közül az egyik.

Választható vizsgák: 070-229, 070-230, 070-234 vagy a 070-305, 070-306, 070-315, 070-316 közül az egyik, amennyiben az nem szerepelt a kötelező vizsgák között.

Microsoft fejlesztőmérnök (MCSD on Visual Studio 6)

Követelmény: a cím elnyeréséhez 3 kötelező és 1 szabadon választható vizsgát kell letenni.

Figyelem! A címhez kapcsolódó vizsgák a 070-229, 070-230 és 070-234 kivételével 2004 júniusától megszűnnek!

Kötelező vizsgák: 070-100, a 070-016 és 070-176 közül az egyik és a 070-015 és 070-175 közül az egyik.

Választható vizsgák: 070-015, 070-016, 070-019, 070-029, 070-152, 070-175, 070-176, 070-229, 070-230, 070-234 közül az egyik, amennyiben az nem szerepelt a kötelező vizsgák között.

Microsoft fejlesztőmérnök (MCSD on Visual Studio .NET)

Követelmény: a cím elnyeréséhez 4 kötelező és 1 szabadon választható vizsgát kell letenni.

Kötelező vizsgák: 070-300, a 070-305, 070-306 közül az egyik, a 070-315, 070-316 közül az egyik, valamint a 070-010, 070-320 közül az egyik.

Választható vizsgák: 070-229, 070-230, 070-234

Felkészülési lehetőségek a vizsgákra

Az egyes vizsgákra való felkészüléshez egyéni és csoportos képzési megoldásokat kínál a Microsoft.

Hivatalos Microsoft oktatóközpontok (MSCTEC) és hivatalos Microsoft tanfolyamok

Az anyag elsajátításának és a vizsgára való felkészülésnek a leghatékonyabb módját a hivatalos Microsoft oktatóközpontokban (Microsoft CTEC) tartott, tapasztalt, minősített Microsoft tréner (MCT) oktató szakemberek által vezetett csoportos tanfolyamok és konzultációk jelentik. Az intenzív kurzusok során az elméleti anyagot folyamatos gyakorlatok és laborok egészítik ki.

Hivatalos Akadémiai Tréningprogram (Microsoft IT Academy Program)

főiskolai, egyetemi oktatási intézményekben, adott hivatalos Microsoft tanfolyami oktatások keretén belül szintén lehetőség van hivatalos Microsoft tanfolyamok hallgatására.

Microsoft Press kiadványok

A legelterjedtebb, legnépszerűbb szakkönyvek az elméleti anyag elsajátításához és a vizsgára való felkészüléshez. A Microsoft Press kiadványai elsősorban önálló tanulásra szánt tréningcsomagok, szakkönyvek, melyek különböző témakörökben, különböző felkészültségű szakemberek részére elméleti és gyakorlati anyagokat, feladatokat, ellenőrző kérdéseket és CD-mellékleten hasznos információkat, teszteket, segédprogramokat is tartalmaznak.

Online források

Számos vizsgafelkészítő könyv és próbateszt is megvásárolható, vagy letölthető a Microsoft és más cégek weboldalairól is az egyes tesztekhez, melyek segítségével felmérheti jelenlegi tudását, és azt, hogy melyek azok a témakörök, melyekben csiszolnia kell még ismereteit. A Microsoft Solution Developer Network (http://msdn.microsoft.com), valamint a Technet (http://technet.microsoft.com) oldalán első kézből származó információhoz juthat a Microsoft valamennyi termékével és technológiájával kapcsolatban.

Hivatalos Microsoft tananyagok (Microsoft Official Curriculum, MOC)

Különböző témakörökben, különböző felkészültségű szakemberek részére, elméleti és gyakorlati anyagokat, feladatokat, ellenőrző kérdéseket és CD-mellékleten hasznos információkat, teszteket tartalmazó hivatalos kiadványok. (Ezek Microsoft partnerek vagy trénerek által, vagy hivatalos Microsoft tanfolyamokon érhetők el.)

