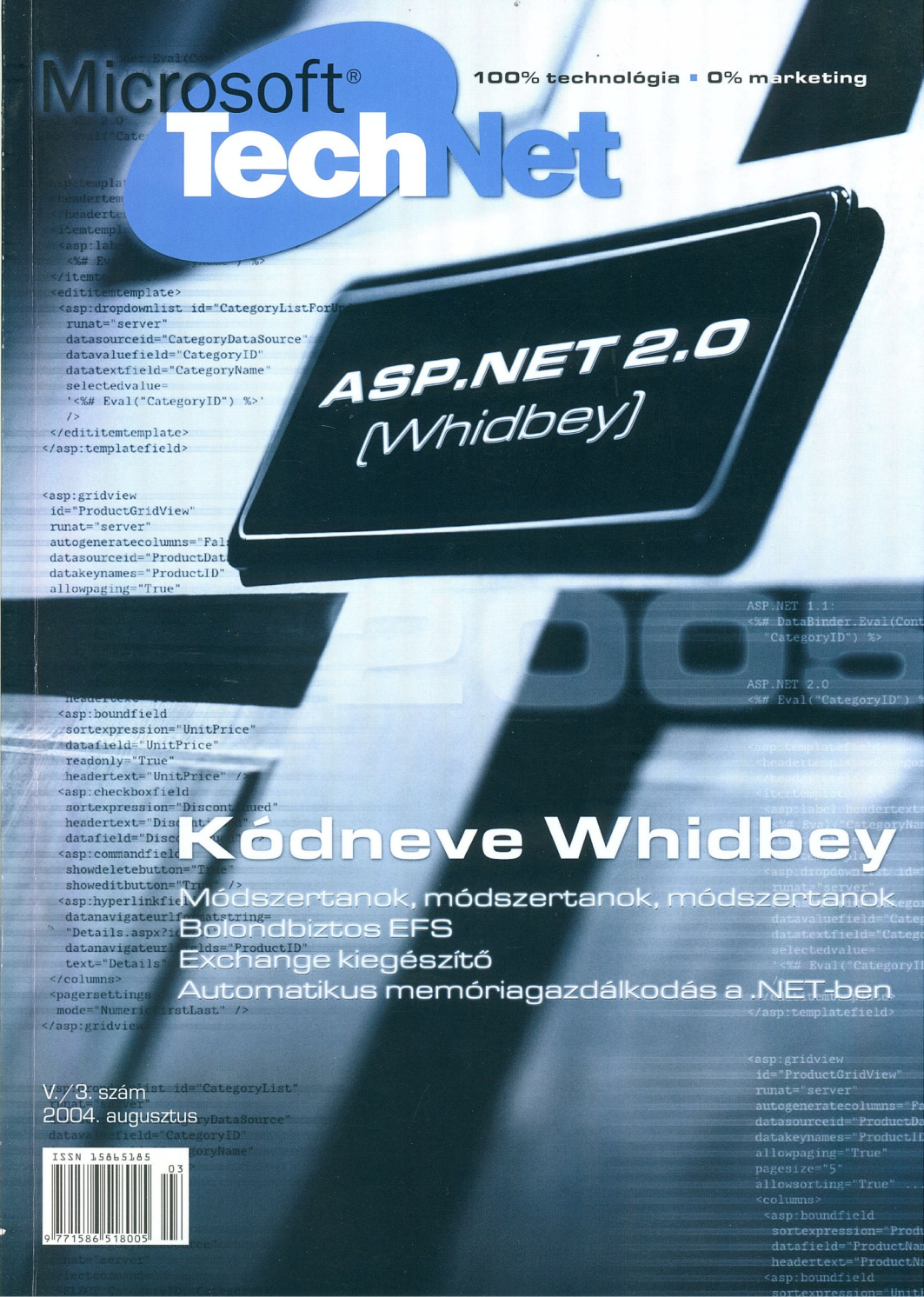
Microsoft® TechNet

100% technológia ▪ 0% marketing



Kódneve Whidbey

Módszertanok, módszertanok, módszertanok

Bolondbiztos EFS

Exchange kiegészítő

Automatikus memóriagazdálkodás a .NET-ben

V./3. szám
2004. augusztus

ISSN 15865185 03
9 771586 518005

Legyen Ön is a NAGYok között!

Microsoft GOLD CERTIFIED Partner | Learning Solutions

Szerezze meg Ön is a hivatalos, nemzetközileg elismert Microsoft minősítések valamelyikét! Új konstrukciók, új képzések, kedvezőbb árak!

A SZÁMALK Továbbképzés a Microsoft első magyarországi aranyfokozatú partnere az oktatási megoldásokban

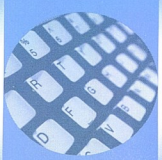
Válassza ki az Önnek megfelelő minősítést és képzési konstrukciót!

Microsoft rendszeradminisztrátor (MCSA) képzés *80 óra*
már nettó 249.000 Ft / főtől
A kisebb informatikai rendszereket és hálózatokat üzemeltető szakemberek számára ajánljuk, akiknek feladatuk lesz Windows 2003 alapú hálózatok telepítése, konfigurálása, adminisztrálása, karbantartása.
Windows XP tréning Kit-tel *Következő kezdési időpont: szeptember 27.*

Microsoft rendszermérnök (MCSE) képzés *160 óra*
már nettó 499.000 Ft / főtől
Nagy, összetett informatikai rendszereket és hálózatokat üzemeltető szakemberek képzése, akiknek feladatuk lesz Windows 2003 alapú hálózatok teljes körű adminisztrálása, támogatása, tervezése.
Windows XP tréning Kit-tel *Következő kezdési időpont: szeptember 27.*

Microsoft adatbázis-adminisztrátor (MCDBA) képzés *156 óra*
már nettó 459.000 Ft / főtől
Microsoft SQL Server 2000/2005 alapú adatbázisrendszerek teljes körű üzemeltetésével, támogatásával, karbantartásával, tervezésével és implementálásával megbízott szakemberek részére ajánlott képzés.
Következő kezdési időpont: szeptember 27.

Microsoft .NET alkalmazásfejlesztő képzés *184 óra*
már nettó 519.000 Ft / főtől
Kezdő fejlesztők, programozók számára ajánlott képzés, akiknek feladatuk lesz Windows és webes alkalmazások készítése, szolgáltatások fejlesztése Visual Studio .NET segítségével.
Következő kezdési időpont: szeptember 27.



Rugalmas időbeosztás, különböző tanfolyamokat és segédanyagokat, vizsgakedvezményeket tartalmazó oktatási csomagok, további (15.000-25.000 Ft értékű) oktatási kedvezmények.

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

SZÁMALK

www.szamalk.hu/tisza

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

Microsoft TechNet

TechNet Magazin
V. évfolyam, 3. szám
2004. augusztus

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.)
Pataki Bernadett

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

2007-ben jön a Longhorn

Nézzünk egy kicsit a jövőbe. A Microsoft közzétette a Windows Server termékcsaláddal kapcsolatos fejlesztéseit, tervezett termékbejelentéseit. Összegezzük, mikor milyen újdonságra számíthatunk.

A Windows Server 2003 SP1 megjelenése 2004 év végére várható. Az SP1 a legújabb biztonsági frissítéseket, javításokat fogja tartalmazni, és olyan szolgáltatásokat foglal magában, melyekkel növelhető a rendszerek biztonsága (ezek megtalálhatók a Windows XP SP2-ben is), továbbá egy „Security Configuration Wizard" varázslót is tartalmaz majd. Az SP1 telepítésekor a tűzfal már alapbeállításként is be lesz kapcsolva. A Windows Server 2003 SP1 még nagyobb teljesítményt és rendelkezésre állási időt is garantál.

A Windows Server 2003, 64-bit for Extended Systems az SP1 csomaggal egy időben jelenik meg, 2004 végén. Ez a teljesen különálló operációs rendszer, a jelenlegi Windows Server 2003 64-bit Edition követő terméke lesz, mely már az olyan újonnan bejelentett 64-bites technológiákat is támogatja, mint például az AMD Opteron vagy Intel Xeon64 processzorok.

2005 második felében jelenik meg a Windows Server 2003 Release2, mely a jelenlegi Windows Server 2003-at hivatott felváltani. A tavaly áprilisban bemutatott Windows Server 2003 verzió megjelenése óta megjelent vagy kiegészített szolgáltatásokon (például Windows SharePoint Services, Rights Managment Services, Automated Deployment Services, Services For Unix 3.5) túl további új szolgáltatásokat, fejlesztéseket tartalmaz majd a Windows Server 2003 R2. A Microsoft véleménye szerint ugyanis egy új, frissített termék sokkal könnyebben használható, mintha külön kellene ezeket a frissítéseket és értéknövelő szolgáltatásokat telepíteni a rendszerre. Az R2 komponensei már tartalmazzák a Windows Server 2003 SP1-et, és hangsúlyos a biztonságos és folyamatos munkahelytől távoli munka támogatása is. Az R2 továbbfejlesztett és hatékonyabb kiszolgáló beüzemelést és felügyeletet tesz lehetővé. Egyébként ugyanazon feltételekkel lehet majd hozzájutni, mint a Windows Server 2003-hoz. A frissítési garanciával rendelkező ügyfelek (SA/EA) természetesen ingyenesen kapják meg.

A Windows Server „Longhorn" változat várhatóan 2007-ben jelenik meg. A termék új generációs webalkalmazás platformot, központosított és egyszerűbb alkalmazásplatform menedzsmentet, valamint teljesen, alapjaitól újratervezett rendszermenedzsmentet, gyorsabb és automatikusabb rendszerfelügyeletet tesz lehetővé. A rendszer az újabb hardvereket és szabványokat is támogatja majd, illetve továbbfejlesztik a közismert alapszolgáltatásokat is. A termék első béta változata a jövő évben várható.

További információ:

www.microsoft.com/hun/

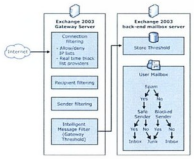
Tartalom

Üzletiintelligencia-szoftverek

MS SQL 2000 Reporting Services III. 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).

Ami a hivatalos Microsoft tanfolyamokból kimaradt...



Exchange 2000/Exchange Server 2003 III. rész

Exchange témában egyelőre ez az utolsó cikk ebben a sorozatban. További apróságok mellett egy új kiegészítő komponenst és ennek a „kiegészítésnek a kiegészítését" ismertetjük.

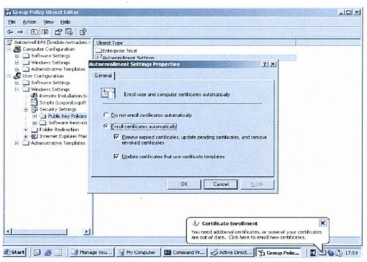
Microsoft CRM v 1.2 — 1. rész

A Microsoft első ASP.NET alapú üzleti alkalmazása

Szeptemberben jelenik meg Magyarországon a Microsoft ügyfélkapcsolat-kezelési rendszere, a Microsoft Customer Relationship Management (CRM) 1.2 lokalizált változata. Cikksorozatunk első része rövid áttekintést nyújt a rendszer szolgáltatásairól és az alkalmazott technológiákról. Leírásunk természetesen nem teljeskörű, néhány példán keresztül mutatjuk be az általunk leginkább érdekesnek ítélt funkciókat.

Dr. Watson

Bolondbiztos EFS



Ebben a cikkben nem az EFS (Encrypting File System) működéséről, vagy használatáról olvashatnak, hanem arról, hogyan tehető az EFS-rendszer bolondbiztossá, hogyan növelhető a megbízhatósága a titkosított fájlok kibontásához szükséges magánkulcsok központi mentésével. Kiválasztott vállalatunknál Windows Server 2003-as tartománykörnyezetet feltételezünk, amelyben egy Enterprise (azaz Active Directoryval integrált) CA Server is van.

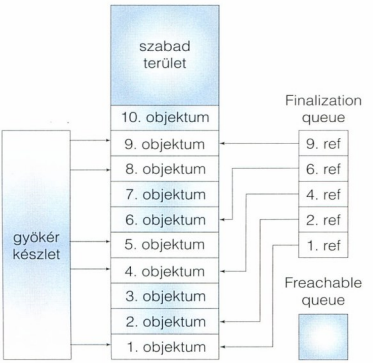
Automatikus memóriagazdálkodás a .NET-ben 2. rész

Finalizáció és teljesítményoptimalizálás

Folytatjuk a GC működésének felderítését. Megismerkedünk az „újraéleszthető" objektumokkal, a gyenge referenciákkal, az objektumgenerációkkal és a GC valós idejű monitorozására szolgáló eszközökkel. Megismertetjük továbbá az erőforrások felszabadítását végző Dispose és Close metódusok helyes megvalósításának módját is.

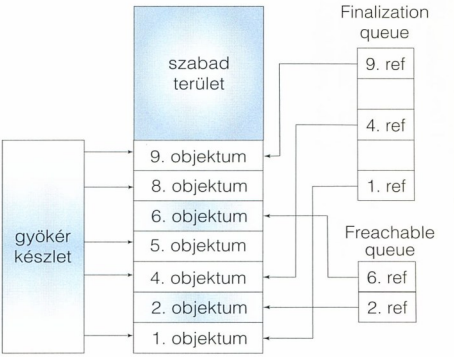
A Finalize metódusok futtatásának részletei

Az első részben leírtak alapján bárkiben joggal merülhet fel a gyanú, hogy valami nincs egészen rendben a Finalize metódusok meghívása körül. Azt mondtuk ugyanis, hogy a GC előbb memóriaszemétnek minősít egy objektumot (vagyis megállapítja, hogy egyetlen referencia sem mutat rá), majd ezután meghívja a referencia nélküli objektum egyik metódusát. Hogyan lehetséges ez? A történet már az objektum létrehozásakor kezdődik. Új objektum létrehozásakor, ha annak osztályában van Finalize metódus, az objektum referenciája bekerül a GC által erre a célra fenntartott sorba (Finalization queue). A sor elemei tehát azokra az objektumokra mutatnak, amelyek Finalize metódusát megszüntetésük előtt meg kell hívni.



A Finalization queue

Az ábrán látható 1., 2., 4., 6., 9. objektum tartalmaz Finalize metódust, tehát referenciáik felkerülnek a Finalization queue-ba. Amikor a GC akcióba lép, a 2., 3., 6., 7. és 10. objektumra nem talál hivatkozást, tehát szemétnek minősíti azokat. Ezután végignézi a Finalization queue-t, és ha talál az eltávolítandó objektumokra mutató hivatkozást, átmozgatja azokat a Freachable queue-ba. A Freachable queue szintén a GC által fenntartott adatstruktúra, amely a Finalize metódus futtatására előkészített objektumok referenciáit tartalmazza. A begyűjtés után a heap az alábbi ábrán látható képet mutatja:



A heap az első begyűjtés után

A 3. 7. és 10. objektum memóriaterülete felszabadult, mivel nem volt meghívandó Finalize metódusuk. A 2. és 6. objektum azonban továbbra is a memóriában maradt, referenciájuk pedig a Freachable queue-ban tárolódik.

A Finalize metódusok meghívását önálló, erre a célra fenntartott programszál végzi, amely akkor aktivizálódik, ha a Freachable queue-ban bejegyzések jelennek meg. A mintaprogramban **[1]** is ellenőrizhetjük ezt, ha *A fő szál hash kódja* menüponttal kapott értéket, és a Finalize metódusokban kiírt hash-értékeket összehasonlítjuk. Az értékek nem egyformák, tehát a Finalize metódusok külön szálon futnak. Az aktuális szál hash kódját a

```
System.Threading.Thread.CurrentThread.
↳ GetHashCode()
```

függvényhívás adja vissza. Érdekes jelenséget figyelhetünk meg, ha a mintaprogram WaitPendingFinalizers metódusából kivesszük a GC.WaitForPendingFinalizers(); sort.

```
static void WaitPendingFinalizers() {
    Console.WriteLine("A Finalize metódusok
↳   meghívása elindult.");
    //GC.WaitForPendingFinalizers();
    Console.WriteLine("A Finalize metódusok
↳   lefutottak.");
}
```

A metódushívás funkciója az, hogy blokkolja az adott szál futását, amíg a Finalize metódusokat futtató programszál elvégzi feladatát. Ha így futtatjuk a *Szemetelés*, majd a *Szemétgyűjtés* menüpontokat, a képernyőre írt üzenetek helyes sorrendje teljesen felborul a párhuzamosan futó szálak miatt. A Finalize metódus lefutása után az adott objektum referenciája törlődik a Freachable queue-ból, így már valóban nincsen hozzá élő referencia. A GC következő futásakor tehát fel fogja szabadítani az objektumhoz tartozó memóriaterületet.
Tekintsük át még egyszer, milyen eseményekből áll egy Finalize metódust tartalmazó objektum élete:

- Az objektum születésekor referencia kerül a Finalization queue-ba.
- Ha a GC futásakor a programunk nem tartalmaz élő referenciát az objektumra, akkor ez a referencia átkerül a Freachable queue-ba.
- A Finalize metódusokat végrehajtó speciális programszál a Freachable queue-ban lévő referencia alapján meghívja az objektum Finalize metódusát, és törli a sorból a referenciát.
- A következő GC-futás felszabadítja az objektum memóriaterületét.

A mintaprogram Finalization queue menüpontja segítségével megfigyelhetjük a GC osztály két statikus metódusának hatását. A *GC.ReRegisterForFinalize(obj)* metódus segítségével objektumreferenciákat adhatunk a Finalization sorhoz, a *GC.SuppressFinalize(obj)* metódussal pedig a paraméterként átadott referenciát eltávolíthatjuk a sorból.

```
static void Final() {
    Szemet obj = new Szemet("0");
    GC.ReRegisterForFinalize(obj);
    GC.ReRegisterForFinalize(obj);
    Console.WriteLine("Az objektum két újabb
↳   referenciát adjuk a Finalization
↳   queue-hoz");
      obj = null;
    Collect();
    WaitPendingFinalizers();
    Console.WriteLine("A Finalize metódus
↳   háromszor futott le.");
    Console.WriteLine("-----");
    obj = new Szemet("0");
    GC.SuppressFinalize(obj);
    Console.WriteLine("Az objektum referenciáját
↳   eltávolítjuk a Finalization queue-ból");
    obj=null;
    Collect();
    WaitPendingFinalizers();
    Console.WriteLine("A Finalize metódus
↳   egyszer sem futott le.");
}
```

Ha egy adott objektum referenciája többször szerepel a sorban, akkor Finalize metódusa is többször fog lefutni (normál alkalmazás esetén ez természetesen nem megfelelő viselkedés), ha pedig eltávolítjuk a referenciát, a metódus nem fut le egyáltalán.

Objektumok újraélesztése

Egy objektum újraélesztése egyszerűen azt jelenti, hogy annak Finalize metódusában az objektum referenciáját elhelyezzük egy olyan változóban, amely később az alkalmazásunkból is elérhető. Amikor az objektum Finalize metódusa elindul, élő referencia csak a Freachable sorban található, de a metódus lefutása után az objektum már az alkalmazásból is elérhetővé válik, így a GC nem fogja megszüntetni azt. Az alkalmazás ekkor újra elérheti az objektum tagváltozóit, és meghívhatja annak metódusait. Az újraélesztheto objektum osztályának kódja a következő:

```
class UjraeleszthetoObjektum:AlapObjektum {
    public string eletjel;
    public UjraeleszthetoObjektum(string s)
↳   :base(s) {
    this.nev=s;
    Console.WriteLine("Az "+nev+" objektum
↳   konstruktora fut.");
    eletjel=s+": még a memóriában vagyok!";
    }
    ~UjraeleszthetoObjektum() {
    Console.WriteLine("Az "+nev+" objektum
↳   Finalize metódusa fut.");
    App.ObjRef=this;
    }
}
```

Az *App.ObjRef* az alkalmazás másik osztályában definiált *UjraeleszthetoObjektum* típusú statikus változó. Az objektummal együtt a belőle hivatkozott objektumok mindegyike is új életre kel. Mindenképpen figyelembe kell azonban vennünk, hogy az ilyen módon újraélesztett objektumok Finalize metódusa már lefutott, így az objektum használata váratlan jelenségeket okozhat.
Ha az újraélesztheto objektum Finalize metódusába még a *GC.ReRegisterForFinalize(this);* hívást is beillesztjük, létrehoztuk az „elpusztíthatatlan" objektumot.

Dispose és Close metódusok használata

A cikk első részében láthattuk, hogy a Finalize metódusok nem minden esetben alkalmasak arra, hogy az erőforrások felszabadítását a kellő időben és sorrendben elvégezzék. A .NET konvenciói szerint az erőforrások felszabadítását az IDisposable interfészben definiált Dispose metódus megvalósításával végezhetjük el. A Dispose metódus helyes megvalósításához a következő szempontokat kell figyelembe vennünk:

- az összes erőforrás megfelelő felszabadítása történjen meg a metódus meghívásakor – tehát a metódus hívja meg az összes tartalmazott objektum és az ősosztály (ha az implementálja az IDisposable interfészt) Dispose metódusát,
- ne okozzon problémát a metódus ismételt meghívása – tehát tárolnunk kell a meghívás tényét, másodszor már semmi tennivalónk nincs,
- ne okozzon (túl nagy) problémát a metódus meghívásának elmulasztása – tehát a GC által futtatott Finalize metódus hívja meg a Dispose-t.
- Ha a Finalize hívja meg a Dispose-t, akkor menedzselt objektumok Dispose metódusait nem hívhatjuk, mert a Finalize metódusok hívásának sorrendjét nem ismerjük, vagyis lehet, hogy azok Finalize metódusa már korábban lefutott – tehát meg kell különböztetnünk, hogy a Dispose hívása explicit (a kódból) vagy implicit (a Finalize metódusból) módon történt.
- Ha meghívtuk a Dispose metódust, akkor a memória felszabadításával ne kelljen megvárni az ekkor már szükségtelen Finalize futását – tehát a Dispose távolítsa el az objektum referenciáját a Finalization queue-ból.

A következő osztályséma megfelel ezeknek a szempontoknak, és lehetővé teszi a belőle létrehozott objektumok biztonságos felszabadítását.

```
//a DisposeObjektum leszármazottja lesz majd
class DisposeAlap:AlapObjektum, IDisposable {
//tárolja, hogy lefutott-e már a dispose
   protected bool disposed = false
   public DisposeAlap() {
     Console.WriteLine("A DisposeAlap
     osztályban megadott konstruktor fut.");
   }
   public void Dispose() {
   Dispose(true);
   //ezután már nem kell Finalize
   GC.SuppressFinalize(this);
   }
   protected virtual void Dispose(bool
   disposing){
   if(!this.disposed) {
     //explicit hívás a kódból
     if(disposing){
       Console.WriteLine("DisposeAlap: A
       menedzselt erôforrások dispose
       metódusainak hívása.");
     }
     //ha a Finalize hívja, csak ez fut le
     Console.WriteLine("DisposeAlap: A nem
     menedzselt erôforrások felszabadítása.");
   }
   disposed = true;
   }
   ~DisposeAlap() {
     Dispose(false);
   }
   //dispose utáni hívásakor kivétel keletkezik
   public void HasznosMetodus(){
     if(this.disposed){
       throw new ObjectDisposedException(
       this.nev,"Már lefutott a dispose
       metódus!");
     }//if
   }    HasznosMetodus
}      //class
```

Tekintsük át az osztály kódját:

Az osztály implementálja az IDisposable interfészt, tehát meg kell valósítania a Dispose() metódust. Az utód osztályok azonban már nem ezt, hanem a Dispose(bool disposing) metódust írhatják felül. Ez a metódus két alapvetően eltérő szituációban hívódhat meg. Ha a kódból meghívjuk az objektum Dispose() metódusát, akkor a hívás true paraméterrel történik, ekkor, ha korábban még nem hívtuk meg a Dispose()-t (ezt a disposed tagváltozó tárolja), megtörténik a menedzselt és nem menedzselt erőforrások felszabadítása is. Ha azonban a Finalize hívja meg a metódust, false paramétert ad át, ekkor csak a nem menedzselt erőforrásokat szabadíthatjuk fel, mivel a Finalize metódusban nem ajánlatos más objektumokra hivatkozni.

Ha az osztályban más metódusokat (pl. HasznosMetudus) is definiálunk, azokban ellenőriznünk kell a *disposed* változó értékét. Ha az érték *true*, kivétel dobásával kell jeleznünk ezt a hívónak.

Az osztályból örökített további osztályok sémája a következő lehet:

```
class DisposeObjektum : DisposeAlap {
   public DisposeObjektum(String s) {
     this.nev=s;
     Console.WriteLine("A "+nev+" objektum
     konstuktora fut.");
   }
   protected override void Dispose(bool
   disposing) {
     if(!this.disposed) {
     try {
       Console.WriteLine("A "+nev+" objektum
       Dispose metódusa fut.");
       if(disposing) {
         Console.WriteLine("DisposeObjektum: A
         menedzselt erôforrások dispose
         metódusainak hívása.");
       }//disposing
       Console.WriteLine("DisposeObjektum: A nem
       menedzselt erôforrások felszabadítása.");
       this.disposed = true;
     }//try
     finally {
       base.Dispose(disposing);
     }//finally
     }//if
   }    //Dispose()
}       //class
```

Az osztály a paraméteres Dispose metódust írja felül (a másikat nem is lehet), a kódba be kell illeszteni az ősosztályban még nem használt erőforrások felszabadítását is. Természetesen meg kell hívnunk az alaposztály Dispose metódusát is. Paraméter nélküli Dispose és Finalize metódus-

ra nincsen szükség, ezekből megfelel az alaposztálytól örökölt változat is.
A mintaprogramban ebből az osztályból hozunk létre példányokat, hogy megfigyelhessük a Dispose explicit és implicit (a Finalize metódusból) meghívásának hatását.
A konvenciók szerint az objektumok felszabadítását elvégző metódus neve Close abban az esetben, ha olyan erőforrásról van szó, amelyet a lezárás után szükség esetén újra megnyithatunk (jellemzően Open metódussal). Ilyen esetben az alaposztálynak tartalmaznia kell egy publikus (de nem felülírható) Close metódust, amelyet általában a paraméter nélküli Dispose is meghív.

Gyenge referenciák (weak references)

Ha alkalmazásunk élő hivatkozást tartalmaz valamely objektumra, a GC természetesen meg fogja találni a hivatkozást, és érintetlenül hagyja az adott objektumot. Az ilyen hivatkozást erős referenciának (strong reference) nevezzük. Ezzel ellentétben, a gyenge referenciával hivatkozott objektumok által elfoglalt memória a GC futásakor felszabadul ugyan, addig azonban az objektumok elérhetőek maradnak az alkalmazás számára. Ha az alkalmazás használni szeretne egy gyenge referenciával hivatkozott objektumot, erős referenciát kell létrehoznia hozzá. Ha ez még a GC futása előtt történik, az objektum továbbra is elérhető marad. Mikor lehet erre szükség? Például akkor, ha alkalmazásunkban könnyen rekonstruálható, de sok memóriát foglaló adatstruktúrákat használunk. Ha az alkalmazást a felhasználó vezérli, nem tudhatjuk, hogy a létrehozott adatstruktúrára hányszor és mennyi ideig lesz még szükség. Nem tudhatjuk tehát, hogy az objektumot mikor lenne célszerű eldobni, illetve meddig kellene megtartani a nagy memóriafoglalás ellenére is.
Ezt a helyzetet kezelhetjük igen egyszerűen és hatékonyan a gyenge referenciák felhasználásával. Az adatstruktúra létrehozása és használata után létrehozzuk a gyenge referenciát, majd egyszerűen eldobhatjuk az erős referenciákat. Ha alkalmazásunk ezután nem kerül memóriaszűkébe, a GC nem fog begyűjtést kezdeni, tehát az adatstruktúra által elfoglalt memória érintetlen marad. Amikor újra szükség lesz az objektumra, új erős referenciát kérhetünk hozzá, nem kell újra létrehoznunk azt. Természetesen ha az alkalmazás túl sok memóriát használ, a GC felszabadítja az elfoglalt memóriaterületet.
A WeakReference osztálynak két konstruktora van:

```
WeakReference(Object target);
WeakReference(Object target, Boolean
trackResurrection);
```

A target paraméterben adhatjuk meg azt az objektumot, amelyre a referencia mutatni fog, a második paraméter értéke pedig azt határozza meg, hogy helyreállítható legyene a hivatkozott objektum erős referenciája a Finalize metódus futása után is. Az első konstruktor a trackResurrection paraméter értékét false-ra állítja, általában csak ilyen gyenge referenciák létrehozásának van értelme. Az újraélesztést nem támogató gyenge referenciákat rövid gyenge referenciának (short weak reference), a másik típust pedig hosszú gyenge referenciának (long weak reference) nevezzük.
Ha az objektumnak nincs Finalize metódusa, akkor a két típus pontosan egyformán viselkedik. A gyenge referencia létrehozása után törölnünk kell az objektum valamennyi erős referenciáját, különben a GC semmiképpen nem tudja felszabadítani az elfoglalt memóriát. Ha az objektumot újra használni szeretnénk, a gyenge referenciát erős referenciává kell átalakítanunk. Az erős referenciát a Weak Reference objektum Target property-je tartalmazza. Ha ennek értéke null, akkor a GC már felszámolta az objektumunkat.

A gyenge referenciák részletei

Amint az már az eddigiekből is látható, a WeakReference objektumok viselkedése eltér a szokásos objektumokétól. A szokásos esetben, ha gyökérhivatkozás mutat egy objektumra, amely egy másik objektumra mutató hivatkozást tartalmaz, mindkettő elérhetőnek, így a GC által érinthetetlennek minősül. Ha azonban a gyökérhivatkozás egy WeakReference típusú objektumra mutat, akkor az abból hivatkozott objektumot a GC memóriaszemétként kezeli. Az alkalmazás felügyelt heap-je két belső adatstruktúrát tartalmaz a gyenge referenciák nyilvántartására: a rövid gyenge hivatkozási táblát (short weak reference table) és a hosszú gyenge hivatkozási táblát (long weak reference table). A táblák a felügyelt heap objektumait azonosító mutatókat tartalmaznak. WeakReference típusú objektum létrehozásakor nem történik memóriafoglalás a felügyelt heap-en (tehát hagyományos értelemben vett objektum nem is jön létre), csak a megfelelő referenciatáblába kerül új hivatkozás, mégpedig a konstruktor paramétereként megadott objektumreferencia. A new operátor által visszaadott érték pedig a táblázatba felvett új referencia memóriacíme lesz. A gyenge referenciákat tartalmazó két táblázat nem része az alkalmazás gyökérkészletének, így az itt található hivatkozást a GC nem fogja figyelembe venni, eltakarítja az objektumot.
Tekintsük át tehát újra, hogyan is történik a GC futása:

- A GC felépíti az elérhető objektumok gráfját, az előző részben ismertetett módon.
- A GC végigvizsgálja a rövid gyenge referencia táblát. Ha a táblában lévő mutató olyan objektumot azonosít, amely nem része a gráfnak, akkor az objektum már nem elérhető, tehát a táblában lévő mutatót a GC null-ra állítja.
- Ezután a GC a Finalization queue-t vizsgálja végig. Ha olyan mutatót talál, amelynek objektuma nem része a gráfnak, a mutatót átmozgatja a Freachable queue-ba. Ezzel egy időben az objektum felkerül a gráfra is, mivel újra elérhetőnek minősül.
- Ezután a hosszú gyenge referencia tábla következik, ha az itt szereplő mutató olyan objektumot azonosít, amely nem szerepel a gráfban (ahol már a Freachable queue-ban lévő mutatók objektumai is szerepelnek), az objektumot szemétnek minősíti, a mutatót pedig null-ra állítja.
- Végül a GC a megmaradt objektumok mozgatásával megszünteti a címtérben maradt lyukakat.

Ez azt jelenti tehát, hogy ha Finalize metódust tartalmazó objektumhoz készítünk hosszú gyenge referenciát, akkor objektumunk (erős referencia nélkül is) „túléli" a GC futását, az erős referencia a finalizálás után is helyreállítható ma-

rad. Ezt a jelenséget mutatja be a mintaprogram következő metódusa:

```
static void WeakRef() {
    WeakRefObjektum obj = new
    WeakRefObjektum("WeakRef");
    //gyenge referencia létrehozása, amely
    //újraélesztést is lehetővé tesz
    WeakReference wr = new WeakReference(obj,
    true);
    obj = null;
    Collect();

    WaitPendingFinalizers();
    //GC futása után is helyreállítható az erős
//referencia
    obj = (WeakRefObjektum) wr.Target;
    Console.WriteLine(obj.eletjel);
    //két GC futás kell a memória
    //felszabadításához
    obj = new WeakRefObjektum("WeakRef");
    wr = new WeakReference(obj, true);
    obj = null;
    Collect();
    WaitPendingFinalizers();
    Collect();
    WaitPendingFinalizers();
    obj = (WeakRefObjektum) wr.Target;
    if (obj== null)
      Console.WriteLine("Nem sikerült a
      helyreállítás.");
    else Console.WriteLine(obj.eletjel);
}
```

A WeakReference objektum Target property-je a hivatkozási tábla megfelelő elemét (vagyis jó esetben éppen a követett objektum mutatóját) adja vissza. Ha a visszaadott érték null, akkor az objektum már megsemmisült. Rövid referencia esetén a GC null-ra állítja a megfelelő tábla mutatóját, amint úgy találja, hogy az objektum már nem elérhető az alkalmazásból. Ha az objektum tartalmazott Finalize metódust, az még nem futott le, tehát az objektum még elérhető, de az erős referencia már nem állítható helyre a Target property segítségével.

A hosszú referenciatáblában lévő hivatkozást a GC csak akkor törli, ha az objektum által elfoglalt memóriaterület már felszabadítható, vagyis a Finalize metódus már korábban lefutott.

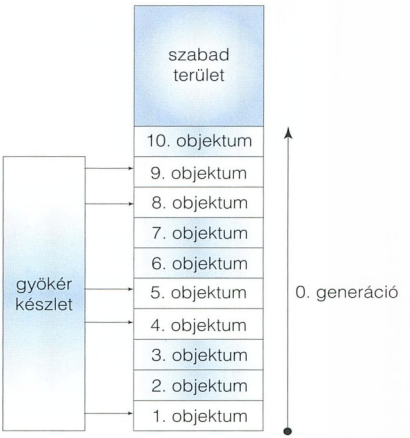
Objektumgenerációk

Az objektumok generációkba sorolása a teljesítmény növelését (a GC futásidejének csökkentését) szolgálja. A generációk használata a következő feltételezések alapján növelheti a teljesítményt:

- Az újabb objektumok várható élettartama rövidebb.
- A régebbi objektumok várható élettartama hosszabb.
- A nagyjából egyszerre létrehozott objektumok általában valamilyen kapcsolatban vannak egymással, az alkalmazás rövid időn belül fogja használni őket, és egyszerre válhatnak fölöslegessé
- A heap egy részének folytonossá tétele hamarabb elvégezhető, mint az egész heap-é.

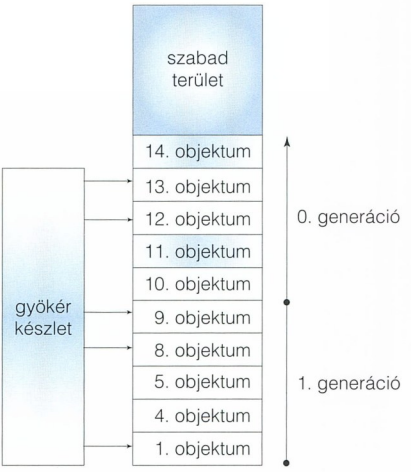
Tekintsük át, hogyan és minek alapján végzi el a GC a heap objektumainak generációkba sorolását.

Kezdetben a felügyelt heap nem tartalmaz objektumokat. A hozzáadott objektumok a 0. generációba kerülnek, egészen addig, amíg GC-futás nem történik (akár a kódból indítjuk, akár automatikusan indul a heap foglaltsága miatt).



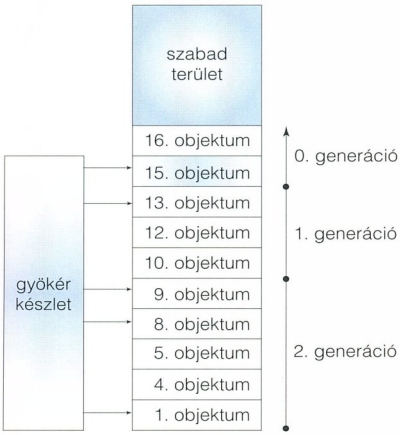
A felügyelt heap az első begyűjtés előtt

Amikor begyűjtésre kerül sor, a GC a szokásos módon szétválasztja a szemetet a még használható objektumoktól, a maradékot átmozgatja az alapcím közelébe, és ezeket átsorolja az 1. generációba. Tehát az 1. generációba azok az objektumok kerülnek, amelyek már „túléltek" egy begyűjtést. A 0. generáció a begyűjtés után üres, de a később létrejövő új objektumok természetesen ide kerülnek.



A felügyelt heap az első begyűjtés után

Ha a 0. generáció megtelik, a GC újabb begyűjtést kezdeményez. Ekkor az 1. generáció „túlélői" a 2. generációba kerülnek, a 0. generáció megmaradt objektumai pedig az 1. generációba.



A felügyelt heap a második begyűjtés után

A .NET szemétgyűjtője jelenleg 3 generációt támogat, tehát a 2. generáció túlélői maradnak a 2. generációban.
Az objektumok generációváltását a mintaprogram következő metódusában figyelhetjük meg:

```
static void Generacio() {
   Console.WriteLine("Az objektum generációk
   maximális száma: {0}",GC.MaxGeneration);
   Szemet[] gen=new Szemet[GC.MaxGeneration+1];
   //létrejön 3 objektum, referenciájuk a tömb
   //elemeiben
   for (int i=0;i<=GC.MaxGeneration;i++)
     gen[i]=new Szemet(i.ToString());
   //egyenként töröljük a referenciákat, és
   //elindítjuk a GC-t, a túlélők magasabb
   //generációba kerülnek
   for (int i=0;i<=GC.MaxGeneration;i++) {
     Console.WriteLine("A(z) "+i.ToString()+".
     objektum generációja:{0}",
     GC.GetGeneration(gen[i]));
     gen[i]=null;
     Collect();
     WaitPendingFinalizers();
   }
}
```

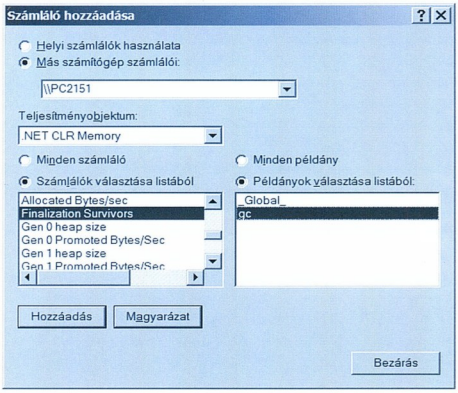
Amikor a heap megtelik, a GC induláskor eldönti, hogy a begyűjtést csak a 0. generációban, vagy a teljes heap-en végezze el. Mivel az új objektumok várható élettartama rövid, jó esély van rá, hogy a 0. generációban végzett begyűjtés számottevő mennyiségű memória felszabadítását teszi lehetővé, a teljes begyűjtéshez viszonyítva jelentősen rövidebb idő alatt. Ha a 0. generációból nem sikerült biztosítani a megfelelő mennyiségű tárterületet, a begyűjtés folytatódhat az 1. és 2. generációban. Jelentős teljesítménynövekedést okozhat az is, hogy az új objektumok mindig folyamatosan, egymás utáni címeken jönnek létre. Az egyik feltételezésünk szerint a közel egy időben létrejövő objektumok szoros kapcsolatban vannak egymással, így általában a hozzáférés is közel egy időben történik. Mivel az objektumok folyamatos címtérben helyezkednek el, a CPU cache kihasználása optimális lehet.

Nagy objektumok kezelése

Szintén a teljesítményoptimalizálás a célja a nagy objektumok (több mint 20 000 byte) különálló kezelésének is. A nagy objektumok önálló, erre a célra fenntartott heap-en foglalnak memóriát. A memóriafoglalás, a Finalize metódusok futtatása és a memória felszabadítása is a már megismert módon történik. Elmarad viszont a memória „tömörítése", vagyis a GC nem tölti fel a megmaradt objektumokkal az alapcímtől kezdve a címteret, megmaradnak a törölt objektumok helyén keletkezett lyukak. Ennek oka az, hogy a nagy objektumok mozgatása túl sok CPU-időt venne igénybe.

A GC monitorozása

A GC működésének valós idejű monitorozására számos teljesítményszámláló szolgál, amelyek megjelenítésére a System Monitor ActiveX kontroll szolgál. A System Monitort legegyszerűbben a PerfMon.exe futtatásával indíthatjuk el.



Teljesítményszámláló hozzáadása

A GC-hez tartozó számlálókat a .NET CLR Memory objektum kiválasztásával érhetjük el. Ezután ki kell választanunk a megfigyelni kívánt alkalmazást és a megfelelő számlálókat a grafikonok megjelenítéséhez.

Szerényi László
szerenyi.l@met.hu

A cikkben szereplő URL-ek:

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

ASP.NET 2.0 (Whidbey)

Mi várható a 2005-ös ASP.NET-ben? I. rész

Adatelérés, adatforrás-vezérlők, GridView

Cikksorozatot indítunk a .NET Framework következő generációjáról, amelynek Whidbey a kódneve. Az első három részben az ASP.NET-ről lesz szó, aztán az ADO.NET-ről, majd a .NET Framework egyéb újdonságairól.

ASP.NET 2.0

Miben különbözött az ASP.NET az ASP-től? Mindenben. Az ASP.NET-fejlesztők egy teljesen új architektúrát dolgoztak ki dinamikus webtartalom előállítására, amely a lehető legjobban igyekszik kihasználni a .NET Framework képességeit. Radikális váltás volt, és három év használat után elmondható, hogy igen jól sikerült a keretrendszer. Egy ilyen start után a fejlesztőknek nehéz felülmúlniuk saját magukat. Ezért a 2.0 nem lesz radikálisan más, sőt a fejlesztők szándéka szerint felülről kompatibilis lesz az 1.1-es verzióval. Azaz nem kell készülni hatalmas váltásra, inkább arra, hogy rengeteg új szolgáltatást fogunk kapni, amelyeket nekünk kellett megírni az 1.1-ben. Az erős alapot most kiegészítik számtalan kényelmi szolgáltatással, valamint most van idejük sok, eddig elhanyagolt terület (pl. többnyelvű GUI) kidolgozására.

Csak az ASP.NET legalább 2000 új típust fog tartalmazni!

Csak az ASP.NET legalább 2000 új típust fog tartalmazni az előző verzióhoz képest, ez jelzi a fejlesztések volumenét!

VS.NET Community Technology Preview 2004. március

Erre a verzióra épülnek a cikkünkben látható példakódok. Nem is annyira a VS.NET a mérvadó, hanem, hogy az e verzióval járó .NET Frameworkre építettem a példákat. Verziószám: 2.0.40301.

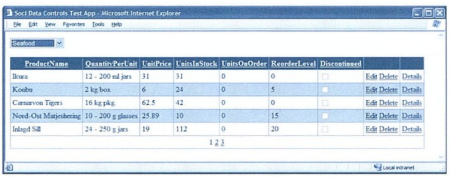
Egy alfa fázisban járó termékről van szó, amely várhatóan csak 2004 nyarán lép át a béta státusba, azaz egyes metódusok vagy objektumok még könnyen változhatnak. A cikk írása közben a 2003. őszi verzióra írt példáimat jelentősen át kellett dolgoznom, annyit változott még a termék, remélem ez a változat már közelebb áll a végleges formához. Csapjunk hát bele!

Adatelérő vezérlők és GridView

Az adatbázisok használata mindig is kiemelt fontosságú volt a webalkalmazások írása során. Az 1.1-ben pár sornyi kóddal könnyen lehetett adatbázisokkal kommunikálni. A korábbi próbálkozásokkal ellentétben a databinding, azaz a vizuális vezérlők adatokhoz történő deklaratív hozzákötése valóban használható volt, de csak egy irányban. Azaz könnyedén meg tudtunk jeleníteni csak olvasható módon adatokat például egy TextBoxban, de a lap felpostázásakor a databinding nem tudta visszaírni a változtatásokat az adatbázisba. A jó hír, hogy a 2.0-ban ez már működik. Ráadásul olyannyira automatizálták a folyamatot, hogy egy árva kódsor leírása nélkül is működik a kétirányú adatkötés! Persze azért deklaratívan le kell írni, milyen adatokat és hová szeretnénk mozgatni. Ezt az aprólékos, unalmas munkát ügyesen tudja automatizálni a Visual Studio .NET. Ha például a Server Explorerből rádobunk egy adatbázistáblát egy WebFormra, akkor azonnal kapunk egy táblázatos nézetet, amely élő adatbázis-kapcsolattal rendelkezik, azaz nemcsak megmutatja a tábla tartalmát, hanem még módosítani is lehet azt (INSERT, UPDATE, DELETE).

No de lássuk ezt részleteiben! Először is azt kell átgondolnunk, hogy még ha meg is akarjuk úszni kód nélkül a bulit, az adatbázis-elérés részleteit akkor is le kell írni valahol. Ez lesz az aspx lap, a WebForm. Kell valaki, aki a felpostázott, módosított tartalmat elkapja, ezért kell egy olyan vezérlő, amely képes kommunikálni az adatbázissal, és kiolvassa a hozzákapcsolt vezérlő adatait. Ezek az új Data Source Controlok. (Az persze joggal kritizálható, hogy egy ilyen GUI-ba gyömöszölt adatelérés korrekt megoldásnak tekinthető-e. Szerintem csak nagyon egyszerű alkalmazásoknál helyes így dolgozni.)

Példánkban termékkategóriánként listázzuk ki a termékeket. A kategóriákat egy legördülő lista tartalmazza, amelyben kategóriát váltva az alsó grid automatikusan frissül, a kiválasztott ételtípust megjelenítve:



Adatelérés kód nélkül az új GridView vezérlővel

Célunk, hogy a termékek adatai módosíthatók is legyenek, illetve legyen módunk törölni a nemkívánatos sorokat. Mindez kód nélkül? Igen. Lássuk, hogyan!
Első lépésként deklaráljuk a lap fejlécét:

```
<%@ page language="C#"
 compilewith="Default.aspx.cs"
 classname="NetAcademia.Default_aspx"%>
```

Látható, hogy nem *src*, hanem *compilewith* attribútum hivatkozik a háttérkódra, és nem *inherits*, hanem *classname* mutatja meg a mögöttes osztály típusát. (Most ugyan nem lesz háttérkódunk, de nem árt látni, itt mi változott.) Azért változtak az elnevezések, mert megváltozott a feldolgozási modell. Eddig az ASPX elemző által generált osztályt leszármaztatták az *inherits*-ben megadott saját code behind osztályunkból. Így tudtunk együttműködni az aspx lappal, valamint így tudott a VS.NET tervezője kódot injektálni a háttér forráskódunkba. Ezen megoldás nagy hátránya, hogy a Designer firkálgat a mi kódunkba, ami néha konfliktusokat okoz (az a @#$! designer kitörölte a kódomat). Persze oda van írva, hogy ne bántsuk azt a bizonyos kódrészletet, mert az a designer felségterülete, azért mindig vannak felfedező kedvű emberek.
A 2.0 fordítási modellje teljesen más. Mind C#-ban mind VB.NET-ben bevezetik a *partial*, azaz részleges osztályokat. Ez lehetővé teszi, hogy egy osztály tartalmát tetszőleges számú forráskódban írjuk meg, amelyeket majd a fordító gyúr egybe. Ezt az ASP.NET (és a WinForms is) úgy használja ki, hogy az ASP.NET parser és a VS.NET által generált kód is a mi háttérkódunktól független, külön fájlba kerül. Nem is látjuk ezeket! Ebben az esetben szó nincs semmiféle öröklésről, hanem egyszerűen a mi kódunkat hozzáfordítják a generált kódhoz. Ezért a 2.0-ban már nem Code Behind, hanem Code Beside technológiáról beszélünk. Például az előbbi fejléchez ez a részleges code beside osztály passzol:

```
using System;
namespace NetAcademia {
 public partial class Default_aspx {  }
}
```

De térjünk rá az adatelérésre! A felül látható legördülő lista így hozható létre:

```
<asp:dropdownlist id="CategoryList"
 runat="server"
 datasourceid="CategoryDataSource"
 datavaluefield="CategoryID"
 datatextfield="CategoryName"
 autopostback="True">
</asp:dropdownlist>
```

Ami új, az a *datasourceid* jellemző. Itt egy Data Source Controlt kell megadni. Esetünkben ez egy SqlDataSource lesz, amellyel SQL Server vagy bármilyen egyéb relációs adatbázis adatait tudjuk kezelni:

```
<asp:sqldatasource
 id="CategoryDataSource"
 runat="server"
 selectcommand=
 "SELECT CategoryID, CategoryName,
 Description, Picture
 FROM dbo.Categories"
 insertcommand=
 "INSERT INTO Categories(
 CategoryName, Description)
 VALUES (?, ?)"
 updatecommand=
 "UPDATE Categories SET
 CategoryName = ?,
 Description = ?
 WHERE (Categories.CategoryID = ?)"
 deletecommand=
 "DELETE FROM Categories
 WHERE (Categories.CategoryID = ?)"
 providername="System.Data.OleDb"
 connectionstring=
 "Provider=SQLOLEDB.1;
 Integrated Security=SSPI;
 Initial Catalog=Northwind;
 Data Source=(local);"
 datasourcemode="DataSet"
 cacheduration="60"
 enablecaching="True">
</asp:sqldatasource>
```

Az attribútumok zöme triviális, ami érdekes, az a három utolsó. A *datasourcemode* azt adja meg, hogy az adatelérés *DataSet* vagy *DataReader* alapú legyen-e. A *DataReader* nyilván valamivel gyorsabb, de nem lehet se szűrni, se cache-elni, se rendeztetni a kimenetet. (Módosítani lehet, mert arra ott vannak az SQL-parancsok, amiket közvetlenül végre tudnak hajtani.)
DataSet esetén az utolsó két attribútummal maximum 60 mp-ig tartó cache-elést állítottunk be. Azaz ha van elég szabad memória a cache manager részére, akkor a lekérdezést nem kell megismételni 60 mp-ig. Ami külön izgalmas, hogy meg lehet adni adatbázistartalom-függő cache-elést is, amit az *SqlCacheDependency* attribútummal lehet szabályozni. (Erről bővebben a sorozat következő cikkében.)
A kategórialista feltöltve. Amikor felpostázódik a tartalma, le kellene szűrni a termékeket a kiválasztott CategoryID alapján. Ez már nagyon kódszagú feladvány, de nem az adatelérő vezérlőkkel! Lássuk csak:

```
<asp:sqldatasource
 id="ProductDataSource"
 runat="server"
 selectcommand=
 "SELECT ProductID, ProductName,
 QuantityPerUnit, UnitPrice,
 UnitsInStock, UnitsOnOrder,
 ReorderLevel, Discontinued
 FROM dbo.Products
 WHERE CategoryID = ?"
 insertcommand=
 "INSERT INTO Products
 (ProductName, SupplierID,
 CategoryID, QuantityPerUnit,
 UnitsInStock, UnitsOnOrder,
 ReorderLevel, Discontinued)
 VALUES (?, ?, ?, ?, ?, ?, ?, ?)"
 updatecommand=
 "UPDATE Products SET
 ProductName = ?, QuantityPerUnit = ?,
 UnitsInStock = ?, UnitsOnOrder = ?,
 ReorderLevel = ?, Discontinued = ?
 WHERE (Products.ProductID = ?)"
 deletecommand=
```

```
"DELETE FROM Products
WHERE (Products.ProductID = ?)"
providername="System.Data.OleDb"
connectionstring=
"Provider=SQLOLEDB.1;
Integrated Security=SSPI;
Initial Catalog=Northwind;
Data Source=(local);">
<selectparameters>
 <asp:controlparameter
  name="?"
  propertyname="SelectedValue"
  type="Int32"
  controlid="CategoryList">
 </asp:controlparameter>
</selectparameters>
</asp:sqldatasource>
```

Az adatelérő vezérlőben megadhatunk paramétereket, amelyeket postback esetén automatikusan kiolvas a vezérlő, és a paraméterektől függően hajtja végre a *selectcommand*-ban tárolt parancsot. Esetünkben a CategoryList vezérlő (a korábbi lista) *SelectedValue* jellemzőjét olvassa ki, amelyet a WHERE CategoryID = ? feltételben a ? helyére helyettesít be. Stringművelettel? Természetesen *nem*. A mai paranoiás világban nem szabad SQL Stringeket ragasztgatni házi barkácsmódszerekkel. Az alábbi parancs figyelhető meg az SQL Profilerben, ha kiválasztjuk a 2-es ID-jű termékkategóriát:

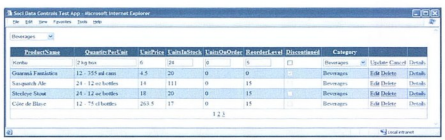
```
exec sp_executesql
N'SELECT ProductID, ProductName, ...
FROM dbo.Products
WHERE CategoryID = @P1',
N'@P1 int', 2
```

Ember legyen a talpán, aki így a parancsba be tud injektálni plusz sql kifejezést! (A tudomány mai állása szerint nem lehet.) Paraméterforrás lehet vezérlő tetszőleges jellemzője, querystring, form paraméter, session változó vagy cookie érték is. Már csak egy lépés van hátra, meg kell mutatni a leszűrt sorokat. Erre egy új vezérlőt veünk be, amelynek *GridView* a becses neve. Ez a mostani *DataGrid* felturbózott változata:

```
<asp:gridview
 id="ProductGridView"
 runat="server"
 autogeneratecolumns="False"
 datasourceid="ProductDataSource"
 datakeynames="ProductID"
 allowpaging="True"
 pagesize="5"
 allowsorting="True" ...>
 <columns>
  <asp:boundfield
   sortexpression="ProductName"
   datafield="ProductName"
   headertext="ProductName" />
  <asp:boundfield
   sortexpression="UnitPrice"
   datafield="UnitPrice"
   readonly="True"
   headertext="UnitPrice" />
  <asp:checkboxfield
   sortexpression="Discontinued"
   headertext="Discontinued"
   datafield="Discontinued" />
  <asp:commandfield
   showdeletebutton="True"
   showeditbutton="True" />
  <asp:hyperlinkfield
   datanavigateurlformatstring=
   "Details.aspx?id={0}"
   datanavigateurlfields="ProductID"
   text="Details" />
 </columns>
 <pagersettings
  mode="NumericFirstLast" />
</asp:gridview>
```

A formázáshoz kapcsolódó sorokat kidobtam a kódból, az élő példákban **[1]** természetesen benne vannak. Látható, hogy a *datasourceid* a szűrt terméket generáló adatforrásra utal, a *datakeynames* pedig a forrás *DataSet* elsődleges kulcsot tartalmazó oszlopát azonosítja. De miért érdekli ez a gridet? Nos, az update és delete műveletekhez meg kell ragadni a kérdéses sort, és ehhez szükség van a kulcs értékére. Látható, hogy kértünk rendezést és lapozást is. Mindkettőt *DataSet*, illetve *DataView* szinten oldják meg, ami messze nem tökéletes megoldás. Nagyszámú sor esetén elfogadhatatlan, hogy 100000 sor leugrik a webformra, amiből aztán pár sort kivéve mind eldobják. Az ADO.NET 2.0-ban bevezettek az *SqlCommand* objektumon egy *ExecutePageReader* metódust, amivel korrektül, szerveroldali kurzorokkal tényleg csak a szükséges sorokat hozzák le. Sajnos azonban ebben a verzióban a *GridView* még nem használja ki ezt a lehetőséget (vagy én nem vettem észre, hogy kihasználja). A pletykák szerint ez a lapozós ADO.NET szolgáltatás ki is marad a végleges Frameworkből. Majd meglátjuk jövőre.