VIZSGÁK LISTÁJA

Kód	A vizsga tárgya	Kapcsolódó tanfolyamok kódjai
070-015	Designing and Implementing Distributed Applications with MS Visual C++ 6.0	
070-016	Designing and Implementing Desktop Applications with MS Visual C++ 6.0	
070-019	Designing and Implementing Data Warehouses with Microsoft SQL Server 7.0	
070-028	Administering Microsoft SQL Server 7.0	
070-029	Designing and Implementing Databases with Microsoft SQL Server 7.0	
070-081	Implementing and Supporting Microsoft Exchange Server 5.5	
070-086	Implementing and Supporting Microsoft System Management Server 2.0	827+828
070-100	Analyzing Requirements and Defining Solutions Architectures	
070-152	Designing and Implementing Web Solutions with Microsoft Visual InterDev 6.0	1017
070-175	Designing and Implementing Distributed Applications with Microsoft Visual Basic 6.0	1016
070-176	Designing and Implementing Desktop Applications with Microsoft Visual Basic 6.0	1013
070-210	Installing and Configuring and Administering Windows 2000 Professional	2152/1560
070-214	Implementing and Administering Security in Windows 2000 Environment	2150/2153
070-215	Installing and Configuring and Administering Microsoft Windows 2000 Server	2152/1560
070-216	Implementing and Administering Microsoft Windows 2000 Network Infrastructures	2153/1560
070-217	Implementing and Administering Microsoft Windows 2000 Directory Services Infrastructures	2154/1560
070-218	Managing a Microsoft Windows 2000 Network Environment	2126
070-219	Designing Microsoft Windows 2000 Directory Services Infrastructure	1561
070-220	Designing a Secure Microsoft Windows 2000 Network	2150
070-221	Designing Microsoft Windows 2000 Network Services Infrastructure	562
070-222	Migrating from Microsoft Windows NT 4.0 to Microsoft Windows 2000	2010
070-223	Installing, Configuring, Administering Microsoft Windows 2000 Clustering Services	2087
070-224	Installing, Configuring, Administering Microsoft Exchange 2000 Server	1572
070-225	Designing and Deploying a Messaging Infrastructure with Microsoft Exchange Server 2000	1573, 2355
070-226	Designing Highly Available Web Solutions with Microsoft Windows 2000 Server	2088
070-227	Installing, Configuring, Administering Microsoft ISA Server 2000 Enterprise Edition	2159
070-228	Installing, Configuring, Administering Microsoft SQL Server 2000 Enterprise Edition	2072
070-229	Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition	2073
070-230	Designing and Implementing Solutions with Microsoft BizTalk Server 2000 Enterprise Edition	2379
070-232	Designing Highly Available Web Solutions with Microsoft Windows 2000 and Application Center	2203
070-234	Designing and Implementing Solutions with Microsoft Commerce Server 2000	2185
070-244	Supporting and Maintaning Microsoft Windows NT 4.0 Server Network	
070-270	Installing, Configuring, and Administering Microsoft Windows XP Professional	2272
070-284	Implementing and Managing Microsoft Exchange Server 2003	2400
070-290	Implementing, Managing and Maintaining a Microsoft Windows Server 2003 Environment	2273, 2274, 2275
070-291	Implementing, Managing and Maintaining a Microsoft Windows Server 2003 Network	2276, 2277, 2208
070-292	Upgrading System Administrators Skills from Windows 2000 to Windows Server 2003	2209
070-293	Planning, Implementing and Maintaining a Microsoft Windows Server 2003 Network Infrastructure	2278
070-294	Planning, Implementing and Maintaining a Microsoft Windows Server 2003 Active Directory	2279
070-296	Upgrade System Engineers Skills from Windows 2000 to Windows Server 2003	2210
070-297	Designing a Microsoft Windows Server 2003 Active Directory and Network Infrastructure	2282
070-298	Designing Security for a Microsoft Windows Server 2003 Network	830
070-299	Implementing and Administering Security in a Microsoft Windows Server 2003 Network	előkészületben

070-300	Analyzing Requirements and Defining Solutions Architectures in Microsoft .NET	2710
070-305	Developing Web Applications with Microsoft Visual Basic .NET and Visual Studio .NET	2373, 2310, 2389, 2415
070-306	Developing Windows Applications with Microsoft Visual Basic .NET and Visual Studio .NET	2373, 2565, 2389, 2350
070-310	Developing Web Services and Server Components with Microsoft Visual Basic .NET	2524, 2557, 2389
070-315	Developing Web Applications with Microsoft Visual C# .NET and Visual Studio .NET	2124, 2310, 2389, 2349
070-316	Developing Windows Applications with Microsoft Visual C# .NET and Visual Studio .NET	2349, 2555, 2389, 2350
070-320	Developing Web Services and Server Components with Microsoft Visual C# .NET	2524, 2557, 2389

Figyelem! A szürkével jelzett vizsgák 2004-ben megszűnnek! A részleteket lásd a Microsoft weblapján!