Egy kicsit spékeljük meg a példánkat. Gyakori igény, hogy azokat az értékeket, amelyek egy fix, diszkrét, véges halmazból kerülnek ki, listából választhassuk ki, ne kelljen beírni. Esetünkben a termék kategória ilyen. Szerkesztő nézetben ezt szeretnénk látni:



Kibővített szerkesztőfelület kategórialistával

Normál nézetben természetesen csak a kategória neve látszik, lista helyett statikus szövegként. Az új funkcióhoz a grid adatforrását kissé át kell konfigurálni:

```
selectcommand=
"SELECT ProductID, ProductName,
QuantityPerUnit, UnitPrice,
UnitsInStock, UnitsOnOrder,
ReorderLevel, Discontinued,
CategoryName, Products.CategoryID
FROM dbo.Products
INNER JOIN Categories
ON Products.CategoryID =
Categories.CategoryID
WHERE Products.CategoryID = ?"
```

A listázó megjelenítéshez fel kell oldanunk a Products.CategoryID mező értékét a tényleges kategória-

névre és a CategoryID értékére is szükségünk lesz, hogy be tudjuk állítani a legördülő listát a termékhez tartozó kategóriasorra. A CategoryID-t belevesszük az *updatecommand*-ba:

```
updatecommand=
"UPDATE Products SET
CategoryID = ?,
ProductName = ?, QuantityPerUnit = ?,
UnitsInStock = ?, UnitsOnOrder = ?,
ReorderLevel = ?, Discontinued = ?
WHERE (Products.ProductID = ?)"
```

A plusz oszlop megjelenítését végző sablon:

```
<asp:templatefield>
 <headertemplate>Category
 </headertemplate>
 <itemtemplate>
  <asp:label headertext="Category">
   <%# Eval("CategoryName") %>
 </itemtemplate>
 <edititemtemplate>
  <asp:dropdownlist id="CategoryListForUpdate"
   runat="server"
   datasourceid="CategoryDataSource"
   datavaluefield="CategoryID"
   datatextfield="CategoryName"
   selectedvalue=
   '<%# Eval("CategoryID") %>'
   />
 </edititemtemplate>
</asp:templatefield>
```

A példa egyúttal azt is demonstrálja, hogy a databinding kifejezések jelentősen egyszerűbbek lettek.

```
ASP.NET 1.1:
<%# DataBinder.Eval(Container.DataItem,
  "CategoryID") %>

ASP.NET 2.0
<%# Eval("CategoryID") %>
```

Listázás nézetben a *Label* látszik, szerkesztő nézetben a lista. Így már érthető, miért kellett a plusz két oszlop a *selectcommand*-ba. Zárásul már csak egy lépés van hátra. Az *updatecommand*-nak meg kell mondani, hogy az első paraméter, a CategoryID értékét a kategórialistából felpostázott értékből helyettesítse be. Ezt is megúszhatjuk kód nélkül, köszönhetően a paraméterezhető szűrési lehetőségnek:

```
<updateparameters>
 <asp:formparameter
  name="?"
  type="Int32"
  formfield=
  "ProductGridView$_ctl3$CategoryListForUpdate">
 </asp:formparameter>
</updateparameters>
```

A megoldás egyetlen szépséghibája a lista id-ja (*formfield*). A gridbe ágyazott *DropDownList* generált kliensoldali neve kiolvasható a vezérlő *ClientID* tulajdonságából, de ehhez meg kellene szerezni a vezérlőre mutató referenciát. Ez már több kód, mint amit egy databinding kifejezés esztétikusan elbír, ezért ezt már háttérkódba kellene írni (ettől most eltekintünk).

Még két kérdést szeretnék megválaszolni. Miért OLEDB? Miért nem natív SQL Serveres osztályok? Egyszerűen azért, mert ez az alfa verzió az OLEDB providerekkel még simábban működik, mint az SQL-essel. De gondolom, a béta verzióban már át tudom írni a kódot SQL Server specifikusra. A másik, hogy mivel egyszerűbb így deklaratívan leírni mindent, nem lenne egyszerűbb programsorokkal kifejteni, mit is akarunk csinálni. Nos, kézzel valóban elég kiábrándító a fenti sorokat beírni. De itt jön a képbe egy jó IDE. A VS.NET a fenti kódot 99%-ban képes legenerálni. A deklaratív módszer pont a kódgenerátoroknak kedvez, amelyek valamely programnyelven sokszor csak nehezen karbantartható módon tudnak kódot írni, a szabályosabb xml leírás sokkal könnyebben megfogható a varázslókkal. Szóval elmondhatjuk, hogy a fenti megközelítés inkább varázsló, mint emberbarát.

A DetailsView vezérlő

A *GridView* egyszerűen használható eszköz sok sor megmutatására. Sok oszlop esetén azonban problémás a megjelenítés és a szerkesztés is a túl hosszú sorok miatt. Ráadásul a rengeteg adatban elveszik a felhasználó szeme. A *DetailsView* egy rekordot mutat meg, minden egyes oszlopát egy új táblázatsor *TextBox*-ában vagy egyéb vezérlőjében (pl. *CheckBox*). Sajnos a 2003. őszi verzióban még jól működő vezérlőben olyan, nem dokumentált változtatásokat eszközöltek, amely miatt a 2004. márciusi .NET Framework alatt nem tudtam működőképessé tenni a korábbi példáimat.

Az ObjectDataSource adatforrás

Nagyon jól látszik, hogy a *DataSet* alapú modell mellé egyre erősebben kezd felsorakozni az üzleti objektum alapú adatkezelés. Ebben az esetben nem *DataRow* objektumok reprezentálják az üzleti információt, hanem általunk írt osztályokból létrehozott objektumok, amelyek tagváltozókban tárolják az általában adatbázisból származó adatokat. Ebben a modellben az üzleti logikát az osztály, entitás metódusaiban implementáljuk.

A modellt támogatandó lesz egy *ObjectDataSource* nevű adatforrás. Hasonlóan a korábban látott *SqlDataSource*-hoz, ez is forrása lehet DataBound vezérlőknek. Az *ObjectDataSource*-t fel kell paraméterezni, hogy az adatelérő réteg mely metódusaival képes a select, insert, update és delete utasításokat végrehajtatni. Ha sikerül használható dokumentációt fellelni, akkor a cikksorozat következő részében bemutatom e vezérlő használatát is.

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

A cikkben szereplő URL-ek:

[1] http://www.netacademia.net/tudastar/default.asp?upid=2656

Windows szolgáltatások 3. rész

NT AUTHORITY\NetworkService

A sorozat előző részében a LocalService fiók nevében futó szolgáltatásokat pécéztük ki, most a másik, a LocalSystem fiók használatának mértékéhez képest törpe minoritásnak számító NetworkService fiókhoz köthető szervizek kerülnek a fókuszba.

A második 9

Ebben a részben összesen 9 szervizt vizsgálunk meg. Azonos csoportba sorolásuk oka, hogy mind az újonnan (Windows XP, Windows Server 2003) bevezetett NetworkService (Hálózatszolgáltatás) fiókkal működnek, amelyről a korábbi részek alapján már tudnunk kell egy-két dolgot, de azért nézzük meg a tulajdonságaikat röviden újra:

- gyengített jogosultsági körrel futnak, ami körülbelül megfelel egy átlagos felhasználói fiók lehetőségeinek, de úgy, hogy közben a helyi erőforrásokhoz alig van jogosultságuk,
- a hétköznapi értelemben vett erőforrás-hozzáférés (például objektumokhoz) nem megengedett,
- az ezzel a fiókkal futó szolgáltatások az adott számítógépfiók (Computer Account) hitelesítő adatai segítségével érik el a hálózati erőforrásokat,
- és végül ehhez a fiókhoz sincs jelszó hozzárendelve.

A számítógépfiók

Azt ugye tudjuk, hogy minden számítógépet, amelyen a Windows NT/2000/XP vagy a Windows Server 2003 fut, egyértelműen lehet azonosítani egy egyedi számítógépfiókkal is. Ezt a fiókot a felhasználói fiókokhoz hasonlóan hitelesítésre és naplózásra is lehet használni egy tartományban vagy egy hálózatban. Természetesen akkor is létezik a számítógépfiók, ha a gép nem tagja egy tartománynak, de az biztos, hogy a tartományvezérlő (ha több van, akkor az a tartományvezérlő, amelyik a RID, azaz a Relative Identifier Master szerep birtokosa) a tartományba lépéskor ezt frissíti. Ennek a fióknak a neve (gépnév$), és a szintén ekkor generált jelszava segítségével fog ezután a gép belépni a tartományba, még a felhasználói belépés előtt. Ez a belépés aztán rezidens marad, nem is kell például az adott gépen történő felhasználó kilépésekor megújítani (bár egy alkalommal azért lezajlik automatikusan a háttérben az újrahitelesítés: ha definiáljuk az SMB kapcsolat leválasztást a Csoportházirendben). Száz szónak is egy a vége, ezt a fiókot és a jelszavát használja az operációs rendszer a következő szervizek futtatására:

- DHCP Client
- Distributed Transaction Coordinator
- DNS Client
- FTP Publishing Service
- License Logging
- Performance Logs and Alerts
- Remote Procedure Call (RPC) Locator
- Simple TCP/IP Services
- Windows Media Services

DHCP Client (DHCP-ügyfél)

A szerviz rövid neve: Dhcp
Az alkalmazás neve: dhcpcsvc.dll (svchost.exe)
Függés: AFD Networking Support Environment, TCP/IP Protocol Driver, IPSEC Driver
Függesztés: WinHTTP Web Proxy Auto-Discovery Service
Porthasználat: TCP: 68; UDP: 67, 68, 1029
Alapértelmezett indítás: automatikus

A DHCP-ügyfél neve kicsit félrevezető, anno furcsa is volt, hogy miért is fut automatikusan a fix IP-címmel ellátott szerveren ez a szolgáltatás. Ám aki leállítja, annak szintén fura, vagy inkább megrázó élményben lesz része, mivel a DHCP-szerverrel való mindennemű kapcsolattartás mellett az IP-címek regisztrálását és DNS-adatok frissítését is ez a szerviz intézi. Ez a folyamat a rendszerindítás után következik, például egy DHCP-kliens gépnél rögtön a DHCP-adatok megérkezése után, de néhány más esemény is kiválthatja, úgymint:

- a kliens IP-címének hozzáadása, megváltoztatása vagy törlése,
- az IP-cím élettartamának változása (pl. a gép újraindítása) vagy az TCP/IP konfiguráció újraépítése (ipconfig /renew),
- a kliens adatainak manuális frissítése a dinamikus DNS engedélyezése esetén (ipconfig /registerdns),
- tagkiszolgáló előléptetése tartományvezérlővé.

Ha a szervizt leállítjuk, a fentiek értelmében nem lesz eredményes kapcsolatunk a DHCP-szerverrel (ergo muszáj manuálisan konfigurálni a TCP/IP-t, vagy jön az APIPA), és a dinamikus DNS-frissítés sem fog működni. Ha a gépünk DNS-szerver is egyben, akkor ilyenkor a jól ismert zónaeltávolítást és -visszaállítást sem tudjuk alkalmazni (net stop/start netlogon). És még sorolhatnánk... Ha esetleg le is tiltjuk, akkor a kapcsolódó (az előző számban tárgyalt) WinHTTP

Web Proxy Auto-Discovery szolgáltatás sem fog rendelkezésre állni. De nem éri meg kézi indítására állítani sem, mert (ellentétben néhány másik társával) ez a szerviz nem fog elindulni automatikusan, ha szükség lesz rá.

Distributed Transaction Coordinator (Elosztott tranzakciók koordinátora)

A szerviz rövid neve: MSDTC
Az alkalmazás neve: msdtc.exe
Függés: Remote Procedure Call, Security Accounts Manager
Függesztés: –
Porthasználat: –
Alapértelmezett indítás: automatikus

Üzemeltetők körében talán nem a mindennaposan emlegetett szolgáltatások körébe tartozik a DTC, pedig néhány területen igencsak fontos, igencsak rá épül jó pár más kiszolgáló alkalmazás, mint például az SQL Server, a BizTalk Server, a Commerce Server, vagy például a Message Queue Server. De használatos számítógéprendszerek, állományrendszerek közötti tranzakciók kezelésére is. A DTC a Component Services része (korábban Microsoft Transaction Server-ként volt ismert) és a külső, úgynevezett kétfázisú tranzakciók irányítója. Ha leállítjuk vagy letiltjuk, akkor a fenti alkalmazásokon kívül mindazon megoldásokkal lesznek menedzselési problémáink, amelyek a COM+ technológiát használják.

DNS Client (DNS-ügyfél)

A szerviz rövid neve: Dnscache
Az alkalmazás neve: dnsrslvr.dll (svchost.exe)
Függés: TCP/IP Protocol Driver, IPSEC Driver
Függesztés: –
Porthasználat: TCP: 53
Alapértelmezett indítás: automatikus

Ennek a szolgáltatásnak a lényege valószínűleg értelemszerű mindenki számára: a kliensoldali névfeloldásról és a DNS-adatok gyorsítótárazásáról van szó. A DNS-ügyfélszerviz a következő szolgáltatásokat nyújthatja egy kliens (mármint a DNS szempontjából kliens) gépen:

Teljes körű gyorsítótárazás

A lekérdezések eredményéből adódó erőforrásrekordok a kliens gyorsítótárába tárolódnak, majd onnan újrafelhasználásra is kerül(het)nek, persze a TTL *(Time to Live = élettartam)* értékének függvényében.

RFC kompatibilis negatív gyorsítótárazás

Kibővítve a szimpla gyorsítótárazás funkcióját a DNS-ügyfél képes arra is, hogy a lekérdezésekre kapott hibás válaszokat is eltárolja. De miért kell eltárolni a hibás bejegyzéseket? A magyarázathoz tudnunk kell a módszer lényegét. Arról van szó, hogy negatív választ akkor kap a kliens, ha az adott névhez nem tartozik erőforrásrekord. Ilyenkor ezt is elteszi a gyorsítótárba, de megjelöli negatívként, és erről a „skarlát betűről" tudni fogja a következő alkalommal *(ami akár másodpercek múlva is lehet)*, hogy az adott gép felé menő lekérdezéssel már nem kell bajlódni, hiszen nem ér semmit. Persze itt is van élettartam, sőt rövidebb is *(maximum 5 perc)*, mint a helyes címek esetén, hiszen előfordulhat, hogy az az összerendelés, ami pár perce még nem volt használható, mostanra már értékes rekorddá nőtte ki magát. Ez a folyamat egyébként a 2308-as számú RFC dokumentum alapján működik **[1]**.

A nem reagáló DNS-szerverek kihagyása

Amikor a DNS-ügyfél elkezd dolgozni, egy úgynevezett keresési lista lapján teszi mindezt. Ebben a listában a szóba jöhető DNS-szerverek prioritási sorrendben szerepelnek. Abban a sorrendben, ahogyan beállítjuk a TCP/IP paraméterekben, vagy ahogy például megkapja a gép a DHCP-szervertől. Értelemszerűen az elsődleges után jön a másodlagos és így tovább, viszont abban az esetben, ha valamelyik kiszolgáló nem elérhető, akkor ideiglenesen hátrébb sorolódik a listában, automatikusan.

Mivel a Windows 2000 óta a DNS működése a tartomány, a tartományvezérlők, a globális katalógus és még sok-sok további szerep megtalálása miatt kulcsfontosságú egy tartományban, ezért ennek a szerviznek az esetek nagy százalékában működnie kell. Ha nem így van, azaz leállítjuk vagy letiltjuk, akkor komoly problémákkal nézünk szembe, hiszen az egykori legendás „… az AD hibák 110%-a a DNS-ből ered" mondás százalékértéke mára már kb. vagy 200-nál járhat ☺.

License Logging (Licencnaplózás)

A szerviz rövid neve: LicenseService
Az alkalmazás neve: llssrv.exe
Függés: –
Függesztés: –
Porthasználat: TCP: 135 (csak a replikációhoz)
Alapértelmezett indítás: letiltva

Ennek a szerviznek a használata szintén velejárója a Windows-világnak. Segítségével az eszköz vagy felhasználó alapú ügyfélliceneket *(CAL = Client Access License)* használó beépített komponensek, például az IIS, a Terminálszolgáltatások, a fájl- és nyomtatószolgáltatások vagy a külső kiszolgáló alkalmazások, mint az SQL Server vagy az Exchange Server ügyféloldali felhasználásának mértékét állíthatjuk be a törvényes keretek közé.

A Windows Server 2003-ba beépített License Logging szerviz telephelyenkénti bontásban gyűjti és összegzi a licencekkel kapcsolatos információkat a telephelyi licencszerveren. Ez azt jelenti, hogy lehetőség nyílik arra, hogy földrajzi vagy szervezeti alapon történjen a licencek beszerzése és menedzselése. Ennek a magyarázata elsősorban ott keresendő, hogy szemmel láthatóan a Microsoft ezen komponensét sem a tipikus magyar viszonyokra tervezték ☺, hanem a sok ezer gépes, sok telephelyes viszonyok közé. De – komolyra fordítva a szót – ez persze semmilyen gondot nem okoz a használatában. Ha nem működik ez a szerviz (ne felejtsük, el, hogy ugyan a Windows 2000 Servernél még automatikusan futott, viszont a Windows 2003 Server összes típusánál már alapból tiltva van), akkor az igen, de „csak" annyit, hogy nem tehetünk eleget a licencek elektronikus adminisztrálásának.

Performance Logs and Alerts (Teljesítménynaplók és riasztások)

A szerviz rövid neve: Sysmonlog
Az alkalmazás neve: Smlogsvc.exe
Függés: –
Függesztés: –

Porthasználat: TCP: 139
Alapértelmezett indítás: kézi, leállítva

Most a kicsit talán nehezen kiismerhető, ám rendkívül hasznos megoldás háttérszolgáltatása jön, amely jó barátja minden üzemeltetőnek: ez a Teljesítménynaplók és riasztások szerviz. Ennek segítségével nyílik lehetőség valamilyen okból problémás helyi vagy távoli számítógépek teljesítményadatainak gyűjtésére. Amikor egy ilyen típusú figyelést hajtunk végre, akkor általában ezt időzítve tesszük, hiszen a legtöbb probléma hosszabb távon jön elő. Ez a szerviz abban segít, hogy az időzítés szerinti működéskor elindítja a naplózást, illetve riasztás(oka)t is kiválthat, ha úgy állítjuk be. A szolgáltatás indulása és leállása tehát lehet manuálisan kiváltható (pl. a valós idejű adatgyűjtéskor), de lehet teljesen automatikusan abban az időszeletben (mondjuk, naponta x percben), ahogyan behangoljuk, például a „logman" paranccsal vagy a Teljesítménynaplók és riasztások konzollal. Ha a szerviz fut, és leállítjuk, akkor az adatgyűjtés befejeződik, és a további, általunk generált időzítések szerinti működésnek is búcsút mondhatunk. Ezért célszerű az alapbeállításon hagyni, hiszen ha letelik egy-egy időzítés, vagy nem történik esemény, úgyis automatikusan leáll.

Remote Procedure Call Locator
(Távoli eljáráshívás lokátor)

A szerviz rövid neve: RpcLocator
Az alkalmazás neve: locator.exe
Függés: Workstation szerviz
Függesztés: –
Porthasználat: TCP: 135
Alapértelmezett indítás: kézi, leállítva (tartományvezérlőkön automatikus)

Ez a szerviz lehetővé teszi az RPC klienseknek hogy megtalálják a hálózaton lévő RPC kiszolgáló(ka)t, továbbá azt is, hogy az RPC névszolgáltatás adatbázisát kezelhessék. A szerviz manuális indításúra van állítva az alapértelmezés szerint, ha viszont leállítjuk vagy letiltjuk, akkor a kliens felől érkező RPC-forgalom nem talál célba, azaz az adott gép nem találja meg az illetékes RPC szervert, így az RPC névfeloldó adatbázist sem. Ha a szervizt a tartományvezérlőn „vágjuk tarkón", akkor nagyobb galibát okozunk, hiszen az összes kliens és maga a tartományvezérlő is zavarba jön.

Simple TCP/IP Services
(Egyszerű TCP/IP szolgáltatások)

A szerviz rövid neve: SimpTcp
Az alkalmazás neve: tcpsvcs.exe
Függés: AFD Networking Support Environment
Függesztés: –
Porthasználat: lásd külön-külön
Alapértelmezett indítás: automatikus

Ez a szerviz az olyan utólag, a Windows-összetevők közül feltelepíthető extra TCP/IP szolgáltatásokat implementálja, mint az Echo, a Discard, Character Generator, Daytime és a Quote of the Day protokollok.

Character Generator (port 19, RFC 864)
Ez a komponens hasznos hibakereső eszköz a mátrixnyomtatók tesztelésére és hibaelhárítására, mert olyan adatokat küld a printerre, ami egy 95 nyomtatható ASCII-karakterből álló karakterkészletre épül.

Daytime (port 13, RFC 867)
A hét napját, a hónapot, az évet, a pontos időt és az időzóna adatait adja vissza. Léteznek olyan alkalmazások, amelyek ennek a szolgáltatásnak a kimenetét a rendszeridő változásának vagy a rendszeridő és egy másik gép rendszerideje közti eltérés hibakeresésére vagy megfigyelésére használhatják.

Discard (port 9, RFC 863)
Válasz vagy visszaigazolás nélkül eldobja a portra érkező összes üzenetet. Használható TCP/IP tesztüzenetek fogadására és továbbítására, illetve bizonyos esetekben az alkalmazások üzeneteldobási funkcióként is használhatják.

Echo (port 7, RFC 862)
A kiszolgálóportra érkező összes üzenet adatát visszaküldi. Az echo parancs hálózati hibakereső és figyelőeszközként hajthat hasznot.

Quote of the Day (port 17, RFC 865)
Ez egy abszolút kényelmi szolgáltatás, amely egy idézetet küld vissza egy- vagy többsornyi szövegként. Általában a levelezőprogramok szignójában szoktuk megjeleníteni ezeket az idézeteket, és a %windir%\System32\Drivers\Etc\Quotes helyről választódhatnak ki, véletlenszerűen.
Ezeknek a protokolloknak a használatához tudnunk kell, hogy külön-külön nem engedélyezhetők, illetve nem is tilthatók le, csak egyben (ha eltávolítjuk), ezért ha nincs szükség rájuk, ne is telepítsük. Ha viszont feltelepítettük, és elindítottuk a szervizt, akkor ez az öt protokoll automatikusan engedélyezve lesz a hálózati interfészen. A szerviz leállítása vagy tiltása viszont csak ezekre a szolgáltatásokra lesz hatással.

Windows Media Services
(Windows Média szolgáltatások)

A szerviz rövid neve: WMServer
Az alkalmazás neve: WMServer.exe
Függés: Remote Procedure Call
Függesztés: –
Porthasználat: TCP: 1755 (MMS), 554 (RSTP), 80 (HTTP), 135 (DCOM, WMI); UDP: 161 (SNMP), 1755 (MMS), 135 (DCOM, WMI)
Alapértelmezett indítás: automatikus

Ez a szerviz sem alapértelmezett, viszont ha feltelepítjük, automatikusan indul, és streaming media (adatfolyam-kiszolgáló) szolgáltatást nyújt az IP alapú hálózatok felett. A Windows Server 2003-ban lévő WMS a korábbi 4 különálló szolgáltatást cseréli le egyre, amely számtalan protokollt ismer (lásd Porthasználat). Ezzel a szolgáltatással például hang- és mozgókép-adatfolyamokat kezelhetünk, továbbíthatunk és archiválhatunk az intraneten vagy akár az interneten keresztül. Ha a szervizt leállítjuk, a streaming media szolgáltatás ignorálásán kívül más nem fog történni.
Cikksorozatunk következő részében elkezdjük a legnagyobb csoport, a LocalSystem fiókkal futó szolgáltatások vizsgálatát.

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

A cikkben szereplő URL-ek:

[1] ftp://ftp.rfc-editor.org/in-notes/rfc2308.txt

Két üzleti platform rövid kiértékelése

A Linux és a Windows összehasonlítása

Sok vállalat van, amely a nagyszámítógépekről és a UNIX alapú rendszerekről az új Intel alapú megoldásokra való átállást fontolgatja, hiszen azok az előbbiek költségeinek törtrészéért nyújtanak ugyanolyan, sőt akár nagyobb teljesítményt.

Bármilyen könnyű legyen is kiválasztani a hardvert, ennél fontosabbá lépett elő az operációs rendszert illető döntés. A platform sokkal több puszta operációs rendszernél. Magában foglalja az alkalmazáskiszolgálót, a biztonsági szolgáltatásokat, a tranzakciófeldolgozást és az erőforrások címtárát. Ha ezek mind integrálva vannak egymással, jóval hatékonyabb üzemeltetést, egyszerűbb felügyeletet és jelentős üzleti hasznot tesznek lehetővé. A platformot illető döntés azért is kulcsfontosságú, mert a platform határozza meg, hogy futtatni tudja-e a vállalat a legkorszerűbb alkalmazásokat, biztonságosan és hatékonyan kezelheti-e informatikai erőforrásait, valamint, hogy versenyképessége megőrzése érdekében képes-e beépíteni rendszerébe a folyamatosan fejlődő új alkalmazásokat. Napjaink gazdasági nehézségei miatt sok vállalatnál szorosabbra fogták az informatikai költségvetést.

Vannak nagyvállalatok, amelyek hatalmas összegeket ruháztak be nagyszámítógépekbe és UNIX rendszerű számítógépekbe. A beruházás számítógépes berendezésekben, módosított és egyedi alkalmazásokban, valamint támogató személyzetben ölt testet. A hardver/szoftvergyártó által esetleg megkövetelt szolgáltatási szerződés díja is jelentős összegre rúghat. Ha mindezt figyelembe vesszük, érthető, hogy a döntéshozók olyan lehetőségek után kutatnak, amelyekkel csökkenthetik az üzemeltetési és a támogatási költségeket. Sok vállalat fontolgatja, hogy a drágább vállalatspecifikus UNIX rendszerekről való áttérést a Linux bevezetésével oldja meg. A Linux sok verziója ingyenes, pontosabban a telepítő média áráért szerezhető be. Népszerűségre tett szert bizonyos fejlesztők körében, és a hardvergyártók közül is egyre többen támogatják.

A helyzet iróniája, hogy hosszú távon sokkal többe kerülhet a vállalatnak, ha a Linuxot választja. A számítástechnikai iparágat kutató cégek közül egyre több mutatta ki és egyre több ügyfél is dokumentálta, hogy a Linuxot választó vállalatok többet költenek a további szoftverekre, a munka- és konzulensi költségekre. Még ennél is fontosabb, hogy jelentős többletkockázatot vállal, aki üzletvitelhez nélkülözhetetlen alkalmazásait Linuxon futtatja: a szállító nem vonható felelősségre, a különféle disztribúciók nem kompatibilisek egymással, a Linux üzleti modellje hosszú távon nem gazdaságos.

A gazdasági kényszer arra készteti a vállalatokat, hogy kevesebből érjenek el többet, ugyanakkor úgy próbálják megoldani az üzleti problémákat, hogy ezzel előnybe kerülhessenek versenytársaikhoz képest. A Microsoft olyan platformot biztosít vásárlói számára, amelynek kiemelt célja, hogy minden módosítás nélkül megoldást nyújtson a fontos informatikai problémákra, javítsa a termelékenységet, és egyszerűbbé tegye az informatikai felügyeletet.

A legfontosabb alkalmazási területek kiszolgálása

A Microsoft úgy véli, hogy a platformot a legfontosabb alkalmazási területeket szem előtt tartva kell elkészíteni. A Windows termékcsalád tervezésekor olyan megoldás volt a cél, amely minimális kiegészítő szoftverekkel vagy azok nélkül ad megoldást a gyakori üzleti problémákra. Ez a szemléletmód gyors bevezetést tesz lehetővé, így vásárlóink gyorsabban juthatnak piacra versenytársaiknál. A platform a következő alkalmazási helyzetek kezelésére alkalmas:

- Informatikai infrastruktúra kiépítése: a vállalaton belül minden felhasználói fiók, jog és erőforrás felügyeletére módot adó egységes, integrált címtárszolgáltatás.
- Szoftveripari szabványokon alapuló integrált biztonsági funkciók, amelyek garantálják a fiókok, az erőforrások és az eszközök biztonságát, a vállalat vagyonának védelmét biztosító hitelesítési lehetőségek.
- Beépített kommunikációs lehetőségek, amelyekkel a gazdasági szakemberek út közben és otthonról is biztonságosan érhetik el vállalati e-mail fiókjukat, adataikat és az üzleti alkalmazásokat.
- Magas szintű együttműködés a portálalkalmazásokkal, az irodai csomagokkal, a webszolgáltatásokkal és a folyamatos átvitelű multimédiás lehetőségekkel való integrációnak köszönhetően.
- Az infomunkások hatékonyságának fokozása, nemzetközi szintű, az ügyfelek/kollégák részvételével zajló online értekezletek és információmegosztás révén.

A termelékenység maximalizálása

A Windows Server több száz üzleti alkalmazást támogat, beépített eszközökkel könnyíti meg az informatikai felügye-

letet és a végfelhasználók támogatását, és beépített alkalmazásokkal járul hozzá a termelékenység maximalizálásához. A Windows szabványos kezelőfelületét világszerte több millió végfelhasználó és fejlesztő ismeri. Hosszú idő óta a Windows jelenti a mércét a használat egyszerűsége, a testi fogyatékosoknak nyújtott kisegítő lehetőségek, illetve az alkalmazások kompatibilitása terén. További funkciói közül megemlíthetjük a nagyvállalati használatra kész címtárszolgáltatást, amely a hálózat védelmének és erőforrásainak (pl. a tárolókiszolgálók) kezeléséhez szükséges. Tartalmaz asztali gépek felügyeletére szolgáló eszközöket, biztosítja az utazó és a távoli felhasználók munkafeltételeit, valós idejű kommunikációs funkciói segítségével pedig virtuális értekezletek tarthatók, illetve otthonról és a külső telephelyen végzett munka közben is egyszerűen elérhetők a számítógépek és az adatforrások. Az asztali gépeken a Windows XP egyszerűbbé teszi a leggyakoribb műveleteket, például a fájlok és a mappák kezelését, az asztalnak az egyéni munkastílushoz és ízléshez való hozzáigazítását, valamint a futó alkalmazások felügyeletét. A Windows XP-nek köszönhetően több felhasználó is dolgozhat ugyanazon a számítógépen, saját jelszóval és saját mappáik biztonságos elérésével. Kidolgozott fájlvédelmi eljárásai révén kiküszöbölhető a rendszerfájlok felülírása, a rendszer-visszaállítási funkcióval pedig, ha a szükség úgy kívánja, eredeti állapotába állítható vissza a számítógép. A Windows 2000 Server, a Windows Server 2003 és az Active Directory révén a rendszergazdák egyszerűen zárolhatják az asztali környezetet, megakadályozva ezzel a felhasználókat abban, hogy megváltoztassák a beállításokat, és hogy engedély nélkül alkalmazásokat telepítsenek, illetve az egész vállalatot átfogó módosításokat foganatosíthatnak a biztonsági szabályokban.

Egyszerű bevezetés és felügyelet

Az új operációs rendszerek, illesztőprogramok, alkalmazások és javítások telepítése a rendszeresen előforduló feladatok közé tartozik, bármelyik operációs rendszerről legyen is szó. A Microsoft Systems Management Server (SMS) 2003-nak köszönhetően jelentős előrelépés következett be a javítások és a programok Windows Server 2003-ra történő telepítésében. Az SMS 2003 a Microsoft platform teljes körű változás- és konfigurációkezelési megoldása, amelynek segítségével a szervezetek gyorsan és költséghatékonyan biztosíthatják a szükséges szoftvereket és frissítéseket a felhasználók számára. Az SMS 2003 az alkalmazásbevezetés módját is továbbfejleszti: megbízhatóan és egyszerűen biztosíthatók vele a nélkülözhetetlen üzleti és irodai alkalmazások a felhasználók számára, akkor és ott, ahol szükségük van rájuk. Segítségével a Microsoft Windows környezet biztonsági javításainak kezelése is megoldódik: a bevezetés hatékonyságát megkönnyítő egyéb elemei mellett célzottan képes eljuttatni a számítógépekre a frissítéseket.

A Microsoft hatalmas összegeket áldozott és egy külön fejlesztőcsapatot jelölt ki arra, hogy egyszerűbbé tegye a Windows 2000 Server és a Windows Server 2003 felügyeletét. Számos beépített eszköz és szolgáltatás létezik, amelyek segítségével kivitelezhető a kiszolgáló távfelügyelete, a helpdesk alkalmazott átveheti a problematikus asztali gépek vezérlését, távolról megoldhatja a felhasználó problémáját. A felhasználók automatikusan tájékoztatást kapnak a szoftverfrissítésekről, és ha kérik, automatikusan telepíthetik azokat. A minimális felhasználói felülethez szokott UNIX-hívek kedvéért a Microsoft a gyakori felügyeleti műveletekhez parancssoros kezelőfelületet készített, amely be van építve a Windows XP és a Windows Server operációs rendszerekbe.

Jövőkép

A Microsoft egész története során azok közé a cégek közé tartozott, amelyek hatalmas összegeket kockáztatnak a jövőbeli technológia trendek kapcsán. A szoftvercégek sorában 2003-ben a Microsoft vezette a listát: több mind 6 milliárd dollárt és 35 ezer alkalmazottat szentelt a kutatás-fejlesztési céloknak.

A döntéshozóknak tisztában kell lenniük a kereskedelmi és a nem kereskedelmi szoftverek üzleti és fejlesztési modellje közötti különbségekkel. A Microsoft úgy véli, hogy a nem kereskedelmi szoftverek, éppen ebből a jellegükből fakadóan, nem lesznek képesek arra, hogy a Microsofttól megszokott gyorsaságú innovációval folyamatosan újdonságokkal jelenjenek meg a piacon. A Microsoft termékek sikere annak tulajdonítható, hogy létezik egy olyan szervezet, amely egyértelmű, előretekintő jövőképpel rendelkezik a termék fejlesztéséről, és amely kötelezettséget vállal arra az ügyfelekkel szemben, hogy valóra is váltja a jövőképet.

A birtoklás összköltsége

A legtöbb vállalat esetében a szoftverek bekerülési költsége jóval kisebb, mint a szoftver testreszabásával, bevezetésével, támogatásával és karbantartásával járó költségek. Sok informatikai projekt esetében a konzultációs költségek és a licenc költségek aránya 8:1 is lehet, tehát minden licencre fordított forintra nyolc forint konzultációs díj jut. Ehhez párosul még az az elterjedt szemlélet, hogy a projekteknek 12 hónapon belül meg kell térülniük, így egyre fontosabb, hogy a felhasznált szoftverek módosítás nélkül vagy kis módosításokkal futtathatók legyenek. A Microsoft azon az álláspontón van, hogy egy platform értéke nem mérhető pusztán a hagyományos birtoklási összköltséggel (TCO-val), ennél is fontosabb figyelembe venni a birtoklásból származó nagyobb üzleti értéket.

Vezető elemzők felhívják arra a figyelmet, hogy az operációs rendszerek esetében az ár mindössze kis hányadát, nem egyszer kevesebb mint 10 százalékát teszi ki a teljes birtoklási költségnek (TCO). Számításba kell venni a munkaerő-felvétellel, kiképzéssel és a konzultációs szolgáltatásokkal járó költségeket is. Az öt évre vetített birtoklási összköltség kiszámításakor a felügyeletet, az üzemeltetést, az alkalmazásfejlesztést, a támogatást és a szoftverfrissítések kezelését kell figyelembe venni.

A jobb összevethetőség érdekében a Microsoft az International Data Corporation (IDC) piackutató céget kérte fel, hogy az ügyfelek megkérdezése révén számítsa ki a Linux és a Windows vállalati használatának öt évre vetített valós költségeit. A vizsgálatban öt tipikus alkalmazási terület birtoklási összköltségét vetették össze. Ezek a következők voltak: webkiszolgálás, biztonság, fájlmegosztás, nyomtató-kiszolgálás, hálózati infrastruktúra. A felmérés több mint 100 vállalkozásra terjedt ki. Figyelembe vették a költségek között a hardver és a szoftver bevezetéséből és

állásidejéből származó költségeket, a kiszolgálók üzemeltetéséhez szükséges főállású alkalmazottak számát, valamint a személyi állomány kiképzésének költségeit. Összességében a Windows az öt közül négy alkalmazási területen (a webkiszolgálás a kivétel) kisebb birtoklási összköltséggel üzemeltethető. A vizsgálat legfontosabb eredménye az volt, hogy a Linux jóval több munkaköltséget vonz. Összességében a Linux háromszor annyi támogatási dolgozót igényelt, mint a Windows. A Linux esetében több fejlesztőre volt szükség a hálózat, a kiszolgálók, az asztali gépek, a helpdesk és az alkalmazásfejlesztés kezeléséhez.

A Gartner tanulmánya az asztali gépek birtoklási összköltségéről

Az IDC említett vizsgálata a kiszolgálókra összpontosított, azonban az asztali gépek esetében is előnyösebb a Windows a Linuxnál. Ha tökéletesen szeretnénk eljárni, párhuzamosan be kellene mutatnunk olyan vállalatok tapasztalatait, ahol a Linux, és olyanokét, ahol a Windows XP van telepítve az asztali gépekre. Ez idáig azonban nem tudunk olyan vállalatról, amely nagy számban vezetett volna be Linuxot az asztali számítógépekre, és hajlandó lett volna vizsgálatnak vetni alá magát. A Microsoft azonban felismert bizonyos trendeket a Gartner TCO-modelljén alapuló vizsgálatok alapján.

A Windows 2000 és a Windows XP, valamint az Office XP alkotta platform, illetve a Linux és az Open Office (a Star Office ingyenes verziója) birtoklási összköltségét összehasonlító vizsgálat azt mutatta ki, hogy a Microsoft választásával évente és számítógépenként körülbelül 30 százalékos megtakarítás érhető el. A megtakarítások jelentős része a Windows XP beépített felügyeleti funkcióból, illetve a kevesebb szervizelés miatti leállásból származik. Konkrétan a Linuxra és az Open Office-ra vonatkozóan a következőket tartalmazza a jelentés: *„Hiányoznak a nagyszámú munkaállomás kezelésére alkalmas standard szolgáltatások, például a címtár alapú felügyelet (a Novell NDS és a Windows 2000 vagy Windows 2003 Active Directory megfelelője). Minthogy a fent említett funkció hiányzik, és mivel a rendszerfelügyelet nem a szabványos infrastruktúra része, ezért egymást átfedő funkciójú eszközöket kell megvásárolni (például a Tivolit), illetve a több különböző környezetből fakadóan később nőnek a felügyeleti költségek."* (Pohjala, Janne, „Gartner TCO Study for European City Government," 2001. december).

Alkalmazáskompatibilitási problémák

A nagyvállalatok számítástechnikai környezete jellemzően összetett és heterogén. Az informatikai cégek közötti választási lehetőségek és a megszerezhető érték maximalizálását szolgálja, ha sok olyan független szoftvergyártó és rendszerintegrátor vállalat létezik, amely alkalmazásokat készít az adott cég platformjára, és támogatja azokat. További érték származik abból, ha ugyanezek a szoftvergyártók és rendszerintegrátorok tartósan áldoznak a termékek továbbfejlesztésére és támogatásra, meghosszabbítva ezzel ezeknek a platformoknak az élettartamát és gazdaságosságát. A Gartner 2003 márciusi tanulmányában („Linux Operating System Technology: Prospective") a következő áll: *„Bár folyamatban van az alkalmazások Linuxra történő átalakítása („portolása"), arra nézve nincs garancia, hogy mindegyik Linux disztribúción futni fognak.."*

Vállalati támogatás és szolgáltatások

Az informatikai platform kiválasztásakor fontos szempont a hozzáértő, szakszerű támogatás elérhetősége. Egy jól működő szoftveres ökoszisztéma garantálja, hogy a vállalatok igenlő választ kapjanak, amikor azt kérdezik, elérhető lesze a segítség, ha szükség van rá, vagy hogy hozzájuthatnak-e a versenyképességet biztosító új rendszerek költséghatékony telepítésére módot adó szolgáltatásokhoz. A Microsoft már több mint 20 éve tekinti stratégiai céljának, hogy termékei támogatására szoftveres ökoszisztémákat építsen ki. Egész iparágak nőttek fel a Microsoft termékei körül. További támogatási formák:

- Legnagyobb számú független szoftvergyártó. A Microsoft alkalmazásokat támogatja a világon a legtöbb független szoftvergyártó, akik tanúsítással rendelkező egyéni alkalmazásokat készítenek Windows rendszerre, széles körű választékot kínálva a Microsoft ügyfeleinek.
- Eszköztámogatás. A Windows alapkiépítésben több mint 19 000 eszközt támogat, és további 41 000 eszköz van tesztelés alatt.
- Tanúsított megoldások. A Windowshoz több ezer tanúsított hardver-illesztőprogram és szoftver szerezhető be külső szoftvergyártóktól, így egyszerűen bővíthető a rendszer új hardverekkel és alkalmazásokkal.
- Szolgáltatásbőség. A Microsoft termékeit több mint 450 ezer MCSE (Microsoft Certified Systems Engineer) mérnök támogatja világszerte, ezenfelül 750 ezer forgalmazó és partner kínál bőséges választékot.

Összegzés

A Linux operációs rendszer ugyan ingyen vagy olcsón beszerezhető, mégis vállalati ügyfelek esetében kisebb a birtoklási összköltsége, ha Windows fut a kiszolgálón és az asztali gépeken. A fenti vizsgálatok nem veszik figyelembe a Linux és a Windows közötti választásból fakadóan elvesztett lehetőségeket és egyéb kvalitatív tényezőket. A nagyvállalati szintű támogatást, az alkalmazások rendelkezésre állását és egyéb kulcsfontosságú összetevőket is felölelő stabil ökoszisztéma hiánya azt jelenti, hogy a Linux egyelőre nem rendelkezik azokkal a jellemzőkkel, amelyeknek köszönhetően hosszú távon is költséghatékony megoldást jelenthetne a vállalatok számára.

A cikkben szereplő URL-ek:

[1] http://www.forrester.com/home/0,6092,1-1,FF.html
[2] http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2878232-3,00.html

(Cikkünk *„A két üzleti platform kiértékelése: a Linux és a Microsoft Windows összehasonlítása"* c. tanulmány tömörített változata. A teljes tanulmány elolvasható a www.microsoft.com/hun/getthefacts címen.)

Módszertanok, módszertanok, módszertanok

A szoftverfejlesztési projektek olyan speciális tulajdonságokkal rendelkeznek, amelyek miatt sajátos szakmai és menedzsmentmódszerekre van szükségünk. Ezeket járjuk körül ebben a cikkben, különböző gyakorlati szempontok figyelembevételével.

Egy – véleményem szerint – mindenki számára ismerős történettel szeretném kezdeni. Képzeljünk el egy fiatal házaspárt, Mr. Férjet és Mrs. Feleséget. Szükségünk van továbbá Anyucira (ő a férj édesanyja), illetve Anyósra (értelemszerűen a feleség édesanyja). Az apák csendes megfigyelői a történetnek. Tegyük fel, hogy egy szép napon a Férj arra ébred, hogy sajtlevest szeretne enni vasárnap ebédre. A Feleség még soha életében nem főzött sajtlevest (képzeletbeli párunk még friss házas), de lelkesen nekiáll a szakácskönyvekben keresgélni, és talál is néhány receptet. Kiválasztja a legszimpatikusabbat, bevásárol, és megfőzi élete első sajtlevesét. Nagy örömmel tálalja délben az ebédet, ám férje az első falatok után megrökönyödve tolja el magától a tányért, és sértődött hangon közli: ez bizony nem sajtleves, Anyuci bezzeg milyen finomat tud főzni.

Az informatikai beszállítók módszertanai különbözők lehetnek, így az általuk képviselt minőségben is lehetnek eltérések. Bölcsek Köve azonban nem létezik, hiába is keressük: a sikerre nem létezik biztos recept.

A Feleség sírva hívja édesanyját, Anyóst, hogy elpanaszolja, milyen szerencsétlenül sült el próbálkozása. Anyós igyekszik őt vigasztalni, és sokéves tapasztalatai alapján azt tanácsolja: „legközelebb száraz fehérbort használj, és pár dekányi füstölt sajtot is tegyél bele, az adja majd a zamatát, apád is így szereti".

Következő héten egyik este Feleség korábban megy haza munkából, hogy férjét meglepje, természetesen igazi jó sajtlevessel. Anyós tanácsait megfogadva, mindenre odafigyelve készíti el az ételt, és Férjet sikerül is meglepnie. Az öröm egészen addig tart, míg Férj meg nem kóstolja. Most már nem mer a vasárnapihoz hasonlóan indulatos lenni, ezért finoman próbálja meg közölni: bizony ennek a sajtlevesnek még mindig vannak hiányosságai. De sebaj, szombatra úgyis Anyucihoz ígérkeztek ebédre, majd ott meglátja Feleség, mi fán is terem ez a remek étel.

A szombat el is érkezik, és Anyuci valóban remekel: a Férj olyan jóízűen eszik, mint már rég, Feleség is érzi, hogy a leves nagyon finom, de nem tudja eldönteni, mi adja ezt a különleges zamatot. Ebéd után tehát szorgalmasan segít Anyucinak a mosogatásban, és próbálja kifaggatni, mi a titka. Anyuci jóindulatúan elmeséli, hogy a sajt szétolvadása után még 5 percig kell főzni, utána leszűrni, és a végén némi snidlinget szórni a tetejére. Feleség szorgalmasan megjegyez mindent, és megígéri Férjének, hogy következő hét végén már nagyon finom sajtlevest főz ő is. Hogy biztos legyen a siker, hét közben egyik nap ismét korábban hazamegy, és anyósa, Anyuci utasításai alapján főz egy adag sajtlevest. Az ő megítélése szerint ez tényleg ugyanolyan, mint a szombati volt, ezért megnyugszik: hét végén végre tényleg sikerül örömet szerezni drága Férjének.

Elérkezik a vasárnap, Férj már várja az ebédet, bár csendben némi fenntartásai is vannak azzal kapcsolatban. Feleség szorgalmasan és lelkesen sürög-forog a konyhában, és az eredmény: finom, gőzölgő sajtleves. Pontosabban: a leves természetesen finom, és Férj végre meg is eszik egy egész tányérral, de még mindig ott a megjegyzés az ebéd végén: ez még mindig nem az igazi…

És mindannyian tudjuk: soha nem is lesz az.

De vajon miért?

A sajtleves titka

Egészen biztosak lehetünk abban, hogy a történet valamennyi női szereplője remek szakácsnő, imád főzni, és remek sajtlevest tud készíteni. Mindegyiküknek megvan a sa-

ját *módszere* arra, hogy milyen összetevőkből és hogyan kell készíteni, miből mennyi kell, milyen hosszan kell főzni, mikor kell leszűrni stb. Kívülálló számára talán nem is feltűnő közöttük a különbség, egy szakértő viszont (a Férj) pillanatok alatt megérzi, ha nem a kedvencét kapja.
Mi lehet ennek az oka? Miért nem tud a Feleség, az Anyós és az Anyuci ugyanolyan levest főzni? Miért nem lesz a Feleség levese soha ugyanolyan, mint Anyucié?
A különbségek elég sokrétűek: a nők egyénisége, a konyha felszereltsége, a tűzhely típusa (gáz, elektromos, kerámialapos stb.), a rendelkezésre álló fűszerek, azok frissessége, minősége mind-mind fontos lehet. Összességében tehát azt mondhatjuk: a legfőbb befolyásoló tényezők a háziasszonyok *adottságai* és a megvalósítás *környezete*.

Az „IT-leves"

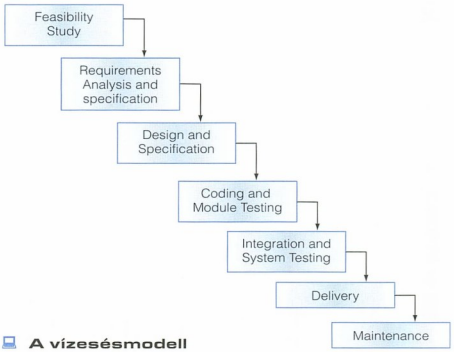
Szép, szép, de hogy kerül a sajtleves egy informatikai cikk bevezetőjébe? Akik olvasták „Az UML és a kosárlabda" című írásom **([1])**, azok már gyanakodhatnak: ismét valamilyen hasonlatról van szó.
A fenti történetet valóban egy informatikai folyamat szemléltetésére szántam: a sajtleves egy tetszőleges szoftverterméket szimbolizál, az asszonyok a szállítók, a Férj pedig a megrendelő. Lássuk tehát a fenti történetet ennek fényében.
A Férj az Anyuci által képviselt minőséghez szokott, azt szeretné kapni újabb beszállítójától, a Feleségétől is. Ő azonban többszörös hátrányban van Anyucihoz képest: sem a Férj pontos igényeit nem ismeri, sem Anyuci módszerét, amellyel a végterméket előállítja. Saját ismeretei, képességei és lehetőségei szerint a legjobbra törekszik, a kívánt minőséget azonban nem éri el.
Ekkor segítségért folyamodik, szakértőket von be, akik jól bevált módszereket ajánlanak. Ezek a praktikák már bizonyítottak, hiszen Apuci és Após is boldogok, és imádják feleségük sajtlevesét. Vagyis a szakértők valós tapasztalataik alapján adják a tanácsot, amely már bizonyított, amely segítségével több sikeres projektet lebonyolítottak már. Tekintve, hogy folyamataik mind tudatosak és jól szervezettek, pontosan meg tudják adni a pontos receptet, ami alapján ők – saját körülményeik között – dolgoznak.
A Feleség nevű beszállító igyekszik mindkettőjük tanácsát megfogadni (bár azok különbözőek). A dolog azonban mégsem működik: nem sikerül az adott minőséget előállítani, nem sikerül megfelelni az elvárásoknak. A bevált módszerek kudarcot vallanak. A hiba oka ott keresendő, hogy az egyes alvállalkozók körülményei mind mások, és ezt a különbséget nem szabad figyelmen kívül hagyni.
Ha az Anyuci gáztűzhelyen főz, ne akarjuk ugyanazokat az elkészítési időket tartani kerámialapos főzőlapunkkal. Ha kertjében frissen terem a petrezselyem, ne akarjuk a szárított ételízesítővel ugyanazt az ízhatást elérni. Ha a tanácsadó egy 200 fős cég alkalmazottja, 20 személyes kisvállalatunknál ne akarjuk ugyanazt az eredményt elérni.