Microsoft információs CD-k (TechNet, MSDN)
Általában havonta megjelenő információs CD-csomagok, melyek a legkülönbözőbb témákban és termékekhez kapcsolódva tartalmaznak támogatási információt, tudnivalót, valamint a legújabb termékeket, szervizcsomagokat és termékbétákat.

Vizsgaközpontok

A Microsoft vizsgákat az oktatóközpontok hivatalos Prometric valamint Pearson VUE vizsgaközpontjaiban teheti le, ahol szabályozott és szigorú követelményrendszernek megfelelő körülmények között vizsgázhat. A vizsgakérdéseket a Microsoft állítja össze, tehát azok összetétele, paraméterei mindkét vizsgaközpont esetében azonosak. A vizsgák során a vizsgaközpont rendjét be kell tartani és szabályzatát el kell fogadni. A vizsgákra az egyes vizsgaközpontok adminisztrátorainál, vizsgaszervezőinél lehet regisztrálni. Ezen kívül lehetőség van online regisztrációra is a Prometric vagy a VUE weboldalán keresztül is a világ bármely központjába, de fontos, hogy ezen esetekben is értesítsék az adott központot. Általános szabály, hogy vizsgázási szándékát, vagy annak lemondását, áttételét 48 órával a vizsga előtt jelezze. Szintén fontos, hogy a vizsgaközponttól kapott vizsgázó azonosítót ne hagyja el, mert minden vizsgaeredménye és adata ezen számhoz lesz/van társítva, és jelentkezésnél erre szüksége van (a vizsgajelentkezési lapon fel kell tüntetni, vagy online jelentkezésnél kell megadni).

A minősítések karbantartása

A vizsga letételét követően annak végeredményét azonnal megtudhatja, és erről a vizsgaközpont által aláírt és lepecsételt okmányt kap. Az első sikeres vizsga esetén, vagyis a Microsoft termékspecialista (MCP) minősítés megszerzésekor e-mailben kapja meg a Microsoft Certified Professional azonosítószámát (MCP ID), valamint egy jelszót, melynek segítségével az MCP-sek számára fenntartott privát weboldalra jelentkezhet fel, illetve készíthet passport-ot. Éppen ezért nagyon fontos, hogy mindig aktív és aktuális e-mail címmel rendelkezzen, mert ezt és az MCP ID-t elsődleges azonosítónak tekinti a Microsoft az ügyintézések során. (**Figyelem!** Az MCP ID nem mindig egyezik meg a vizsgaközpontnál kapott azonosítóval!).
Amennyiben a vizsga letételével egyetemben először egy minősítést is elnyert, akkor a Microsoft az ehhez kapcsolódó okmányokat (oklevél) és kiegészítőket egy csomagocska, ún. Welcome Kit formájában postai úton juttatja el a vizsgázó részére, a jelentkezési lapon megadott címre, körülbelül a vizsgát követő egy hónapon belül. Minden adatát lehetősége van módosítani és karbantartani az MCP-sek privát weboldalán a Microsoft adatbázisában. Ugyanott lehetősége van megtekinteni és kinyomtatni eddig letett vizsgáit és minősítéseit (transcript). Ez igen fontos dokumentum, hiszen ez tartalmazza az aktuális státuszát és vizsgáit, vagyis az oklevelekkel együtt ez a dokumentum igazolja minősítését és annak érvényességét.
A vizsgák természetesen nem örök életűek, vagyis az új szoftververziók, termékek megjelenésével azok frissítése szükséges. Ezeknek az időpontjait és lejáratait a Microsoft határozza meg, és erről az MCP weboldalon, valamint e-mailben tájékoztatja a vizsgázókat. Sok esetben olyan vizsgákat is kihoz és felajánl, melyek segítségével teljes minősítéseket is tud frissíteni, így sokkal kevesebb vizsgát kell újra letennie. (Ilyen például a 070-292 és 070-296-os Windows 2003 minősítésekre frissítő vizsgák.)
A vizsgák megszűnése után azokat letenni már nem lehet, a megszerzett minősítések (a Microsoft jelenlegi állásfoglalása szerint) viszont továbbra is megmaradnak. (Pl. a Windows NT 4.0 minősítések még élnek, sőt ezek még akár a Windows 2003 minősítésekhez is felhasználhatóak). Fontos viszont tudni, hogy a Microsoft az egyéneknek és a cégeknek is csak a legfrissebb és aktív vizsgákkal és minősítésekkel rendelkezők esetében biztosítja a minősítéssel járó előnyöket és lehetőségeket, tehát ezen esetekben érdemes és szükséges megújítani azokat.