Mit tehetünk akkor?

A megoldás: testreszabás. Ha az Anyuci azt mondja, öt percig főzzük, és utána szűrjük le a levest, vegyük figyelembe, hogy neki más a tűzhelye, más sűrűségű szűrője van, másmilyen nagyságú darabokra szokta szelni a hagymát stb.
Ennek megfelelően lehet, hogy nálunk hét perc főzés kell még, és talán nem is kell leszűrnünk a levest, ha a hagyma kellően szétfőtt eközben.
Ha a tanácsadónk azt mondja, hogy ez a projekt négy, jól képzett szakemberrel oldható meg, de nekünk tíz emberünk is van rá, akik viszont kevésbé képzettek, meg kell fontolni, hogy ez elég-e a megvalósításhoz, kevés, vagy épp még túl sok is.
Véleményem szerint mindenképp érdemes a „nagyok", a hozzáértők módszereit tanulmányozni, hiszen rengeteg alapelem kinyerhető, rengeteget lehet tanulni belőlük, és hihetetlen mennyiségű ötlettel gazdagodhatunk. Egyiküknél sem fogjuk megtalálni a bölcsek kövét, de irányt adhatnak arra vonatkozóan, hogy merre érdemes elindulni.
Ennek megfelelően nem célom tehát egyik módszertan mellett sem letenni a voksot. Vannak nagyon jó, átgondolt, összeszedett metodikák, ám ezek kreatív gondolatok és egyéni ötletek, a probléma és a vállalat átfogó látása nélkül semmit nem érnek.
A következő sorokban ennek megfelelően néhány közismert és már bizonyított módszertant szeretnék felvázolni, mégpedig abból a szemszögből, hogy melyik mit nyújthat egy informatikai projekt számára, esetleg némi utalással arra, melyiket mikor érdemes alkalmazni. Még egyszer szeretném azonban hangsúlyozni, hogy a „Bölcsek köve" nevű módszertan nem létezik, még ha össze is hasonlítunk metodikákat különböző szempontok szerint, mindig az adott probléma, az aktuális körülmények, tényezők határozzák meg a folyamatok pontos menetét, így az itt leírtak is csak egyfajta homályos körvonalai ezeknek a dolgoknak.

A Microsoft Solutions Framework

Az MSF a Microsoft módszertana **[2]**, amelynek célja sikeres üzleti-informatikai projektek lebonyolítása. Középpontjában a csapat- és a folyamatmodell áll, amelyet a projektvezetésre, a kockázatkezelésre és a készenlétre vonatkozó úgynevezett diszciplínák, ajánlások egészítenek ki.
Az MSF folyamatmodellje a vízesés- és a spirálmodell előnyeit egyesíti. Megtartja a vízesésmodell mérföldköveit, amelyek a projekt egy-egy fontosabb állomását jelképezik.



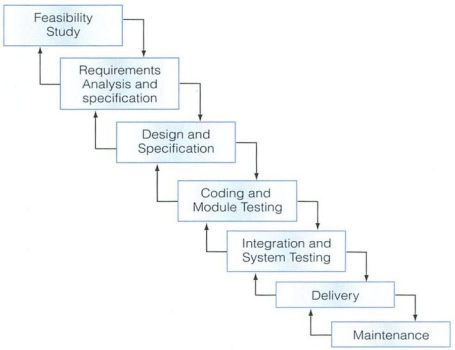
A vízesésmodell

A *vízesésmodell* esetében úgy kell elképzelni a folyamatot, mintha egy valódi vízesésen zúdulnánk lefelé, és egy-egy nagyobb kövön fennakadva a projekt egy következő fázi-

sába lépnénk át. Ezek a *mérföldkövek,* amelyek a projektfolyamat egy-egy állomását jelentik: egy fázist zárunk le, a termék eljut egy bizonyos állapotba, a szükséges dokumentációk elkészültek stb.

A sajtleves példájára visszatérve: egy-egy mérföldkő lehet a recept tanulmányozása, a bevásárlólista összeállítása, az alapanyagok beszerzése, a főzés maga, illetve a tálalás. Ebben az esetben a projekt egy egyenes mentén felvázolható, a mérföldkövek között nincs visszacsatolás sem: ha utólag jut eszünkbe, hogy például fehérbor nincs otthon, semmi lehetőségünk a korrigálásra, nem mehetünk vissza a boltba. A projekt egész egyszerűen meghiúsul, a Férj éhen marad.

Viszonylag hamar felismerték a visszacsatolás szükségességét, és megszületett a visszacsatolásos vízesésmodell. Ennek lényege az előzőhöz képest, hogy az egyes mérföldkövek között már tudunk „viszszafelé" is lépni, azaz ha az egyik lépés során kiderül, hogy valamit elhibáztunk, van lehetőség azt korrigálni, a korábbi fázisokra vonatkozva is.



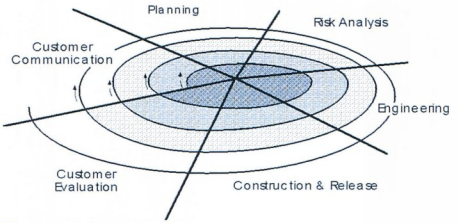
A visszacsatolásos vízesésmodell

Ez a modell tehát arra ad lehetőséget, hogy az előző lépésre visszatérjünk, ha ez szükségesnek bizonyul. Vagyis ha elfelejtettünk fehérbort vásárolni, lehetőségünk van visszalépni, felvenni azt bevásárlólistánkra, újra elmenni a boltba (az új listával), és pótolni a mulasztást.

Jól látható, hogy ez a modell már valamivel rugalmasabb, ám még mindig rengeteg hiányossága van.

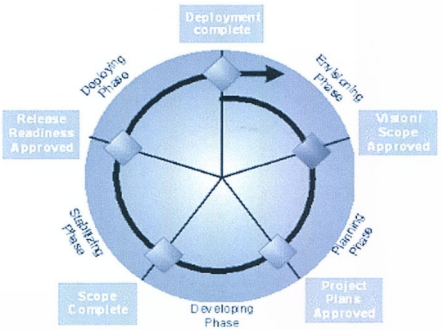
Ezen hiányosságok, problémák áthidalására jelenthet megoldást a spirálmodell alkalmazása, mely a 80-as évek második felében született, Barry Boehm munkásságának köszönhetően. Alapötlete, hogy a szoftverfejlesztést nem egyetlen egyenes mentén képzeli el, mint a vízesésmodell, hanem spirálvonal formájában: az egyes tengelyek a különböző fázisokat jelentik, a spirál aktuális sugara (origótól mért távolsága) a rendszer fejlettségi fokát tükrözi.

A fejlesztési folyamat egyes lépései mind pluszt adnak a majdani termékhez, ezért nő a spirál sugara folyamatosan. A körív pedig azért tér vissza újra a kiindulási ponthoz (természetesen a már megnövelt sugárral, tartalommal), mert a fejlesztési folyamatot ezúttal nem a projekt lezárásáig, az első verzió átadásáig tekintjük, hanem feltételezzük, hogy az újabb felhasználói igények, a fejlesztés és a használat során felgyülemlett tapasztalatok és a különféle egyéb körülmények szükségessé teszik újabb verziók, újabb változatok fejlesztését. Így egy szoftver átadása nem jelenti feltétlenül a folyamat lezárását, újabb és újabb fejlesztések következhetnek.



A spirálmodell

Az MSF ezt a két modellt igyekszik egyesíteni: a spirálmodell inkrementális tulajdonságához hozzáadja a mérföldkövek által nyújtott előnyöket, vagyis egy-egy fázis végén fel kell mérni a helyzetet: mi volt a cél, és abból mit sikerült megvalósítani. Ha a további tervekben módosításra van szükség, azokat is itt tehetjük meg, illetve a projekt lezárására, megszüntetésére is lehetőségünk adódik.



Az MSF inkrementális modellje a mérföldkövekkel

A fenti ábrán jól látható, hogy az MSF folyamatmodellje az alábbi fázisokból áll:

- *Elképzelés (Envisioning):*
  Korai, kezdetleges tervezés, az üzleti igények körvonalazása. A fázis végére ismerjük a fő kockázati tényezőket, és kiválasztásra kerülnek a projekt kulcsfontosságú emberei.
- *Tervezés (Planning):*
  Kidolgozzuk a rendszer funkcionális specifikációját, megtervezzük a folyamatokat, elkészítjük a költségbecslést és a határidőket. Mindezeket úgy kell meghatározni, hogy azok a megrendelőnek és a projektcsapatnak egyaránt elfogadhatók legyenek.
- *Fejlesztés (Developing):*
  A fejlesztési fázis nemcsak a klasszikus értelemben vett kódolást takarja, hanem a tervek véglegesítését, a tesztelési terv és a tesztesetek kidolgozását, valamint a telepítőkészlet elkészítését is.

- *Véglegesítés (Stabilizing):*
  Ebben a fázisban véglegesítjük az adott verziót, hogy az szállításra kész legyen: nemcsak magát a szoftvert, hanem a hozzá tartozó dokumentációkat, kiegészítéseket is lezárjuk.
- *Telepítés (Deploying):*
  Az elkészült, letesztelt programot, a hozzá tartozó komponenseket és kiegészítéseket telepítjük a megrendelő gépeire, elvégezzük a szükséges beállításokat. A felhasználókat betanítjuk a rendszer használatára, hozzáférhetővé tesszük a különböző tanfolyami anyagokat, dokumentációkat számukra. Ezzel párhuzamosan az üzemeltetést is felkészítjük a rendszer átvételére.

Az MSF másik nagy (véleményem szerint a nagyobbik) ütőkártyája a csapatmodell: alapkoncepciója ugyanis az, hogy hat, egymással egyenrangú szerepkört képzel el. Ezen szerepkörök mind azonos súllyal vesznek részt a megvalósításban, és azért fontos a megkülönböztetésük, mert érdekeik gyakran egymásnak ellentmondásosak is lehetnek. Természetesen nem feltétlenül szükséges, hogy a hat szerepkört hat ember képviselje, kisebb csapatoknál lehetőség van bizonyos összevonásokra is, ám ekkor is figyelembe kell vennünk a lehetséges konfliktusokat. A döntések ennek megfelelően közös megegyezés alapján, konszenzussal (tehát nem egyszerű többséggel!) születnek.
Az MSF csapatmodellje az alábbi szerepköröket tartalmazza:

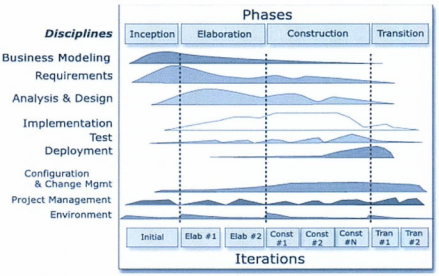
- *Termékmenedzsment:*
  feladata az ügyfél érdekeinek képviselete, ő az úgynevezett „szóvivő", a megrendelő kezelője,
- *Programmenedzsment:*
  arról gondoskodik, hogy a projekt időre befejeződjön
- *Fejlesztés:*
  tervezi és készíti („kódolja") a szoftvert,
- *User experience:*
  a felhasználó szóvivője, a felhasználói felület (UI) tervezője,
- *Release menedzsment:*
  a termék szállításáért felelős.

Mint már említettem, e szerepkörök a projekt során mind egyenrangúak. Munkájuk során mindig szem előtt kell tartani, hogy a sikeres projekt legfontosabb ismertetőjegye az elégedett ügyfél, így mindig arra törekedjünk, hogy őt a lehető legjobban vonjuk be a munkába. Vegyen részt a tervezésben, a kockázatok azonosításában, lássa a készülő felhasználói felületet stb.
Munkánk eredményére, bármi legyen is az (szoftver, szolgáltatás vagy bármi más), mindig termékként tekintsünk, és annak minél jobb minőségéért a lehető legtöbbet tegye meg valamennyi csapattag. Ha mindenki minőségi munkát végez, a termék minősége is jobb lesz (optimális esetben). Mindemellett legyünk nyitottak az új dolgokra, a problémák újszerű megoldásaira, legyen meg bennünk a hajlandóság és az igény a tanulásra. Az informatikai szakterületen nem kell hangsúlyozni a naprakész tudás fontosságát, de sajnos még mindig nagyon sokan vannak, akik nem helyeznek erre kellően nagy hangsúlyt.
Mindezek eléréséhez nagyon fontos a csapat megfelelő motiválása: sajnos a jobb minőség több munkával jár, ezt pedig (főleg eleinte) általában nem fogadják kitörő lelkesedéssel munkatársaink. A motivált emberek, csapatok jobb minőség előállítására képesek, hiszen van „miért" dolgozniuk, a sajátjuknak érzik a problémát. A motiváció elsődleges formája napjainkban a pénz, azonban rengeteg egyéb lehetőségünk is van: közös kirándulások, vacsorák, sportesemények stb. Az emberek általában szeretik, ha „törődnek" velük, és elégedettségüket nem csupán bankszámlájuk vastagságában mérik.

Az MSF fenti két modellje (folyamat- és csapatmodell) az őket kiegészítő ajánlásokkal egy jó keretet adhat munkánknak. Vigyázzunk azonban, hiszen mindez csupán egy nyomvonalat, egy ösvényt határoz meg számunkra, amelyről letérhetünk, választhatunk másik útvonalat, majd ha úgy gondoljuk, visszatérhetünk az MSF-hez. Tipikus ilyen „letérés" szokott lenni az MSF folyamatmodelljének RUP-pal vagy más módszertannal történő kiegészítése, esetleg helyettesítése.

A Rational Unified Process

A Rational Software fejlesztési módszertana **[3]** némileg eltér az MSF-től. Elsősorban a folyamatra koncentrál, a szoftver életciklusát az üzleti modellezéstől egészen a telepítésig felöleli. Használati eset vezérelt, iteratív és inkrementális fejlesztési módszertan. Mit is jelent ez?
Az inkrementális szó jelentését már láthattunk a spirálmodell bemutatásakor: nem egyszerre akarjuk kifejleszteni az egész rendszert, nem szánunk éveket a teljes megvalósításra, miközben a megrendelő elveszíti minden türelmét; ehelyett egyre bővülő funkcionalitással mindig átadható állapotban lesz a rendszerünk. Nem az a cél tehát, hogy minden funkciót egyszerre fejlesszünk, hanem az, hogy egy-egy újabb funkciócsoporttal gazdagodjon folyamatosan a szoftverünk.
Az iteratív megközelítés azt jelenti, hogy a fejlesztési fázisok több iterációból állnak össze. Minden fázisban különböző munkafolyamatokat (workflow) végzünk, amelyek arányát az alábbi ábra szemlélteti:



A RUP fázisai, munkafolyamatai és az iterációk

A RUP munkafolyamatai két nagy csoportba sorolhatók:

1. műszaki, szakmai folyamatok:

- üzleti modellezés,
- követelményelemzés,

- elemzés és tervezés,
- implementáció,
- tesztelés,
- telepítés;

2. támogató üzleti folyamatok:

- konfiguráció- és változáskezelés,
- projektmenedzsment,
- környezet kialakítása.

Értelemszerűen a műszaki folyamatok a megvalósításhoz kapcsolódó konkrét informatikai lépéseket takarják, míg a támogató funkciók a járulékos, nem szorosan szakmai, de legalább annyira fontos teendőket.

A fenti ábra nem szemlélteti ugyan, de RUP is a spirálmodell alapján építi fel modelljét, ami hasonló fejlődést tesz lehetővé, mint az MSF esetében. Attól való eltérést találhatunk viszont, ha a csapatmodellt, a szerepköröket vesszük szemügyre. A RUP ugyanis nem hangsúlyozza külön ezek jelentőségét: kitér ugyan arra, hogy kik lehetnek az egyes folyamatok résztvevői, de fontosságukat és a közöttük lévő kapcsolatokat, rendezettséget nem hangsúlyozza.

Ugyancsak eltérés, hogy a RUP a folyamatmodellbe „belemosva" tartalmazza azokat a támogató, kiegészítő teendőket, amelyeket az MSF külön ajánlásokként tárgyal. Ugyanakkor a Rational Software nagy hangsúlyt fektet a folyamatokat támogató szoftverek fejlesztésére is, amelyek most már gyakorlatilag a teljes életciklust lefedik.

A szoftverek fejlesztése során olyan speciális termék előállítása a cél, amelyet nem lehet párhuzamba állítani az autógyártással, a könyvkiadással vagy a fuvarozással. A szoftver olyan szellemi termék, amely sajátos tulajdonságokkal rendelkezik.

Egyéb módszertanok

Természetesen rengeteg módszertan létezik, amelyet különböző cégek, fejlesztők dolgoztak ki munkájuk megkönnyítése érdekében. Ezek közül egyet emelnék ki, amely az *Extreme Programming* (XP **[4]**).

Az XP a '90-es évek elején született, fő célja az ügyfél-elégedettség. Arra ösztönzi a fejlesztőket, hogy a megrendelői igények változásait mindig rugalmasan kezeljék, bármilyen későn jelentkezzenek is azok. Természetesen erős, összetartó, hatékony csapatra épít, és itt is kikötés, hogy a folyamatban részt vevő valamennyi ember elsődleges feladata minőségi szoftver szállítása. Ennek érdekében négy tényező kap különös hangsúlyt: a kommunikáció, az egyszerűség, a visszacsatolás és a bátorság. Az ügyfél elégedettségének érdekében nagyon fontos, hogy folyamatosan bevonjuk őt a munkába, visszajelzéseivel azonnal reagáljon már az első pillanattól kezdve. Így valóban az ő igényei és elvárásai szerint dolgozhatunk, amelyek akár változhatnak is az idő előrehaladtával: akár a technológia fejlődése miatt, akár egyéb, külső körülmények hatására, sőt az is elképzelhető, hogy a rendszer fejlődésével ráébred arra, hogy ő maga sem volt tisztában pontos igényeivel. Mindezekre fel kell készülni, bármilyen módszertan alapján dolgozunk is. Az XP erre a tényezőre helyezi a hangsúlyt, és ennek érdekében sorakoztatja fel ajánlásait.

Az XP-t talán egy óriási kirakójátékhoz (puzzle) lehetne hasonlítani: rengeteg apró elemből áll, amelyek önmagukban semmit nem jelentenek, együttesen azonban egy gyönyörű képet mutatnak. Sokaknak az XP-ről a párban történő programozás jut eszükbe, vagy az, hogy a megrendelőnek folyamatosan benn kell ülnie a fejlesztők között. Ezek azonban csak egy-egy szeletei a valódi XP-nek, amely ennél egy sokkal összetettebb, finomabb módszertan.

A fejlesztési projektek során joggal merülhet fel a kérdés: miért van szükség kifejezetten IT-módszertanokra, miért nem jók nekünk a klasszikus, kiforrott projektmenedzsment-módszertanok? Hiszen azok „univerzálisak", általánosan elterjedtek. Ha mindenkinek jók, nekünk miért nem? A válasz nagyon egyszerű: pontosan ez az általánosság az, ami miatt ezek a metodikák nem alkalmazhatók szoftverfejlesztés esetén: itt ugyanis olyan speciális termék előállítása a cél, amelyet nem igazán lehet párhuzamba állítani az autógyártással, a könyvkiadással vagy a fuvarozással. A szoftver egy olyan *szellemi* termék, amely sajátos tulajdonságokkal rendelkezik:

- a termék sokszorosítása nem okoz problémát, ellentétben azokkal a termékfejlesztésekkel, ahol az utángyártás folyamata is alapos tervezést, odafigyelést igényel,
- a hordozó, kézzel fogható médium értéke elenyésző a szoftver szellemi értékéhez képest,
- soha nincs két egyforma „gyártási" folyamat: a szoftverek mindig különbözőek, más körülmények között készülnek, így a fejlesztés mindig egyedi, sajátos tulajdonságokkal rendelkezik,
- a szoftverek jelentős része nem tömeges igényt igyekszik kielégíteni, hanem egyedi megrendelés alapján készül.

Mindez indokolttá teszi, hogy a speciális tulajdonságoknak megfelelően speciális módszertanokban gondolkozzunk.

Speciális problémák

A módszertanok mellett számos olyan probléma merülhet fel a fejlesztés során, amelyre korábban már dolgoztak ki megoldási javaslatokat. Rengeteg ilyen speciális ajánlás van, ezek közül csak néhányat ragadnék ki példaként.

Gondolom, a „Design *Pattern*" (DP) kifejezés mindenki számára ismerős. A DP-k gyakran előforduló problémákat írnak le, azok környezetével együtt. Olyan működő megoldási javaslatokat nyújtanak, amelyek gyakorlati tapasztalatok alapján hatékony választ adnak az adott kérdésre. Elsősorban a szoftver tervezésénél használatosak, amikor a felmerülő megrendelői problémákra keressük a megoldást.

Olyan sémák ezek, amelyek csak tanulással, tapasztalatszerzéssel bővíthetők. Célszerű egy saját vagy céges DP-gyűjteményt létrehozni, amelyben hatékonyan kereshetünk, és amelyet folyamatosan karbantartva egyfajta tudástárként használhatunk. Természetesen bizonyos idő után már a fejünkben is összeáll egy kép a leggyakrabban használt tervezési mintákról, ám ezek egyre bővülnek, és számuk is rohamosan nő.

Kérdés lehet, hogy milyen követelményeket állítunk rendszerünkkel szemben: hatékonyság, rendelkezésre állás, tanulhatóság stb. mind fontos lehet, ám gyakran kompromisszumokra kényszerülünk. Érdemes végiggondolni, mi az, ami igazán fontos, és mik azok a tulajdonságok, amelyekből engedni tudunk: egy telefonközpont 1/10 000 hibaaránya tűrhető, sőt jónak mondható, ám ugyanez például egy orvosi műszernél nem megengedhető. Egy bankkártya-leolvasó egységnél 5–10 másodpercet türelmesen kivár az ember, ám egy erőmű vezérlőszerkezeténél ekkora késleltetés végzetes lehet.

A dokumentáció fontosságát rengetegen hangsúlyozzák, mások egyáltalán nem tartják lényegesnek, sőt kifejezetten felesleges időtöltésnek gondolják. Véleményem szerint az egészséges megközelítés valahol félúton van: meg kell találni azt az egészséges arányt, ahol még nem esünk túlzásokba, viszont minden a kellő mértékben le van dokumentálva, a későbbiekben érthető, hozzáférhető iratokat találunk róla. Mindez nemcsak a kódokra, komponensekre vonatkozik, hanem az egyes tevékenységekre is: érdemes rögzíteni a folyamatok pontos menetét, illetve az egyes döntések hátterét is, mit miért csinálunk, milyen háttere, körülményei vannak a határozatoknak stb. Mindez a szakmai és az üzleti folyamatokra egyaránt érvényes. Célszerű kialakítani egy közös vállalati dokumentumtárat, ahol strukturáltan mindenki elhelyezheti saját iratait. Szerkezetének, felépítésének kidolgozása alapos megfontolást igényel, hiszen egyrészt szükség lehet projektenkénti nézetekre is, másfelől a vállalat felépítését, hierarchiáját tükröző szerkezetre is. Természetesen a megfelelő jogosultságkezelést is meg kell valósítani, hogy mindenki csak azokhoz a könyvtárakhoz/dokumentumokhoz férhessen hozzá, amelyek valóban kellenek munkájához.

Szintén a folyamatok támogatásához lehet hasznos különféle belső levelezőlisták létrehozása. Kis cégeknél (de akár nagyobbaknál is) gyakran megfigyelhető, hogy az egyes csapattagok úgy tartják egymással a kapcsolatot, hogy az e-mail címzettjei közé mindig felveszik a megfelelő személyeket. 8-10 név is szerepel a listában esetenként. Sajnos ennek a megoldásnak több buktatója is lehet. Egyrészt akarva-akaratlanul is előfordulhatnak olyan szituációk, amikor kifelejtünk valakit a címzettek közül, mert például újonnan érkezett a céghez, most kapcsolódott be a projektbe stb. Ugyanígy olyannak is elküldhetjük véletlenül, aki nem érintett, sőt rossz helyre is címezhetjük. Ez utóbbira lehet példa két azonos vagy hasonló nevű kolléga: annak idején egyik munkahelyemen `agnes.molnar` felhasználói névvel szerepeltem, amikor érkezett egy `agi.molnar`, természetesen egy egészen más részlegre, az enyémtől teljesen eltérő beosztásba. Nagyon gyorsan rá kellett szoknunk arra az algoritmusra, hogy ha egy e-mail tartalmát nem értettük, gondolkodás nélkül küldjük tovább a másiknak, illetve telefonon is már rutinból irányítottuk egymáshoz a hívásokat. Ugyanilyen problémát okozhat, ha például `amolnar` az azonosítóm, és a következő embernek, aki ezt kapná, már két betűt vesznek figyelembe a keresztnevéből stb. Így akarva-akaratlanul is hozzám érkeznek az `akmolnar`, `andmolnar` stb. címre szánt levelek, annak ellenére, hogy sem Ákos, sem pedig Andrea nem vagyok. Egy projekt keretein belül mindezen problémákat jól karbantarthatjuk naprakész levelezőlistákkal, amelyek beszédes nevet viselnek, és tagjai mindig az aktuálisan érintett csapattagok. Ráadásul a listák archívuma is mindig hozzáférhető, így az sem okozhat problémát, ha valaki véletlenül töröl egy levelet lokálisan vagy megtelt postaládája miatt. A levelezőlisták mellett létrehozhatunk céges blogokat is, ahová az aktuális tudásanyagot töltjük folyamatosan, a munkatársak aktív részvételével.

A dokumentációk készítésénél meg kell találnunk azt az egészséges arányt, ahol még nem esünk túlzásokba, viszont minden a kellő mértékben dokumentált. Mindez nemcsak a kódokra, hanem az egyes tevékenységekre, folyamatokra is érvényes.