További információ:
http://www.microsoft.com/hun/oktaj

A Microsoft-minősítésekhez szükséges vizsgák teljes és friss listája a http://www.microsoft.com/traincert címen található.

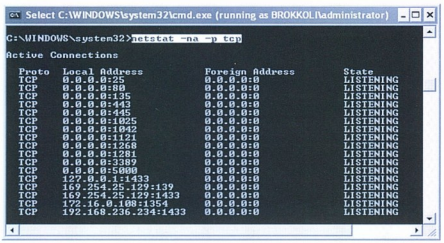
Ha vizsgáival, minősítésével vagy az MCP dokumentumokkal kapcsolatosan kérdése van, az mcphelp@microsoft.com e-mail címre írjon angol nyelvű levelet. Az e-mailben (illetve annak tárgyában) tüntesse fel MCP azonosítószámát, vagy ha ilyennel még nem rendelkezik, akkor a vizsgaközpontnál kapott vizsgázó azonosítóját.

Dr. Watson

Ne higgy a lognak!

Mármint a tűzfalak logjának. Ebben a cikkben alaposan utánajárunk annak a bosszantó jelenségnek, amikor egy tűzfalgazda kiszúrja, hogy egy távoli IP-címről bizonyos TCP-portokat „feszegetnek", vagy netalán port scannelést hajtottak végre. Azt hihetnénk, a támadó személyének vagy helyének felkutatására a legjobb információforrás a tűzfal naplófájlja. Az ottani IP-cím alapján rövid kutatást végezve kiderül, hogy a www.érdektelen.hu webszerver a támadó, ami gazdátlanul lóg a neten, láthatóan ép és egészséges, nincs feltörve. Elkövető tehát nincs. Ez hogy lehet?

Ma már egyre ritkábban fordul elő, hogy egy magányos hős sikeresen feltörne és módosítana egy webhelyet. Nem a hősök lettek butábbak, hanem a szoftverek kódja vált sokkal megbízhatóbbá, jogosultsági rendszere átgondoltabbá – nem beszélve a sokkal képzettebb rendszergazdákról. Talán már nem is reális cél egy-egy weblap lecserélése. Úgy tűnik, a webszerverek read only-vá váltak még a legfelkészültebb hackerek számára is. Ennek tudható be, hogy a hülyegyerek (script kiddie) kora lejárt, helyüket a professzionális adatlopók, ipari kémek vették át. A korkülönbség is jelentős: míg a script kiddie maximum 16 éves, az ipari kém akár húsz is lehet! Még a legjobban védett kiszolgálókról és belső hálózatokról is rengeteg mindent meg lehet tudni pusztán azáltal, hogy furfangos kérdéseket teszünk fel nekik, ők pedig válaszolnak. Az egyik legismertebb információszerzési módszer a port scannelés.



Egy Windows XP nyitott TCP portjai

Portok

Hányas porton fut a webkiszolgáló? 80-as. És az SMTP? 25-ös. De mi az a port? Kikötő, kapu. Bejárat egy távoli számítógép bizonyos szolgáltatása felé. Kétféle port létezik a TCP/IP protokollcsaládban, a kapcsolatorientált TCP (csatorna) és a kapcsolatmentes UDP. Minden számítógép „hallgatózik" bizonyos portokon, várja, hogy valaki megszólítsa. Az alábbi ábra azt mutatja meg, hogy egy hálózati kapcsolattal nem rendelkező, otthoni Windows XP Professional (ez a laptop, amin a cikket írom) mely TCP portokon „figyel". (A NetStat parancsot csak a helyi számítógépen tudjuk használni.) Néhány portot könnyedén és azonnal felismerünk az előbb említetteken kívül, például:

- 135: NetBIOS (Endpoint Mapper)
- 443: SSL
- 445: Windows fájl- és nyomtatómegosztás (CIFS)
- 1433: MS SQL Server
- 3389: Terminal Services

Sokat tudtunk meg ezáltal a számítógépről? Majdnem mindent! Ez egészen biztosan Windows, már csak a termék altípusa és verziószáma kétséges. Ez utóbbi információk kiderítésére speciális szerszámot szoktak használni, a port scannert. Egy jó port scanner ugyanis nemcsak azokat az adatokat dolgozza fel a válaszokból, amiket könnyű kiolvasni (nyitott port itt és ott), hanem azt is, amit nehéz. Pontosan milyen TCP-válasz érkezett? Melyik mezők voltak benne kitöltve? Használt-e bizonyos TCP-opciókat a feladó? Ezek a jellegzetességek ugyanis egy adott TCP-implementációra vezethetők vissza, egyes esetekben gyártó/termék/verzió-specifikusak. Ezt nevezik OS (Operating System) fingerprintnek, az adott TCP-protokoll ujjlenyomatának.

Port Scan

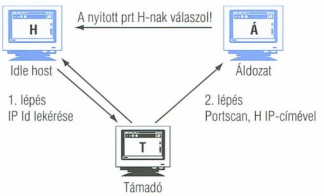
Móricka úgy képzeli, hogy nekiesünk 0-tól 65535-ig az összes TCP és UDP portnak, és megpróbálunk kapcsolatot létesíteni a távoli géppel. Ha a port nyitott, a kapcsolat sikeresen felépül. Ha nem – nem. Ez a „stratégia" azonban több sebből vérzik.

- Teljesen felesleges az összes portot kipróbálni, elég az úgynevezett well-known (közismert) portokat, az 1024-ig terjedő tartományt. Ezáltal lényegesen kevesebb próbálkozás elegendő. Közel ezer port nyitása azonban gyanús, a legtöbb tűzfal már jelenti is a gazdájának, hogy valaki „ante portas".
- Egy Móricától kiinduló portkeresés tulajdonképpen kétirányú felderítés: Móricka megtudja a nyitott portokat, a támadott gép pedig megtudja, ki is Móricka.

Vitathatatlan előny azonban, hogy ezt a kereséstípust tetszőlegesen gyenge jogosultsággal el lehet végezni, hisz nincs benne semmi különleges: Móricka távoli gépekre csatlakozik. Ennyi.
A modern port scannerek rengeteg kifinomult eljárást képesek használni a felderítést végző személyének és IP-címének elrejtésére. Egyik lehetséges módszer, hogy érvénytelen TCP-flag kombinációkat állítanak be a kiküldött csomagban, ezáltal az áldozat megtéved, és nem veszi a portkeresést annak, ami. (Furfangosan megkomponált TCP-csomagok előállítására és kiküldésére azonban már csak egy rendszergazda képes.) Ilyen például a Xmas Tree (F-U-P flagbitek beállítva), a Null Scan (egyetlen flagbit sincs beállítva) stb. De még mindig ott tartunk, hogy Móricka a saját gépét használja scannelésre, tehát jó eséllyel lebukik.

Idle Scan

A most elemzésre kerülő eljárás, az Idle Scan arra való, hogy egy viszonylag alig használt (Idle) ártatlan áldozat számítógépet bízzunk meg a portscanneléssel. Ami történni fog, sima TCP és IP, tehát nem törünk fel semmit, ennek ellenére tetszőleges Internetes számítógépet kinevezhetünk áldozatnak.