Tovább segíthetik munkánkat a különböző verziókezelő- és tesztrendszerek, a követelmények felmérését támogató űrlapok, alkalmazások is. Célszerű lehet a projektek lezárásakor időt és energiát szánni az értékelésre és a leszűrt tapasztalatok dokumentálására is, hiszen így rengeteg tapasztalatot adhatunk át más csapatoknak, illetve a jövőbeli projektek számára.

Összességében elmondható tehát, hogy a különféle módszertanok sokat segíthetnek munkánkban. Ha jól alkalmazzuk őket, és céljaink elérése érdekében a megfelelő komponensekkel egészítjük ki, jó úton haladunk a siker felé. Ahhoz azonban, hogy eldönthessük, melyik módszertant akarjuk alapként beépíteni folyamatainkba, alapos, gondos előkészítésre és tapasztalt, képzett szakemberekre van szükség.

Molnár Ágnes
molnar.agnes@cleware.hu

A cikkben szereplő URL-ek:

[1] http://aghy.uw.hu
[2] http://www.microsoft.com/msf
[3] http://www.ibm.com/software/rational/
[4] http://www.extremeprogramming.org

Üzletiintelligencia-szoftverek

MS SQL 2000 Reporting Services III. 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).

A *Reporting Services*-t (továbbiakban *RS*) bemutató sorozatunk harmadik, és egyben befejező részéhez értünk. A korábbi részekben megismerkedtünk az architektúrával, a szerver felépítésével, a jelentéskészítés életciklusával, a *Visual Studio .NET 2003* által támogatott kimutatáskészítéssel, a *Report Manager* alkalmazással és a jelentéselérési módokkal. Ebben az utolsó részben főként azzal foglalkozunk, hogy miként lehet a korábban ismertetett lehetőségeket testre szabni, kiterjeszteni. Mindezek mellett szót ejtünk kész kimutatásaink telepítési kérdéseiről. Még mielőtt belekezdenénk, aki már rendelkezik telepített RS környezettel, mindenképpen frissítse a hozzá tartozó *Books Online* (továbbiakban *RSBOL*) súgót innen: **[1]**, mert rengeteg változás van benne az eredetileg települt példányhoz képest (persze az *MSDN Library online* oldalain is elérhetjük a legfrissebb állapotot).

Az RS.EXE segédeszköz

Ahogyan az *SQL Server 2000* eléréshez is megszületett egy olyan parancssori eszköz, amelynek paraméterében megadott kapcsolódási információk mellett (szervernév, felhasználónév, jelszó stb.) automatikusan bejutottunk egy olyan környezetbe, ahol közvetlenül kommunikálhattunk a szerverrel (ez volt az *OSQL.EXE*), úgy a *Reporting Server*hez is készült egy ilyen eszköz. A neve *RS.EXE*, és úgyszintén a *\Program Files\Microsoft SQL Server\80\Tools\Binn* könyvtárban található (alapértelmezésben az *SQL Server 2000* telepítő betette a *PATH* útvonalai közé), amennyiben a *RS* telepítésekor az adminisztrációs eszközök telepítését is kiválasztottuk. Ez az eszköz (ellentétben az *OSQL.EXE*-vel) csak scriptfuttatásra használható, amelyet bemeneti állományként adunk meg a paraméterek között. A *script*-fájl *unicode* vagy *UTF-8*-as kódolású szöveges állomány (kizárólag .rss kiterjesztéssel), és a nyelve csakis *Visual Basic .NET*. A segítségével adminisztrálhatunk egy adott *Report Server*-t, adatbázist másolhatunk egyik szerverről a másikra, publikálhatunk új kimutatásokat a szerverbe stb. A futtatásához helyi adminisztrátori jogosultság kell. A paraméterezése:

```
rs {-?} [-i input_file] [-s serverURL]
   {-u username} {-p password} {-l time_out}
   {-b } {-v globalvars} {-t }
```

Mint látható a paraméternevekben, kívülről adhatunk értéket a *script*-ben szereplő globális változóinknak:

```
rs -i Script.rss -s http://servername/reportser-
ver -v report="Company Sales"
```

Érdemes megnézni a példakimutatások szerverbe való publikálásához használható *\Samples\Scripts\ Publish SampleReports.rss* állományt is. Célszerű megkeresni a *script*-ben a *PublishReport()* függvényhívásokat, mert két példakimutatás hiányzik a végéről („*Foodmart Sales*", „*Product Line Sales*"), amit a teljesség kedvéért nem árt kipótolni tanulás gyanánt (persze ez utóbbi két kimutatás működéséhez egy telepített *OLAP FoodMart2000*-es példaadatbázisa is szükséges). A *script* kötelező belépési pontja a *Main*() eljárás:

```
Public Sub Main()
   ' Itt kezdődhet a saját kódunk
End Sub
```

Persze újabb saját eljárásokat és modulszintű változókat is alkalmazhatunk benne. A *script* automatikusan olyan környezetben fut, ahol már létezik a kapcsolat a *Report Server*-rel (létezik már egy *web proxy* osztály), amelynek példányára egy „*rs*" objektumváltozón keresztül rögtön hivatkozhatunk is (pl. *rs.Credentials, rs.CreateFolder(), rs.CreateDataSource(), rs.CreateReport()*). Névtereket nem szabad importálni, a következő névterek már eleve létez-

nek: *System.Web.Services*, *System.Web.Services.Protocols*, *System.Xml*, *System.IO*. Részletesebb információkról, további parancssori eszközökről (mint az *rsconfig*, *rskeymgmt*, *rsactivate*) a *RSBOL*-ban olvashatunk: **[2]**.

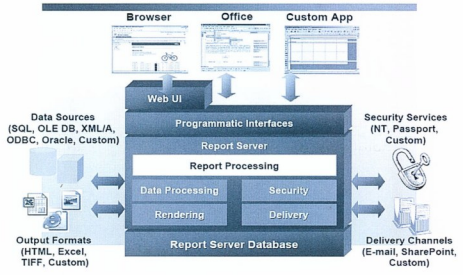
Kimutatások telepítése

Felmerülhet mindenkiben a kérdés, hogyan jut el a fejlesztett kimutatásunk az ügyfél adatbázisába. Az első megoldás a fent említett *script* megírása oly módon, hogy kívülről vegyen át dinamikus paramétereket (mint pl. a kimutatás publikált neve, forrás és célútvonala stb.). Ezzel a megoldással elkészíthető egyetlen jól megírt (hibákat kezelő) *script*, majd egy *batch*-fájlból vagy akár egy verziócserélő kliensalkalmazás *commandshell*-jéből dinamikusan felparaméterezhetjük az *RS.EXE*-t. Az *RS.EXE* futása alatt keletkezett hibák automatikusan a konzolon jelennek meg, a „*–t*" paraméter megadása mellett a pontos kódja is megjelenik, amit ha feldolgozunk, detektálható a hiba oka (az *output*-ot *batch* esetén fájlba irányítjuk, vagy egy *.NET*-es kliensen keresztül indított *RS.EXE* futtatása esetén akár a *Process.StandardError, Process.StandardOtput* is felhasználható a konzolra küldött válasz visszaolvasására). A pontos hibakódok itt találhatók: **[3]**.

További megoldás lehet a korábban megismert *Report Manager ASP.NET*-es alkalmazás, amelyben van lehetőség a felhasználói felületen a fájlrendszerből *RDL* kiterjesztésű kimutatásállományok publikálására. Persze ez nem automatizálható, inkább egyszeri, *ad hoc* beavatkozásként jöhet szóba. Egy újabb megoldásként mi magunk írhatunk egy kis keretrendszert, amely képes valahol a fájlrendszerben tárolt *RDL* állományok automatikus publikálására (a *RS* webszolgáltatás webmetódusain keresztül).

Kiterjesztési lehetőségek

A korábbi részek olvasói számára ismerős lehet a következő *RS* platformról szóló ábra.



A Reporting Services platform

Az ábrából is kivehető, hogy a jelzett szolgáltatások egyedi megvalósításokkal is kiterjeszthetőek (erre utal az ábrán szereplő megannyi „*Custom*" szócska), azaz a *RS* így lett tervezve, hogy képes legyen az általunk elkészített *.NET*-es kód felhasználásával bizonyos működést kiterjeszteni, és ezáltal biztosítani azt az üzleti logikát, amire nekünk éppen szükségünk van az adott alkalmazás esetében. A következő komponensek (az ábrából is kiolvasható) esetében van lehetőségünk a beépített funkciók kiterjesztésére:

- Adatfeldolgozás (Data Processing): a beépített *SQL Server*, *OLEDB*, *Oracle* és *ODBC* adatforrás-támogatásokat terjeszthetjük ki újabbakkal, beolvashatunk egy saját egyedi adatforrásból adatokat, egyedileg rendezhetjük stb.
- Kézbesítés (Delivery): a meglévő e-mail- és fájlmegosztáson felül kiterjeszthetjük például faxra, *pager*-re vagy éppen nyomtatóra történő kézbesítési lehetőségekre.
- Megjelenítés (Rendering): az eddigi *HTML*, *XML*, *Excel*, *CSV*, *Image*, *PDF*, megjelenített formátumokon felül készíthetünk újabbakat.
- Biztonság (Security) szolgáltatások kiterjesztése esetén a támogatott *Windows* és *Passport* alapú biztonságot kiterjeszthetjük például egyéni űrlap alapú (*form-based*) autentikációt támogató biztonságra.

A fenti négy komponens közül háromra már készült Microsoft-ajánlás, pontosabban példakód. Ebből két példát együtt kapunk a RS telepítésével (*\Samples\Extensions* könyvtár), az egyik az *FsiDataExtension*, a másik pedig a *PrinterDeliverySample*. A harmadik példa letölthető innen **[4]**, és telepítés után a fenti kettő mellé települ, a neve: *FormsAuthenticationSample*. Ahogy az elnevezésekből következtethetünk rá, egy kiterjesztés maradt ki: a megjelenítés (*rendering*) kiterjesztése. Ez utóbbi a legkomplexebb és a legbonyolultabb, hiszen a jelentésfájl (*RDL*) minden XML-elemének kombinációját támogatnia kell, a *Report Object Model* oly hatalmas, de a bátrabbak nekieshetnek az *IRenderingExtension* interfész saját implementálásának. Készül hozzá a későbbiekben egy dokumentáció is, de még várni kell egy picit rá, amíg teljes lesz. Addig is azt ajánlják a *RS* fejlesztői, hogy mielőtt valaki belevág, fontolja meg a következő alternatívákat:

- Egy meglévő megjelenítési kiterjesztés módosítása.
- A megjelenítés módosítása egy létező kiterjesztéssel az eszközinformáció-beállítások (*device information settings*) megadásával.
- Az egyedi formázást és megjelenítést biztosító *XML* megjelenítési kiterjesztés kombinálása *XSL* transzformációval (*XSLT*).

A meglévő adatforrások kiterjesztése

Nézzük meg az *FsiDataExtension* példaalkalmazáson keresztül, hogy miként tudunk újabb adatforrásokat létrehozni a *RS* számára, azaz miként is terjeszthető ki az adatfeldolgozás. Legyen az adatforrás egy tetszőleges útvonalon található mappák és állományok adatai (a mappák és fájlok nevei, létrehozási dátumuk, méretük, típusuk). Minden *RS* kiterjesztés előre definiált interfészeken keresztül történhet. Az interfészek eljárások, függvények implementációját kényszeríti ki a saját kódunkban. Így az *RS* keretrendszer – az előre definiált interfészeket is implementáló saját osztályunkat felhasználva – biztos lehet abban, hogy rendelkezésre állnak egy-egy feladat végrehajtásához szükséges függvények, eljárások.

A *Microsoft.ReportingServices.DataProcessing* névtérben a következő interfészeket találjuk: *IDataParameter*, *IDataParameterCollection*, *IDataReader*, *IDataReaderExtension*, *IDbCommand*, *IDbCommandAnalysis*, *IDbConnection*, *IDb*

ConnectionExtension, *IDbTransaction*, *IDbTransaction Extension*. Nézzük meg annak a legfőbb részeit, miként implementálhatjuk ezt a saját adatforrást az interfészleírások tükrében. A következő kódban az *IDataReader* interfész az adatok beolvasását végzi el az *FsiDataReader* osztályunkban. A következő kódok csak részletek, nem implementálnak teljeskörűen egy-egy interfészt, csak a lényeges eljárásokat emelik ki.

```
public class FsiDataReader: IDataReader {
 private FsiConnection m_connection = null;
 internal DirectoryInfo m_dir;
 internal FileSystemInfo[] m_fsi;
 internal int m_curRow;
 internal IEnumerator m_ie;
 internal String[] m_names = {"Név", "Méret",
          "Típus", "Létrehozás dátuma"};
 internal Type[] m_types = {typeof(String),
           typeof(long), typeof(String),
           typeof(DateTime)};
 internal object[] m_cols = new object[4];
 internal int m_fieldCount = 4;

 internal FsiDataReader() {}
 internal FsiDataReader(string cmdText) {}
 internal FsiDataReader(string cmdText,
          FsiConnection connection){
  m_connection = connection; }
 public object GetValue(int fieldIndex) {
  return m_cols[i]; }

 public Type GetFieldType(int fieldIndex) {
  return m_types[i]; }

 public string GetName(int fieldIndex) {
  return m_names[i]; }

 public int FieldCount {
  get { return m_fieldCount; } }

 public int GetOrdinal(string fieldName) {
  return Array.IndexOf(m_names, name); }

 public bool Read() {
  if (m_ie != null) {
   if (m_ie.MoveNext() == true) {
    m_curRow++;
    if (m_fsi[m_curRow] is FileInfo) {
     FileInfo f = (FileInfo)m_fsi[m_curRow];
     m_cols[0] = f.Name;
     m_cols[1] = f.Length.ToString();
     m_cols[2] = "Fájl";
     m_cols[3] = f.CreationTime.ToString(); }
    else {
     DirectoryInfo d =
        (DirectoryInfo)m_fsi[m_curRow];
     m_cols[0] = d.Name;
     m_cols[1] = "0";
     m_cols[2] = "Mappa";
     m_cols[3] = d.CreationTime.ToString();}}
   return true; }
  return false; } }

 internal void GetDirectory(string cmdText) {
  m_dir = new DirectoryInfo(cmdText);
  m_fsi = m_dir.GetFileSystemInfos();
  m_curRow = -1;
  m_ie = m_fsi.GetEnumerator(); }
```

Ezek után elkészíthetjük az *IDbConnectionExtension* interfészt implementáló *FsiConnection* osztályunkat is.

```
public class
FsiConnection:IDbConnectionExtension{
 private string m_connString;
 private ConnectionState m_state =
              ConnectionState.Closed;
 private string m_locName = "File Share
              Information";
 public FsiConnection() {}
 public FsiConnection(string connString) {}

 public string ConnectionString {
  get { return m_connString; }
  set { m_connString = value; } }

 public int ConnectionTimeout {
  get { return 0; } }

 public ConnectionState State {
  get { return m_state; } }

 public IDbTransaction BeginTransaction() {
  throw new NotSupportedException(); }

 public void Open(){
  m_state = ConnectionState.Open; }

 public void Close() {
  m_state = ConnectionState.Closed; }

 public IDbCommand CreateCommand() {
  return new FsiCommand(this); }
 public string LocalizedName {
  get { return m_locName; }
  set { m_locName = value; } } }
```

Hasonlóképpen készül az *IDbCommand*-ot megvalósító *FsiCommand* osztályunk is. Itt a *CommandText* tulajdonságban az SQL adatforrás esetén használt *querystring* helyett a fájlmegosztás útvonala fog állni.

```
public class FsiCommand : IDbCommand {
 FsiConnection m_connection;
 FsiTransaction m_txn;
 string m_cmdText;
 int m_cmdTimeout;
 FsiDataParameterCollection m_parameters =
  new FsiDataParameterCollection();

 public FsiCommand() {}
 public FsiCommand(FsiConnection connection) {
  m_connection = connection; }
 public FsiCommand(string cmdText) {
  m_cmdText = cmdText; }
 public FsiCommand(string cmdText,
          FsiConnection connection) {
  m_cmdText = cmdText;
  m_connection = connection; }
 public FsiCommand(string cmdText,
          FsiConnection connection,
          FsiTransaction txn) {
  m_cmdText = cmdText;
  m_connection = connection;
  m_txn = txn; }

 public string CommandText {
  get { return m_cmdText; }
  set { m_cmdText = value; } }

 public int CommandTimeout {
  get { return 30; }
  set {} }

 public CommandType CommandType {
  get { return CommandType.Text;
  set { if (value != CommandType.Text)
      throw new NotSupportedException(); } }

 public FsiDataParameterCollection Parameters {
  get { return m_parameters; } }

 IDataParameterCollection IDbCommand.Parameters
{
  get { return m_parameters; } }

 public IDbTransaction Transaction {
  get { return m_txn; }
  set { m_txn = (FsiTransaction)value; } }

 public IDataParameter CreateParameter() {
  return (IDataParameter)
      (new FsiDataParameter()); }
```

```
public IDataReader ExecuteReader() {
  if (m_connection == null ||
    m_connection.State != ConnectionState.Open)
   throw new
    InvalidOperationException("Nincs nyitott
                    kapcsolat!");
  FsiDataReader reader = new
             FsiDataReader(m_cmdText);
  reader.GetDirectory(m_cmdText);
  return reader; }

 public IDataReader ExecuteReader
          (CommandBehavior behavior) {
 if (m_connection == null ||
    m_connection.State != ConnectionState.Open)
   throw new
    InvalidOperationException("Nincs nyitott
                    kapcsolat!");
  FsiDataReader reader = new
             FsiDataReader(m_cmdText);
  reader.GetDirectory(m_cmdText);
  return reader; } }
```

A fentiekhez hasonlóan rendre el kell készíteni az *IDataParameter*, *IDataParameterCollection* és *IDb Transaction* interfészeket implementáló osztályokat is. Arra ügyeljünk, hogy egy egyedi elnevezésű, közös névtérhez tartozzon minden egyes osztály. Ha mindez megvan, szükségünk van egy projektreferenciára is: *Microsoft. ReportingServices.Interfaces.dll* (a *Reporting Services\ ReportServer\bin* könyvtárában található). Az elkészült *DLL*-t be kell másolnunk a *\Reporting Services\ReportServer\bin* és a *\Microsoft SQLServer\80\Tools\ReportDesigner\bin* könyvtárakba (privát szerelvény). Így mind a *VS.NET Report Designer*-ben, mind pedig a *Report Manager*-ben lehetőségünk lesz a meglévő négy (*MSSQL*, *ORACLE*, *OLEDB*, *ODBC*) kapcsolati típuson felül ezt az új „*File Share Information*" kapcsolatot is kiválasztani. Ahhoz, hogy ez megtörténjen, a következő bejegyzést kell adni a fenti két könyvtárban lévő *RSReportServer.config* és *RSReportDesigner.config* konfigurációs *XML* állományok *Data* eleméhez:

```
<Extension Name="FSI"
 Type="NameSpace.FsiConnection, DLLName"/>
```

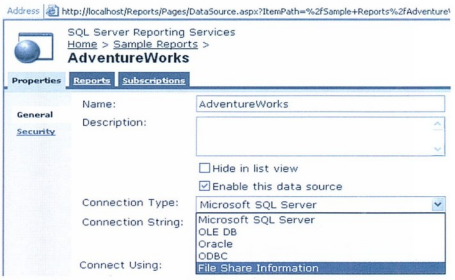
A *NameSpace* helyett a közös névteret kell megadni, azaz az *IDbConnection*, *IExtension* interfészt tartalmazó osztály teljesen minősített nevét, a *DLLName* helyett pedig értelemszerűen a bemásolt *DLL*-ünk nevét, kiterjesztés nélkül. Be kell még állítani a kódhozzáférési (*Code Access Security – CAS*) jogosultságot is a *\Reporting Services\ ReportServer\rssrvpolicy.config* házirend (*policy*) állományban:

```
<CodeGroup class="UnionCodeGroup"
 version="1" PermissionSetName="FullTrust"
 Name="FSICodeGroup" Description="Code group for
 my FSI data processing extension">
 <IMembershipCondition
  class="UrlMembershipCondition" version="1"
  Url="C:\Program Files\Microsoft SQL
    Server\MSSQL\Reporting
    Services\ReportServer\bin\DLLName.dll" />
</CodeGroup>
```

Valamint ugyanezt a *Report Designer*-hez is a *\Microsoft SQL_Server\80\Tools\ReportDesigner\rspreviewpolicy.config* állományban:

```
<CodeGroup class="UnionCodeGroup" version="1"
 PermissionSetName="FullTrust"
 Name="FSICodeGroup"
 Description="Code group for my FSI data
      processing extension">
 <ImembershipCondition
  class="UrlMembershipCondition" version="1"
  Url="C:\Program Files\Microsoft SQL
  Server\80\Tools\ReportDesigner\bin\DLLName.dll"
 />
</CodeGroup>
```

Végül elindítjuk a *VS.NET Report Designer*-t (pl. varázslóval gyorsan összekattinthatunk egy 4 oszlopos mappainformációs listát) vagy a *Report Manager-t*, úgy használhatjuk az adatforrásunk oszlopait, mintha az egy adatbázistábla négy különböző oszlopából olvasódna ki. Hasonlóképpen járhatunk el, ha a feladat például egy XML adatforrásra való kiterjesztés. Bővebben itt: **[5]**.



Kiterjesztett adatforrás a Report Manager-ben

A kézbesítési helyek kiterjesztése

Az eddig leírtakhoz hasonlóan implementálni kell az *IDeliveryExtension*, *IExtension*, *ISubscriptionBase UIUser Control* interfészeket (utóbbival saját kontrollokat is értelmezhetünk az oldal és nyomtatótípus beállításához). Érdemes megnézni a *PrinterDeliverySample* példaalkalmazást, amelynek eredményeképpen nemcsak e-mail- és fájlmegosztás révén tudunk majd kimutatásokat kézbesíteni, hanem nyomtatóra is.

A biztonság kiterjesztése

Ha a beépített *Windows*- vagy *Passport*-hitelesítést űrlap alapúra szeretnénk felváltani, megvan rá a lehetőségünk. Akit érdekel a témakör, letölthetik a példaalkalmazást innen: **[4]**, egy igen részletes és illusztrált leírás is tartozik hozzá, a lényege az eddig említett kiterjesztési technikához hasonlít.

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/yuq3
[2] http://tinyurl.com/2dw93
[3] http://tinyurl.com/27c42
[4] http://tinyurl.com/2oubo
[5] http://tinyurl.com/yqmph

Ami a hivatalos Microsoft tanfolyamokból kimaradt...

Exchange 2000/Exchange Server 2003 – III. rész

Exchange témában egyelőre ez az utolsó cikk ebben a sorozatban. További apróságok mellett egy új kiegészítő komponenst és ennek a „kiegészítésnek a kiegészítését" ismertetjük.

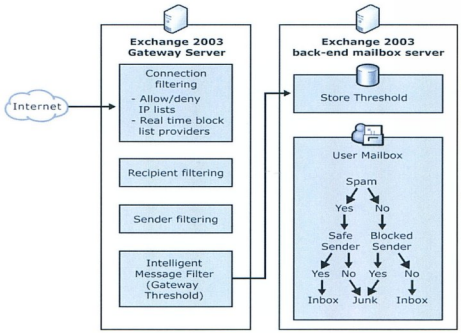
Nézzük, mire is jó az Exchange Server 2003-hoz kínált új kiegészítő komponens!

Intelligent Message Filter

A sokat sejtető nevű komponens egy újabb lehetőség a bejövő levelek szűrésére. Ez a szűrő a spameket, azaz a levélszemetet szűri az alapján, hogy a levél a tartalma és formája mennyire „spamgyanús". Az „intelligencia" szó helyett én inkább a „titkos" szót tettem volna be a komponens nevébe, mert nagyon titkolózik a Microsoft, hogy pontosan miből is állapítja meg ezt a „spamgyanússágot". Valószínű, hogy elsődlegesen a nem korrektül meghatározott feladóból és a címzettek darabszámából. A végeredmény mindenesetre az, hogy minden üzenet kap egy ún. SCL (spam confidence level) értéket, amely alapján tudunk szűrni. Az alábbi táblázatból kiderül, hogy ez az SCL paraméter milyen értékeket vehet fel:

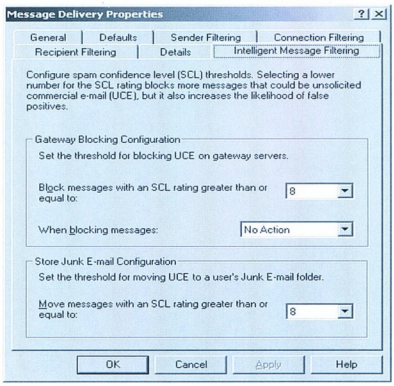
SCL érték	Spamkategória
-1	Exchange szervezeten belüli levelek
0	Nem spam
1 to 9	1: picit spamgyanús levél
	...
	9: nagyon spamgyanús levél

Tehát ezen SCL-paraméter alapján történik a levelek szűrése, méghozzá két szinten. Az első szint maga az internetes bejárat SMTP virtuális szervere, azaz már az Exchange szervezet határán el tudjuk dobni egy bizonyos SCL-szintet elérő kéretlen leveleket. Ennek az a veszélye, hogy esetleg olyan levelek is áldozatul esnek, amelyek igazából számunkra nem spamek, csak úgy néznek ki. Ezért van a másik lehetőség, amelynél a levél eljut a klienshez, és ott kerül be a „Junk E-mail" mappába, ha az SCL-érték az általunk beállított küszöbérték felett van. Itt már a felhasználó definiálhat egy olyan szabályt, hogy egy bizonyos feladó mégiscsak „Safe sender", így az ilyen levél a normál Inboxba kerül. A teljes folyamat, illetve az egyes szűrők egymáshoz viszonyított helyzetét az 1. ábra mutatja. Természetesen, ha nincs külön Gateway funkciót ellátó Exchange kiszolgálónk, akkor a két szint feladatát egy Exchange is el tudja látni.



1. ábra A szűrési folyamat és a szűrők helyzete

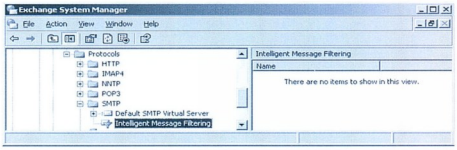
Ha telepítettük az IMF komponenst, akkor a Global Settings/Message Delivery tulajdonságlapján tudjuk ezt a kétszintű szűrést hangolni:



A Gateway résznél a következő eseményeket kérhetjük:

No Action	Csak annyit csinál, hogy beállítja az üzenetek SCL-attribútumát.
Archive	Berakja a beállított SCL-szint feletti üzenetet az Exchsrvr\Mailroot\vsi n\UCEArchive könyvtárba. Ha mégis kézbesíteni kellene ezeket a levelet, akkor egyszerűen pakoljuk át a pickup könyvtárba.
Delete	Egyszerűen elfogadás után törli az üzenetet.
Reject	Közli a küldő SMTP kiszolgálóval, hogy nem fogadhatja ezt a levelet.

Természetesen az adott SMTP virtuális szerveren is engedélyezni kell a szűrő alkalmazását, ami a többi szűrőhöz képest kicsit más helyre költözött, nem a VS tulajdonságlapjáról lehet elérni, hanem az SMTP konténerben található új objektumból.

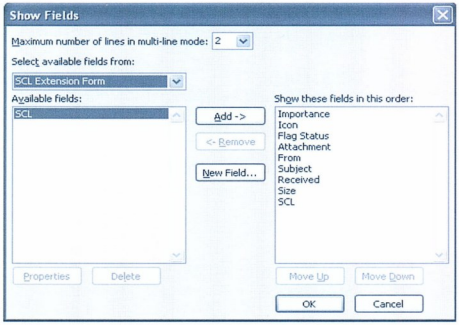


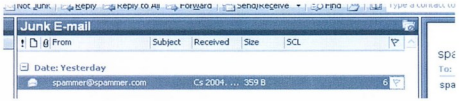
A Store-ra, azaz a felhasználók Junk e-mail-mappájába kerülő üzenetekre vonatkozó SCL-szint természetesen kisebb vagy egyenlő, mint a Gateway-nél beállított, hiszen amit be sem engedünk, azzal nincs sok értelme Store szinten bármit is kezdeni.

Hogyan láthatjuk az SCL-szintet?

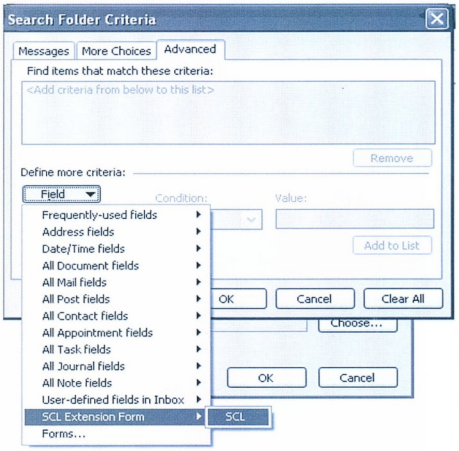
Az IMF-en az SCL szintek beállításához – miután nem ismerjük az algoritmust, amellyel ezek meghatározódnak – jó lenne legalább látni, hogy az egyes spam levelek milyen SCL-értéket kapnak. A szomorú helyzet az, hogy alaphelyzetben ezt nem láthatjuk. Ezért is említettem korábban, hogy az „intelligens" helyett inkább a „titkos" jelzőt kellett volna a termék nevében szerepeltetni. De szerencsére nem ilyen rossz a helyzet, mert némi buherával ez is megjeleníthető. Először is el kell készíteni egy SCL.CFG fájlt, ami egy speciális űrlap-definíció, ami már tartalmazza ezt az új SCL-attribútumot, és hozzá kell adni az Outlookhoz mint egy egyedi űrlap a következők szerint **[1]**. Ezután már megjeleníthetővé válik az SCL-érték a különböző mappanézetekben.

A következő ábrán az utolsó oszlopban meg is jelenítettem az SCL-értéket, ami például egy telnet ablakban előállított, az adatmezőjében hiányos fejléccel rendelkező levél esetén 6-osnak adódik.



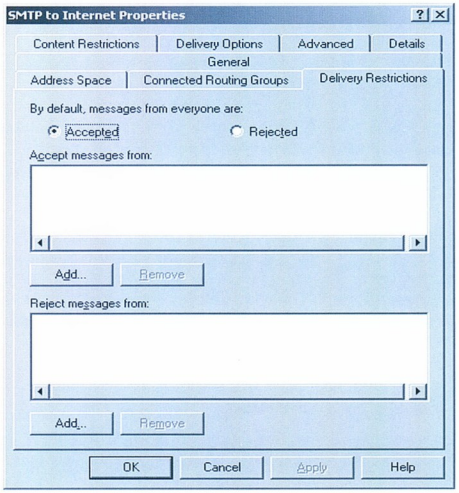


Természetesen az SCL-értékeket felhasználhatjuk Search Folderek szűrőfeltételeinél és Rules Wizard szabályoknál is. Ilyenkor a kritériumnál az Advanced fülön a keresett mezőt az „SCL Extension Form" pontnál találjuk meg:



További apróságok: felhasználó tiltása konnektoron

Bizonyos esetekben szükséges lehet konnektorok használatának tiltása személyek vagy csoportok részére, például az internetes levelezést tilthatjuk le ilyen módon. Úgy tűnik, hogy erre van is felület az Exchange kiszolgálónkon:



Sajnos az itt beállítható tiltásokat és engedélyeket az Exchange alaphelyzetben figyelmen kívül hagyja. Ahhoz, hogy ezek működésbe lépjenek, definiálni kell egy registry bejegyzést:

```
HKLM\System\CurrentControlSet\Services\Resvc\Par
meters
CheckConnectorRestrictions (DWORD) = 1
```

A beállítás érvényre jutásához még az SMTP és a Microsoft Exchange Routing Engine szolgáltatást is újra kell indítani. Ha ezek után olyasvalaki szeretne levelet küldeni egy ilyen konnektoron keresztül, akinek ezt megtiltottuk, sajnos nem túl bőbeszédű NDR hibaüzenetet kap:

```
Reporting-MTA: dns;smtp-out.iqjb.hu
Final-Recipient: rfc822;VegetariJanos@odakint.hu
Action: failed
Status: 5.7.1
X-Display-Name: ' VegetariJanos@odakint.hu'
```

Érdemes erről a hibajelzésről tájékoztatni az érintett felhasználókat.

M: meghajtó visszavarázsolása

Az Exchange 2003-ban megszűnt az Exchange 2000-nél megszokott M: meghajtó, ami az Exchange adatbázisoknak adott egy fájlrendszerszerű nézetet. Ennek oka az volt, hogy nagyon sokszor előfordult, hogy figyelmetlen rendszergazdák ráeresztették a víruskeresőt erre a meghajtóra, és az ott mindenféle galibát okozott.
Bizonyos esetekben azonban jó szolgálatot tehet nekünk ez a meghajtó, ezért szükséges lehet a megjelenítése, amit szintén egy registry-beállítással tehetünk meg:

```
HKLM\SYSTEM\CurrentControlSet\Services\EXIFS\Par
meters
DriveLetter (REG_SZ) = M
```

Természetesen nem muszáj M-nek hívni ezt a meghajtót, lehet bármilyen, még nem foglalt betű is.
A beállítás érvényre juttatásához – miután ez egy rendszerszintű szolgáltatás – a gépet újra kell indítani.

Sürgős helyreállítás nem a Recovery Storage Groupba

Új lehetőség az Exchange 2003-nál, hogy fel lehet venni egy újfajta adatbáziscsoportot, az ún. Recovery Storage Groupot. Ehhez hozzá lehet rendelni üzenet adatbázisokat, és ekkor a helyreállítás mentésből nem az „igazi" üzenet adatbázist írja felül, hanem erre az alternatív helyre történik a visszatöltés. Erről az alternatív helyről aztán az Exmerge segédprogram segítségével lehet például a véletlenül törölt üzeneteket visszarakni a figyelmetlen tulajdonos levelesládájába. De mi van akkor, ha épp használjuk a Recovery Storage Groupot, de pont egy olyan adatbázist kellene helyreállítani az „igazi" helyére, amit hozzárendeltünk az RSG-hez? Egyik lehetőség, hogy töröljük ezt onnan, de ha ezt nem akarjuk, mert még dolgunk van vele, akkor ki lehet kapcsolni az RSG-ba történő helyreállítást a következő registry-beállítással:

```
HKLM\System\CurrentControlSet\Services\MSExchange
IS\ParametersSystem\
Recovery SG Override (DWORD) = 1
```

Jelszó módosítása OWA-n keresztül

A jelszó megváltoztatása az Outlook Web Access felületen keresztül nem túl egyszerű. Először is azért, mert alaphelyzetben az Exchange 2003-nál ez a lehetőség ki van kapcsolva, amit az alábbi registry-érték módosításával kell bekapcsolni:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\MSExchangeWEB
DisablePassword (DWORD) = 0
```

Ha ezt megtettük, akkor már megjelenik a jelszó megváltoztatását lehetővé tevő gomb a Beállítások oldalon:

Azonban ekkor sem örülhetünk még, mert ez még nem elég, a gombra kattintva egy hibás weboldalat kapunk. Merthogy még tanúsítványt kell szerezni az OWA-t futtató virtuális webkiszolgálóra, hogy lehetővé váljon az Exchange virtuális könyvtárakon az SSL titkosított csatornán történő kommunikáció. Ezután a következőket kell még elvégezni:

1. Ebben a virtuális webkiszolgálóban újabb virtuális könyvtárat kell felvenni IISADMPWD néven.
2. Ezt a *\Windows\System32\Inetsrv\iisadmpwd* könyvtárhoz kell kötni.
3. Ellenőrizzük, hogy a virtuális könyvtár hozzáférésénél be vannak-e állítva a Read és a Run Scripts (Such As ASP) jogok.
4. Az újonnan felvett virtuális könyvtár tulajdonságlapján az Authentication and Access Control szekcióban ellenőrizzük, hogy az Enable Anonymous Access engedélyezve van-e.

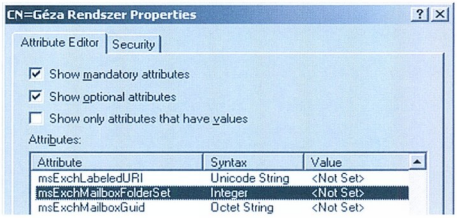
Innentől kezdve a „Jelszó módosítása" gombra kattintva megjelenik ez az oldal:

OWA funkcionalitás beállítása

A jelszó megváltoztatásának lehetőségén kívül az OWA felület számos egyéb funkcióját tudjuk hangolni. Ezt két szinten is megtehetjük: az egyik a kiszolgáló szintje, ilyenkor a következő registry-kulcsot kell igényeink szerint beállítani:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\MSExchangeWeb\OWA
DefaultMailboxFolderSet (DWORD)
```

A másik szint a felhasználónkénti beállítás, amit viszont az ADSIEdit eszközzel végezhetünk a felhasználói objektum attribútumai között:



Mindkét esetben a következő bináris helyi értékekkel tudunk különböző OWA-funkciókat ki-be kapcsolni:
Minden funkció: -1 (FFFFFFFF) Az egyes bitek (jobbról balra): levelezés, naptár, névjegyalbum, feladatok, napló, feljegyzések, nyilvános mappák, emlékeztetők, új levélről értesítés, gazdag kliens, helyesírás-ellenőrző, S/MIME, keresőmappák, aláírás, szabályok, témák, levélszemét. A keresőmappák csak akkor látszanak az OWA felületen, ha korábban definiáltunk ilyet az Outlookból, egyébként az alaphelyzet szerinti keresőmappák nem látszanak, és nem is lehet az OWA felületből ilyeneket csinálni.

Connection Filterhez saját Relay Block List készítése

Új szűrési lehetőség az Exchange 2003-ban a Connection Filter, ami a kiszolgálónkra kapcsolódó SMTP hostok IP-címét nézi meg különböző „feketelistákon", és e szerint bezárja a kapcsolatot, vagy engedi tovább a kommunikációt. (Igazából nem a kapcsolódás időpontjában bont, hanem a „rcpt to:" SMTP-parancs kiadása után.) Számos ilyen publikus RBL lista létezik, amelyeket a Connection Filter tulajdonságlapján fel tudunk venni, azonban teszteléshez nem árthat egy saját RBL listát készíteni. Ennek módja egyszerű. A DNS kiszolgálónkba vegyünk fel egy új forward lookup zónát, mondjuk, *rbl.sajat* néven. E zónában semmilyen dinamikus frissítésre nincs szükségünk. Vegyünk fel egy üres Host rekordot, ami a DNS-szerverünk IP-címét tartalmazza. A tesztelendő IP-címet pedig vegyük fel ebbe a zónába Host rekordként, úgy, hogy a Host neve legyen az IP-cím, de fordított oktet sorrendben. Azaz például egy 192.168.103.181 című gépet úgy tudunk feketelistára tenni, hogy az *rbl.sajat* zónába felveszünk egy 181.103.168.192 nevű hostot, szokásjog alapján 127.0.0.x címmel. Az x különböző dolgokat jelent, nincs teljes egyetértés a különböző feketelistákon. Például:

X	Jelentés
3	Spam-forrás
4	Open relay

Ezek után a Connection Filternél felvehetjük Block List Service-ként a mi *rbl.sajat* feketelistánkat.

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

A cikkben szereplő URL-ek:

[1] http://tinyurl.com/yqwnl

Microsoft CRM v 1.2 - 1. rész

A Microsoft első ASP.NET alapú üzleti alkalmazása

Szeptemberben jelenik meg Magyarországon a Microsoft ügyfélkapcsolat-kezelési rendszere, a Microsoft Customer Relationship Management (CRM) 1.2 lokalizált változata. Cikksorozatunk első része rövid áttekintést nyújt a rendszer szolgáltatásairól és az alkalmazott technológiákról. Leírásunk természetesen nem teljeskörű, néhány példán keresztül mutatjuk beaz általunk leginkább érdekesnek ítélt funkciókat.

A Microsoft CRM a kis- és középvállalatok számára kifejlesztett ügyfélkapcsolat-kezelési megoldás. Elsősorban az operatív, azaz kereskedelmi folyamatok automatizálására és az ügyfélszolgálati tevékenységek támogatására használhatjuk, de működése és felépítése miatt – mint azt a későbbiekben ismertetjük – lehetőségeink ebben még korántsem merülnek ki.

A Microsoft CRM teljes mértékben .NET technológiára épülő rendszer, amely számos további Microsoft technológiát és szerveroldali alkalmazást (Active Directory, IIS, SQL, Exchange) használ működése során. A CRM könnyen illeszthető más üzleti rendszerekhez (InfoPath, MapPoint) illetve bármilyen ERP rendszerhez a BizTalk szerveren keresztül. A program felhasználói felülete a Microsoft Project 2002/2003 rendszerekben már sikerrel alkalmazott webes megjelenést követi.

Jelenleg két verzió közül választhatunk: *alap (Standard) és professzionális (Professional),* ezen belül lehetőségünk van *kereskedelmi (Sales),* illetve *ügyfélszolgálati (Customer Service)* licenc vásárlására, melyek egyben a CRM *két fő modulját* is meghatározzák. A licencek közötti funkcionalitásbeli különbségekre utalni fogunk cikkünkben a megfelelő helyeken.

Kereskedelem (Sales) modul

Elsőként ismerkedjünk meg a kereskedelmi modul kínálta funkciókkal! Mint általában az ügyfélkapcsolatokkal foglalkozó szoftverekben, a kereskedelmi folyamatokat a *Megkeresés (Lead) – Lehetőség (Opportunity) – Ajánlat (Quote) – Megrendelés (Order) – Számla (Invoice)* „állomások", illetve a hozzájuk kapcsolt *tevékenységek (Activity)* határozzák meg. Az adott kereskedelmi folyamatokhoz pedig *partnereket (Account)* és *kapcsolattartókat (Contact)* társíthatunk.

Anélkül, hogy a kereskedelmi modul minden lehetőségét feltérképeznénk, nézzünk inkább egy példát a rendszer használatára! Tételezzük fel, hogy egy kereskedelmi akcióban néhány száz emailt küldünk termékeinkkel kapcsolatban olyan cégeknek, amelyek eddig még nem vásároltak tőlünk (most az egyszerűség kedvéért ezt ne spamnek, hanem direkt marketingnek nevezzük). A CRM nyelvére lefordítva ez egy *megkeresés* és egy ehhez kapcsolt *tevékenység* lesz. Esetünkben ez egy CRM e-mail, ami természetesen sablon alapján készülhet. Az emailhez elég megadnunk az üzenet szövegét, a többi adatot (címzett, megszólítás, aláírás, stb.) a CRM majd hozzáteszi. Amennyiben levelünkre választ kapunk, az automatikusan visszatalál a CRM rendszeren belül az általunk rögzített megkereséshez.

Tegyük fel, hogy akciónk sikeres volt, és a cég érdeklődik termékeink iránt. A megkeresésből egy mozdulattal *lehetőséget* készíthetünk, és felvehetjük az adott céget *partnereink (Account)* közé (természetesen a megkeresést nem csak mi indíthatjuk, de érzékeltetni akartuk a megkeresés és a lehetőség közötti különbséget). Ha van *kapcsolattartó (Contact),* azt is felvesszük; ez egyébként automatikusan megtörténik a megkereséskor bevitt adatok alapján. Ezután a lehetőségből *ajánlatot* készíthetünk, majd az *ajánlatból* megrendelés, végül pedig *számla* készül.

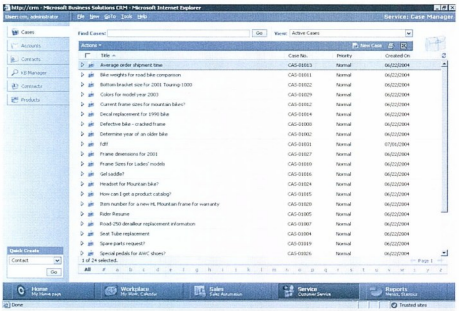
Tevékenységeink mindig a megfelelő helyre kerülnek rögzítésre:, így például a megrendeléssel kapcsolatos levelek, telefonok stb., a megrendeléshez, a számlával kapcsolatosak pedig a számlához. A megrendelést és számlát egy automatizált *munkafolyamat (workflow)* segítségével küldhetjük át ERP rendszerünkbe. Az egész folyamatot nagyrészt automatizálhatjuk is, úgy, hogy a felhasználó számára előírjuk, hogy mikor milyen lépéseket kell elvégeznie.

Az előbbi esetben például a beérkezett válaszokhoz kötelező tevékenységet rendelhetünk, amelyeket mindenkép-

pen el kell végezni, mielőtt lezárhatnánk a folyamatot. Megadhatjuk például, hogy a termék kiszállítása előtt telefonon egyeztetni kell a vevővel. A rendszer addig nem fogja lezárni a megrendelést, míg ez meg nem történik.
Hasznos lehet, hogy termékeinkhez, partnereinkhez, illetve a folyamat bármely pontjához kapcsolhatjuk versenytársaink adatait is (csak a professzionális verzióban). Természetesen rengeteg további automatizálásra van lehetőségünk a workflow-k segítségével. Az eladási folyamat legfontosabb automatizmusait az alapváltozat már eleve tartalmazza.
A felhasználói felület webes, de a modul Microsoft Outlookba integrált felhasználói felülettel is rendelkezik *(Sales for Outlook Client)*, amely amellett, hogy offline működést is lehetővé tesz, már általában ismerős a felhasználók számára, így a beépülő CRM modul használata könnyen megtanulható és használható.

Ügyfélszolgálat (Customer Service) modul

Az ügyfélszolgálat modul a *szerződések (Contracts)* és *esetek (Cases)*, illetve az azokhoz kapcsolódó tevékenységek kezelésén alapul.
Eset lehet bármilyen, partnereink által kezdeményezett, megoldandó feladat: a vállalatunkkal kapcsolatos egyszerű kérdés (például milyen termékeink vannak) megválaszolásától a termékreklamációkon át a szervizügyintézésig.
A szerződés: adott partnerünkkel kapcsolatban milyen eseteket és hogyan oldunk meg. Megadhatjuk mikor állunk az ügyfél rendelkezésére, illetve milyen módon (telefonon, helyszínen) és milyen díjszabással végezzük tevékenységeinket. Ha már van szerződésünk a partnerrel, az elvégzett tevékenységeket automatikusan ki is számlázhatjuk, és a szerződésben rögzíthetjük, milyen rendszerességgel tesszük ezt.



Felhasználói felület – ügyfélszolgálat modul

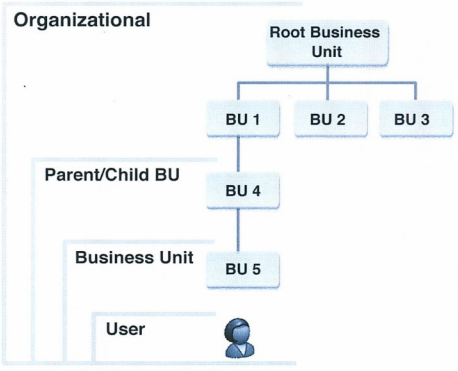
A fenti ábrán az ügyfélszolgálati modulban az aktív, még meg nem oldott esetek listáját látjuk. A cikksorozat további részeiben részletesebben is fogunk ezzel foglalkozni, most „kedvcsinálóként" néhány hasznos funkció: a különféle eseteket például automatikusan a megfelelő felhasználóhoz tudjuk irányítani, vagyis mindegy ki rögzítette az esetet, az fog foglalkozni vele, aki ért hozzá. Így egy csapásra megoldhatjuk azt is, hogy a partnerek ügyes-bajos és sürgős dolgai ne a két hónapig szabadságon lévő munkatárs asztalán landoljanak. További lehetőségünk arra, hogy ne tűnjenek el nyomtalanul partnereink kérései: ha meghatározott időn belül nem kerül egy eset megoldásra, a rendszer automatikusan üzenetet küldhet egy ezért felelős felhasználónak (menedzser).

Technológia

A Microsoft CRM ízig-vérig .NET technológiákra épül. A rendszer felhasználói felületei XML webszolgáltatásokon keresztül kommunikálnak a CRM adatbázisaival, az adatok szinte kizárólag XML-ben utaznak az alkalmazás különböző komponensei között. A CRM fejlesztői több mint nyolcszáz .NET osztályban valósították meg a funkcionalitást, amelyek nagy része a külső fejlesztők számára is elérhető a CRM SDK-n keresztül. A CRM rendszer nemcsak „lóg a levegőben", hanem igen sok létező Microsoft technológiát integrál. A továbbiakban ezek közül a legfontosabbakkal való együttműködést tekintjük át. Némelyikkel kicsit részletesebben is foglalkozunk majd a következő részben, ahol a CRM szerver és a Sales for Outlook ügyfélalkalmazás telepítését ismertetjük.

Active Directory (AD)

A Microsoft CRM a felhasználók jogosultságainak kiosztására és ellenőrzésére kizárólag az AD szolgáltatásait használja, ezért a CRM felhasználók csak létező AD felhasználók közül kerülhetnek ki. A CRM rendszer és az AD ilyen jellegű szolgáltatásai között a kapcsolatot a **CRM Security Service** tartja.
A rendszer működéséhez natív vagy W2K3 módban működő tartomány szükséges, amennyiben több erdőben egy CRM rendszert használunk, az erdők között kétirányú trust kapcsolatokat kell létrehoznunk. Telepítéskor meg kell adjuk, hogy melyik *szervezeti egységbe (OU)* kerüljenek a CRM *üzleti szervezeti egység (Business Unit)* struktúrája alapján létrehozott további OU-k, melyekben a *szerepkörök (Security Role)* szerint a *rendszer felhasználói csoportokat (NT Users Group)* hoz létre. Ezekbe a csoportokba kerülnek majd a felhasználók.

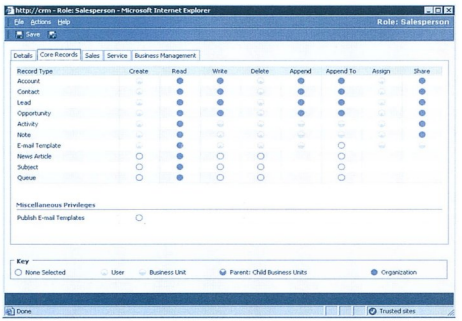


CRM hozzáférési szintek

Természetesen a jogosultságok kiosztását a CRM felhasználói felületén végezhetjük el, ehhez nincs szükség az AD felügyeleti eszközeire, és különösebb AD ismeretekre sem. Vállalatunk szervezeti felépítésének alapegységei a CRM rendszerben az üzleti szervezeti egységek (Business Unitok). Ezekben helyezhetjük el a felhasználókat, illetve a szerepköröket, amelyek a felhasználók hozzáférését határozzák meg az egyes CRM üzleti objektumokhoz. Ezeket az objektumokat rekordoknak nevezzük. CRM rekord például a partner, a megrendelés, a számla, a termék, a tevékenység és a megjegyzés. A rekordok mezőkből állnak, mint például a név, termékkód, ár.
A hozzáféréseket az alábbi módon adhatjuk meg:

- meghatározzuk a szerepkörhöz kapcsolódó hozzáférési szintet: user, business unit, parent-child business unit és organizational.
- a jogosultságot rekordonként és műveletenként adhatjuk meg, azaz beállíthatjuk, hogy az adott rekordon milyen műveleteket lehet végezni.
- a szerepkört felhasználóhoz rendeljük

Példa: a felhasználóhoz rendelt szerepkör hozzáférési szintje *business unit,* a jogosultság a *lehetőség* rekordra olvasás szintű. Amennyiben a felhasználóhoz ezt a szerepkört rendeljük, a business uniton belül minden *lehetőség* típusú rekordra olvasás joga lesz.

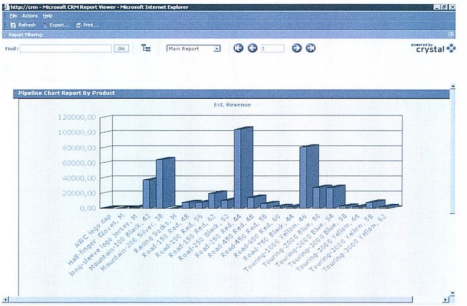


Szerepkörök beállítása a CRM-ben

Nem véletlen, hogy a szerepköröket ilyen részletességgel határozhatjuk meg. A vállalatok életében döntő fontosságú, hogy a felhasználók hozzáférjenek a számukra fontos adatokhoz, azonban – különösen az utóbbi időkben – legalább ennyire hangsúlyos az üzleti adatok belső támadásokkal szembeni védelme. A CRM jogosultságkezelése erre igen kifinomult megoldást kínál.