Az Idle Scan kezdeti lépései

Először vegyük szemügyre az IP protokollt. Ennek adatstruktúrájában a feladó és a célgép IP-címén kívül egy folyamatosan növekvő számot, az Identification mezőt figyelhetjük meg. Bármelyik IP-implementációt nézzük is, azt tapasztaljuk, hogy minden kibocsátott IP-csomagban ez a szám folyamatosan növekszik. Most vegyük az alábbi helyzetet:
Első lépésként a T támadó valamilyen adatot kér a H hosttól (mondjuk egy weblapot), de csupán azért, hogy hozzájusson annak jelenlegi IP Identification értékéhez. Majd hamis IP-címmel (H IP-címével) nekiáll portot scannelni az Á Áldozaton, ahol a logban H jelenik meg támadóként. Egy apró baj van csupán, hogy Á válaszai nem T-hez futnak be, mert ez már csak így van a hamisított IP-címekkel.
Tehát van már valaki, aki tudja Á nyitott portjait, de az nem a Támadó, hanem a H host.
Ha megvizsgáljuk, hogyan is reagál az Áldozat a portscannelésre, kiderül, hogy mást válaszol H-nak, ha nyitott portot találtunk el, és mást, ha zártat. A nyitott portokra eljuttatott TCP (Syn) csomagra egy (Syn,Ack) válasz érkezik, ezzel jelzi a szerver, hogy „jöhetsz barátom, nyitva az ajtó!". Ha azonban csukott porton próbálunk bemenni, kétféle „válasz" is várható. Egy normális operációs rendszer egy TCP (Rst) csomaggal válaszol (reset connection), míg egy tűzfal egész egyszerűen csöndben marad (nem válaszol). Nomármost jelen helyzetben ezek a válaszok Á-hoz futnak be, mit mondjak, váratlanul. Vizsgáljuk meg az eseteket!

2. Nyitott port (Syn,Ack). Ön mit szólna, ha egyszer csak egy őrült odaszaladna magához, és azt mondaná váratlanul, minden előzmény nélkül, hogy (Syn,Ack) azaz „gyere bébi, fogjuk meg egymás kezét"? Én elküldeném melegebb éghajlatra. H tehát elküldi Á-t melegebb éghajlatra (Rst). Nem mellesleg ezzel megnöveli az IP Identification számlálóját, hisz minden kiküldött csomag esetén növeli eggyel.
3. Csukott port (csend, vagy Rst). Ha H nem kap semmit, nem is válaszol semmit. Szerencsére szintén nem válaszol egy „eltévedt" (Rst) csomagra sem.

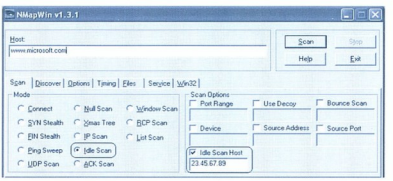
Foglaljuk össze:

- Ha nyitott portot találtunk el Á-n, az IP Id mező eggyel növekedett H-n
- Ha zárt portot kérdeztünk Á-n, az IP Id változatlan maradt H-n

Már csak le kell kérdeznünk H-ról az IP Id aktuális értékét. Ha eggyel nagyobbat kapunk, mint először, az annyit jelent, hogy csukott portot tapogattunk Á-n, mivel a +1 a mi válaszcsomagunk Id-je. Ha kettővel növelt értéket kapunk, Á-n nyitott portot találtunk el, oda is küldött csomagot H.
Most már azt is látjuk, miért kell egy viszonylag csendes, senki által nem látogatott host ehhez a keresési eljáráshoz. Ha valaki egy népszerű, napi tízezer találatot kapó webhelyet próbál H-ként használni, az IP Id értéke nem marad veszteg, hanem folyamatosan pörög. (Csendes gépet könnyebb találni, mint terheltet!)

NMapWin

Az Idle Scan kivitelezéséhez kész eszközöket találunk a neten. A www.insecure.org-ról letölthető, egyébként linuxos NMap-ból már van Windowsos változat, az NMapWin. Ezzel gyerekjáték az Idle Scan kivitelezése, aki tud választani egy opciót és be tud pipálni egy jelölőnégyzetet, annak ez nem okozhat gondot.



Ezerféle Port Scan az NMapban, köztük az Idle Scan

A maradék 10-12 portkeresési üzemmód megismerése házi feladat!

FÖTI MARCELL
marcellf@netacademia.net
MCT, MCSE, MCDBA, MZ/X

MI MÁR LÁTJUK,

ahogy a következő NAGY ÖTLET megszületik.

Egy fejlesztőnek az ötlet már önmagában siker. Épp ilyen fontos, hogy ezek az ötletek a mindennapi életben is megvalósuljanak. Ezért teszünk meg mindent, hogy a fejlesztők kezébe olyan szoftvereket adjunk, amelyekkel megvalósíthatják elképzeléseiket. Az ötleteket, amelyekkel később mindenki nyer.
Neked lehetőség. Nekünk kihívás.™

Your potential. Our passion.™

Microsoft®