Crystal Reports

A rendszerben rögzített adatokat természetesen használni is fogjuk valamire: segítségükkel elemezzük üzleti folyamatainkat. Ehhez szükségünk van egy jelentéskészítő eszközre. A Microsoft itt lemondott a saját fejlesztésről, és partnerként egy, ezen a területen igazi nagyágyút vont be: a Crystal Decisions-t. Maga a jelentéskészítő pedig a **Crystal Enterprise for Microsoft CRM,** azaz egy kifejezetten a Microsoft CRM rendszerhez készült eszköz, amely a CRM rendszerrel együtt automatikusan telepítésre kerül. Ennél a verziónál az adatforrás kizárólag az MSCRM lehet, illetve csak a CRM-en keresztül tekinthetők meg a jelentések, módosításukra nincs lehetőségünk.
Amennyiben a rendelkezésünkre álló, egyébként bőséges számú, összesen 119 jelentés nem elég, és újakat akarunk létrehozni vagy a meglévőket módosítani, szükségünk lesz a **Crystal Reports 9 Professional, Developer,** vagy **Advanced** verziójára. Valószínűleg rossz vásárt nem csinálunk majd vele, mert a legtöbb ERP rendszerhez használhatjuk jelentéskészítő eszközként.



Crystal Reports jelentés

SQL

A CRM szerver a Microsoft SQL Server legalább Standard változatát igényli. Vigyázzunk, emiatt egy kisebb cégnél is a Small Business Server Premium verziójára van minimálisan szükségünk, az MSDE (Desktop Engine) – amit a Standard verzióval kapunk – nem lesz elég (a teljes szöveges indexelés hiánya miatt már a telepítés sem lesz sikeres). A Sales for Outlook ügyfélalkalmazás egy külön MSDE példányt igényel, ennek telepítését a telepítő elvégzi helyettünk.
A CRM rendszer négy adatbázist hoz létre telepítésekor a következő neveken és tartalmakkal: CRM adatok: **<OU> + MSCRM,** CRM rendszerleíró adatbázis: **<OU> + Metabase,** Crystal Reports adatok: **<OU> + <Server name> + CRM_Crystal,** disztribúciós adatok a Sales for Outlook Client alkalmazáshoz: **<OU> + MSCRMDistribution.**
Az adatbázis sémáját közvetlenül soha ne módosítsuk. Kizárólag a rendszerleíró adatbázison keresztül tehetjük ezt úgy, hogy az adatbázis folyamatos működése és sértetlensége biztosítva legyen.
Ha a CRM rekordokhoz új mezőket szeretnénk adni (például olyan egyéb adatokat is szeretnénk rögzíteni, amelyekre a meglévő mezők nem elegendőek), rendelkezésünkre áll egy felügyeleti eszköz, a *Deployment Manager,* melyben többek között erre is. lehetőségünk van. A tervezésnél vegyük azonban figyelembe, hogy néhány rekord nem módosítható. A séma módosítását a cikksorozat következő részeiben még tárgyalni fogjuk.

Internet Information Services

Mielőtt a CRM szervert telepítenénk, létre kell hoznunk egy virtuális könyvtárat, ahová a telepítés során a CRM ASPX lapjai kerülnek. Nem kell ezt megtennünk a Sales for

Outlook Client telepítésekor (a telepítő ezt elvégzi helyettünk), bár ebben az esetben is szükséges lesz az IIS. A kliensgépre ugyanazok az ASPX lapok kerülnek, illetve azon ugyanazok a szolgáltatások működnek, mint a CRM szerveren.
A szerver működéséhez minimálisan 5.5-ös verzió szükséges (a Windows 2000 szerverben még ez található), de elsősorban biztonsági okokból a 6.0-ás verzió javasolt; a Sales for Outlook Client telepítésekor is ez kerül a gépre. A kliensgépek esetében mindenképpen figyelembe kell vennünk, hogy telepítéskor egy teljes funkcionalitású webkiszolgáló kerül a felhasználók gépére, így az ezzel kapcsolatosan felmerülő biztonsági kérdésekkel is foglalkoznunk kell.

Exchange E-mail Router

Ezt az összetevőt Exchange szerverünkre kell telepítenünk, melynek verziója 2000 vagy 2003 lehet. Az Exchange E-mail Router a CRM és az Exchange közötti kapcsolattartást végzi: kezeli a CRM rendszerből indított leveleinket, illetve a postaládánkba beérkező leveleket továbbítja a CRM felé. Lehetőségünk van csak az egyedi azonosítóval ellátott leveleket beengedni, ebben az esetben a „levélválogatást" elvégzi a router. A CRM rendszerből indított, és valamilyen tevékenységünkkel, illetve bármely rekorddal (például egy *lehetőséggel*) összefüggő e-mail egyedi azonosítót, GUID-ot (Global Unique Identifier) kap. Az azonosító az e-mail tárgy mezőjében kerül elhelyezésre, ez biztosítja hogy a válasz „odataláljon" a megfelelő helyre, azaz ahhoz a rekordhoz, amelyhez elküldött e-mailünk is tartozik. Ezt a feladatot is az Exchange E-mail Router szolgáltatás végzi.
Természetesen „beengedhetjük" a CRM rendszerbe összes levelünket is, és saját magunk kapcsolhatjuk őket a rekordokhoz, ez azonban nem javasolt: megfelelő jogosultságok birtokában akár magánlevelezésünkhöz is hozzáférhetnek más CRM felhasználók.

Kovács László
(MCSE+S, MCT, CRM vezető oktató)
kovacsl@sbuilders.hu
Kovács Zoltán
(MCSE, MCDBA, MCSD, MCT vezető fejlesztő)
kovacsz@sbuilders.hu
A szerzők a Számalk Oktatási Rt. Továbbképzés bivatalos Microsoft oktatói.

Fókuszban a projektmenedzsment

Júniusban Budapesten volt a Nemzetközi Projektmenedzsment Szövetség 18. világkongresszusa. A rendezvény támogatásában tevékenyen részt vállalt a Microsoft is, amely elkötelezett híve a projektmenedzsment fejlődésének.

Napjainkban a vállalatoknak, szervezeteknek folyamatosan újabb, összetettebb és szorosabb együttműködést igénylő feladatokat kell megoldaniuk, éppen ezért fontos a különböző üzleti folyamatok, rendszerek, projektek áttekinthetővé tétele. A gyors és hatékony megoldásokat kínáló projektmenedzsmentet támogató eszközrendszerek egyre nagyobb szerepet kapnak a vállalatoknál.
„Az információk összegyűjtése és elosztása jelentős időráfordítást kíván a projektvezetőktől. A Microsoft Windows SharePoint Services-zel integrált Nagyvállalati Projektmenedzsment (Enterprise Project Management, EPM) megoldása számos funkcióval támogatja a probléma megoldását. Segít például a projekttervek intranetes közzétételében, a projektmunkák engedélyeztetésében, projektmegbeszélések szervezésében és megtartásában, jelentések generálásában, online kérdőívek készítésében és még rengeteg más hasznos funkcióval bővíti a napi munkát segítő lehetőségeket" – mondta el az IPMA konferencián tartott előadásában dr. Shadt György, a Microsoft vezető tanácsadója.
A Microsoft kiváló eszközöket kínál a projektek menedzselésére. Az Office Enterprise Project Management (EPM) lehetővé teszi, hogy a szervezetek még hatékonyabban felügyelhessék és irányíthassák emberi erőforrásaikat, projektjeiket és folyamataikat. A Microsoft Office EPM megoldásával a teljes szervezet együtt dolgozhat a folyamatok állandó javítása érdekében, a hatékonyság növelhető, a tudás és az információk megosztása megkönnyíthető, és mindez a beruházás gyors megtérülését eredményezi. A jól ismert eszközök és a Microsoft Office Rendszer számos programjával való könnyű integrálódás révén az EPM megoldás elősegíti a vállalaton belüli széleskörű használatot, amivel pontosabban folyik a munka, jól láthatók a felelősök, és egyszerűbb a jóváhagyási folyamat is.

További információ:
www.microsoft.com/hun/project

A termék ingyenes próbaverziója itt rendelhető meg:
http://www.microsoft.com/hun/office/project/cdorder.asp

Dr. Watson

Bolondbiztos EFS

Ebben a cikkben nem az EFS (Encrypting File System) működéséről, vagy használatáról olvashatnak, hanem arról, hogyan tehető az EFS-rendszer bolondbiztossá, hogyan növelhető a megbízhatósága a titkosított fájlok kibontásához szükséges magánkulcsok központi mentésével. Kiválasztott vállalatunknál Windows Server 2003-as tartománykörnyezetet feltételezünk, amelyben egy Enterprise (azaz Active Directoryval integrált) CA Server is van.

Ha megkérdezzük a vállalatok rendszergazdáit, szerintük használja-e valaki náluk az EFS-t (titkosító fájlrendszer), a válaszolók 95 százaléka azt fogja mondani, hogy nem. Természetesen ez nem igaz. A Vezérigazgató Elvtárs és sok más felhasználó, aki kényes adatokkal dolgozik már régen megtanulta a szomszéd Pistikétől a titkosítás használatának fortélyait, és a legfontosabb fájljait titkosítva tárolja a merevlemezen, anélkül, hogy erről bárkinek is szólt volna. Ami titok, az titok. Bezzeg felszínre kerül a titok, amint elvész a fájlok megnyitásához szükséges magánkulcs! Ez pedig csupán idő kérdése, lévén hogy a felhasználó nemhogy azt nem tudja, hol van a létfontosságú magánkulcs, hanem azt sem, hogy ez a fogalom mit takar. Eljön a pillanat, amikor a Vezérigazgató Elvtárs betoppan az informatikushoz a laptopjával, és könyörögve kéri, hogy állítsa vissza az eredeti állapotot. Ilyenkor fejveszett kapkodás kezdődik, mert az informatikusnak sincs a leghalványabb elképzelése sem erről a titokzatos rendszerről (tisztelet a kivételnek!). Némi olvasgatás után rájön, hogy a fájlok megnyitásához szükséges kulcsok a Vezér Elvtárs profiljában voltak, és máris összeáll a katasztrófa teljes képe: a kulcsok elvesztek, mivel:

- valakik újratelepítették a Windowst vagy
- letörölték a felhasználói profilt vagy
- a Vezér maga törölte le a kulcspárt az Internet Explorer Tools ' Internet Options menüjében matatva, vagy
- megsérült a profilkönyvtár tartalma.

Ez csak néhány gyakori eset, lehetséges azonban, hogy még ennél is kacifántosabb eljárás során veszett el a kulcs. Ez utóbbi akkor fordul elő, amikor a Vezér azt hiszi magáról, ért a számítástechnikához. Ilyenkor például kiexportálja az EFS tanúsítványt .CER fájlba (ebben nincs ám bent a privát kulcs!), és ezután, mint aki jól végezte dolgát, újrainstallálja a Windowst, majd az új oprendszerbe beimportálja az immár elárvult nyilvános kulcsot. Ahelyett hogy PFX-be exportált volna a bolond. Ésatöbbi, ésatöbbi, nem ragozom tovább. Egy szó mint száz, a magánkulcs végérvényesen elveszett.

A fájlok megfejthetők lennének az úgynevezett Recovery Agent magánkulcsával – de az sincs meg. Illetve természetesen megvan, csak senki sem tudja, hogy hol. Vagy mégis csak tudja valaki, hogy a legelőször telepített tartományvezérlőn az Administrator profiljában van, de azt már nem sejti, hogyan lehetne összepároztatni az RA magánkulcsát a Vezér fájljaival. (Elárulom az egyik módszert: ki kell exportálni PFX-be, majd bejelentkezni Administratorként a laptopon, beimportálni a magánkulcsot is tároló PFX fájlt, és megnyitni a titkosított dokumentumokat.)

Elveszett kulcs = visszavont kulcs?

Foglalkozzunk még egy picit az elveszett EFS-kulccsal. Minden valamirevaló PKI-rendszerben alapkövetelmény, hogy ha egy magánkulcs kompromittálódik, a hozzá tartozó tanúsítványt vissza kell vonni. Ilyen eset lehet, ha a kulcs nyilvánosságra kerül, a kulcsot tároló Smart Card elvész stb. Ha ezeket az eseteket összevetjük az imént vázolt EFS-katasztrófával, lényeges különbséget vehetünk észre. Az EFS-magánkulcs általában egyszerűen megsemmisül, letörlődik, tehát azzal senki visszaélni nem tud. Emiatt felesleges a visszavonásával bíbelődni, úgysincs belőle egyetlen példány sem. Sőt, talán érdemesebb volna az elveszett példányt pótolni, hogy a felhasználó ott folytathassa titkosítási praktikáit, ahol abbahagyta. És itt jön be a képbe a Windows 2003 Certificate Services nagy újdonsága, a magánkulcsok központi mentésének lehetősége! Ha megsemmisül egy kulcs, állítsuk vissza!

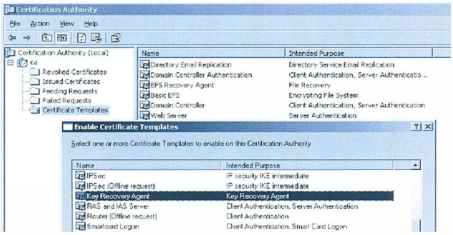
Központi kulcsarchiválás, kulcsvisszaállítás

Teljesen magától értetődik, hogy a központi kulcsarchiválást csak megfelelően biztonságos és megbízható környezetben, szigorú biztonsági házirend szabályok betartása esetén érdemes használni. Ellenkező esetben a központi kulcstár átmegy központi kulcsgyárba, és pont a CA Serveren kerül nyilvánosságra a felhasználók jelszóval védett, profilban tárolt magánkulcsa.

Apropó profil. Mielőtt belemerülnénk a központi kulcsarchiválás rejtelmeibe, érdemes megjegyezni, hogy csak azokat a magánkulcsokat lehet ilyen módon archiválni, amelyek egyébként is menthetők lennének, PFX-exportálás segítségével. Tehát eleve kiesnek a körből a Smart Cardon generált, illetve az exportálhatatlan típusú kulcspárok és tanúsítványok. Ennek oka abban keresendő, hogy központi archiválás ide vagy oda, az RSA-kulcspárok az ügyfélnél generálódnak, és onnan kerülnek archiválásra, ha ezt a kulcsok kezelése egyáltalán lehetővé teszi.
Az EFS szinte kiált a központi kulcsarchiválásért, mivel eleve nem képes együttműködni Smart Carddal (csak tudnám, miért?), és mint már említettem, a kulcs általában nem nyilvánosságra kerül, hanem elporlik a felhasználó keze között.

Key Recovery Agent

A központi kulcsarchiválás legeslegelső lépése, hogy kijelöljük azt a személyt vagy személyeket, akik katasztrófa esetén képesek lesznek a titkosítva tárolt magánkulcsot kinyerni a rendszerből, és odaadni a Vezérnek. Ők lesznek a Key Recovery Agentek (KRA), kulcsvisszaállító ügynökök. Amennyiben egynél több KRA-t jelölünk ki, számolnunk kell azzal, hogy egy KRA önmagában nem képes visszaállítani a mentett kulcsokat (lásd erről a TechNet tavalyi számait). Úgy válik egy normális Windows-felhasználó KRA-vá, hogy – stílszerűen – szert tesz egy KRA típusú X.509-es tanúsítványra. Ezt a tanúsítványtípust a CA Server alapfelállásban nem bocsátja ki, erre meg kell kérnünk őkelmét:

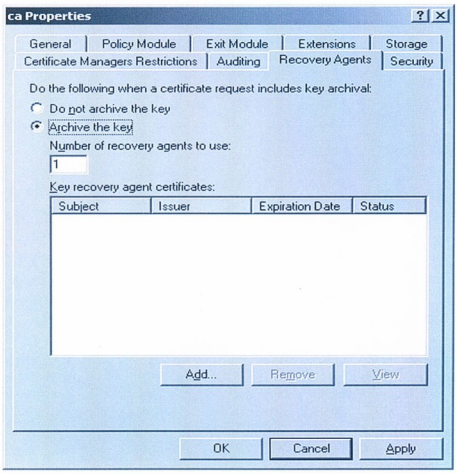


Key Recovery Agent tanúsítvány-típus engedélyezése a CA-n

Ha valaki megnézné, kinek is van joga ilyen típusú tanúsítványt igényelni, azt tapasztalná, hogy a Domain Admins, Enterprise Admins csoport tagjai válhatnak KRA-vá. Ez nyilván túl népes társaság. Ha megkurtítjuk eziránya jogaikat, visszaadják maguknak, hisz azért admin az admin, hogy azt is megtehesse, amit nem tehet meg! De nem kell ettől megijedni, mivel további védelmi vonal állja útját ezeknek a „gazfickóknak": a KRA-tanúsítványt a CA nem adja ki automatikusan, hanem egy CA Manager elfogadásától teszi függővé a kibocsátást. A CA Manager szerepet pedig teljesen külön lehet választani a többi Windows-csoportól az úgynevezett Role Separation segítségével (ami egyébként Common Criteria-követelmény a CA-rendszerrel kapcsolatban). Amíg egy CA Manager el nem fogadja a tanúsítványkérést, addig az a Pending Requests mappában szunnyad, és a KRA nem lép életbe. Most tételezzük fel, hogy a tanúsítvány elfogadásra került. A következő lépés a CA Server felkészítése az archiválásra.

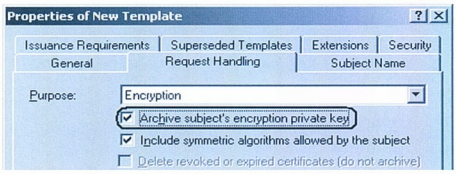
A kulcsarchiválás bekapcsolása

A CA Server tulajdonságlapján lehet beállítani, kik lesznek azok a KRA-k, akiknek a nyilvános kulcsával minden archivált magánkulcs titkosításra kerül. Érdemes megfontolni, hogy egynél több KRA-t jelöljünk ki, hisz a körúti butikot sem nyithatja ki egy lyány, meg kell várnia a kollegina megérkezését. Egy CA-rendszer összes magánkulcsa pedig X nagyságrenddel értékesebb, mint az a néhány falatnyi női holmi, amihez egynél több „ügynök" jelenlétében lehet csak hozzáférni. Az én kis példámban csak egyetlen KRA lesz, de ez csak a demójelleg miatt van így!



KRA-k kijelölése a kulcsarchiválásra

Itt értelemszerűen az Add... nyomógombbal lehet felvenni KRA-k tanúsítványát, míg a „Number of recovery agents to use:" az egyidejűleg szükséges butikoslányok száma.
Az OK gomb megnyomása és a CA újraindulása után a CA Server kulcsarchiválásra kész. Nem így a tanúsítványok! Minden egyes kulcsarchiválásra kijelölt tanúsítványsablonon be kell pipálni az archiválást engedélyezését:



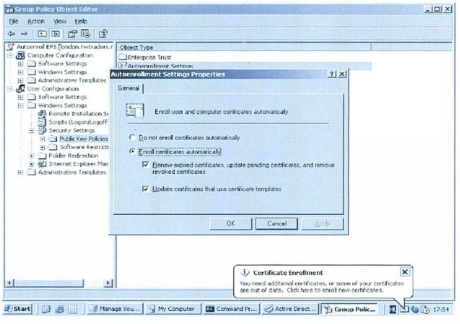
Magánkulcs archiválásának engedélyezése a tanúsítványsablonon

És kész – lenne, ha a Basic EFS tanúsítványsablon ismerné ezt a lehetőséget. Mivel azonban az EFS egy „ősi" szolgáltatás a sötét XX. századból, az ő sablonja feleannyi beállítást tartalmaz, mint szerencsésebb utódai. Nincs más hátra, alkotni kell egy új, Windows 2003-as verziójú EFS-tanúsítványsablont!

Ennek részleteivel nem fárasztanám a Tisztelt Olvasót. Akit komolyan érdekel a kulcsarchiválás és a PKI-rendszerek üzemeltetése, úgysem ússza meg a négynapos PKI Workshop tanfolyamot. Folytassuk tehát az elvekkel.
Elvileg az új sablon elkészítésével feltettük az i-re a pontot. Ugyanakkor nem árt megjegyezni, hogy a munkaállomások csak akkor fárasztják magukat új EFS-tanúsítvány beszerzésével, ha még nincs nekik olyan. De pont a legturbóbb felhasználókra ez nem igaz, már rég van EFS-kulcspárjuk, hisz hónapok óta titkosítanak. A magánkulcs archiválását viszont csak egy új sablonnal készült új kulcspár teszi lehetővé. Hogyan cseréljük le egy távoli gépen az ott meglévő kulcspárokat és tanúsítványokat? Group Policyval!

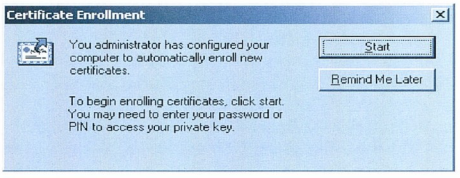
Tanúsítványok automatikus kibocsátása

Szerencsére a Windows 2003 Group Policy már felhasználók számára is lehetővé teszi az automatikus tanúsítványkiosztást. Ezt a lehetőséget fogjuk kiaknázni az új EFS-sablonunk terítésére. Az alábbi ábrát úgy próbáltam összehozni, hogy rajta legyen minden fontos jel ahhoz, hogy Önök is megtalálják a beállítást. Külön érdekessége az ábrának, hogy pont akkor csapott bele az automatikus tanúsítványigénylés a Windowsba, amikor reszketű ujjamat a PrtScr gomb fölé vittem. Balra lent, a tálcán látszik a buborékot dobó ikonocska, mely új tanúsítvány eljövetelére figyelmeztet. (Teljesen suttyomban, a háttérben is működik, pusztán a látványosság kedvéért állítottam be úgy a házirendet, hogy kezdjen buborékozni, amint új tanúsítvány igényelhető.)



Az automatikus tanúsítványigénylés GPO-beállítása

Ha a felhasználó a buborékra kattint, megindul a kulcsgenerálás, sőt, kulcsarchiválás folyamata.

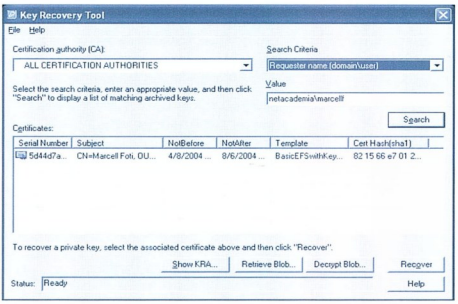


Automatikus tanúsítványkiosztás

Egy nagyon fontos dologról ne feledkezzünk meg: az automatikus tanúsítványgeneráláshoz nem elegendő önmagában a csoportházirend megfelelő beállítása. Jogosultság is kell hozzá. Minden egyes AutoEnroll-tanúsítványsablonon Read, Enroll és Autoenroll jogot kell adni a célszemélyeknek – a mi esetünkben egyszerűen az Authenticated Usersnek. Mostantól bárki bátran elhagyhatja EFS-kulcsait, lesz aki visszaadja neki: a KRA! De hogyan?

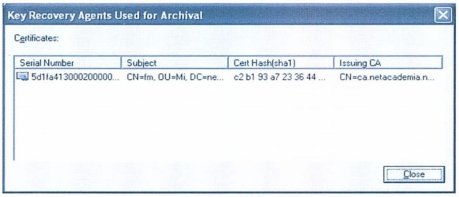
Elveszett kulcsok visszaállítása

2003-ban már megjelent egy cikk a TechNetben kulcsvisszaállításról Fülöp Miklós tollából. Ő a beépített parancssori lehetőséget taglalta. Én most inkább a sokkal könnyebben használható grafikus felületet mutatom meg. A Windows 2003 Resource Kitben található a KRT.EXE (Key Recovery Tool) program, amivel (a KRA magánkulcsának birtokában!) játszva visszaállíthatjuk Vezér Elvtárs vagy Gipsz Jakab elveszett magánkulcsait.
A sokféle tanúsítványkeresési módszer közül a domain\usert választottam, és megkerestem a saját archivált kulcsaimat:



Archivált kulcspárok listája

Ha egynél több KRA lett beállítva, most megtudhatjuk, hogy melyiküket kell berángatni szabadságról, ha a Show KRA gombra kattintunk. Mint kiderült, az én másik felhasználói fiókom (fm) az a KRA, akire szükség lesz:



Itt olvasható, melyik KRA-ra van szükség a visszaállításhoz

A Recover gombra kattintva pedig elkészíthetjük azt a PKCS 12-es (.PFX) fájl, amivel a hiányzó kulcsok a felhasználó számítógépén pótolhatók.

Föti Marcell
marcellf@netacademia.net
MCT, MCSE, MCDBA, MZ/X

Arccal az MCSA felé!

Akciónk keretében a Microsoft Windows 2003 rendszeradminisztrátor minősítés (MCSA) megszerzéséhez szükséges 3 + 1 tanfolyamot megrendelők részére cégünk a vizsgákat ingyenesen biztosítja.

Az MCSA minősítés előnyei:

- Az Ön szakmai felkészültségének elismerése az iparágon belül.
- Közvetlen hozzáférés az MCP tagok weboldalán keresztül a Microsoft legfrissebb technikai és termékinformációihoz.
- Az adott minősítést igazoló logók elérhetősége és használata (szabályozott kereteken belül).
- Ingyenes vagy kedvezményes meghívások itthoni és külföldi konferenciákra, tanulmányutakra, rendezvényekre.
- Ingyenes hozzáférés a Microsoft MCP Online Magazine-ra és egyéb minősítés-specifikus számítástechnikai kiadványokhoz.
- A kedvezményes előfizetés a Windows & .NET Magazinokra.
- Illetve néhány közvetett előny (például karrierlehetőségek itthon és külföldön).

Az MCSA-képesítés a következő feladatok elvégzésére készít fel:

- Microsoft Windows Server 2003 környezetek menedzselése és hibaelhárítása.
- Active Directory kialakítása és üzemeltetése.
- Group Policy Objektumok létrehozása és konfigurálása. Felhasználói környezet kezelése csoportos házirenddel.
- Erőforrások hozzáférésének kezelése.
- A kiszolgáló teljesítményének monitorozása.
- Katasztrófa előtti állapot visszaállításának kezelése (Disaster Recovery).
- Szoftverek karbantartása Software Update Services használatával.
- TCP/IP alapú hálózati környezetek kialakítása.
- Biztonságos hálózat kialakítása.
- Munkaállomások telepítése és konfigurálása.
- Az új ISA Server 2004 konfigurálása és üzemeltetése.

A részvétel feltételei:

- A résztvevőknek egy éven belül el kell végezniük mind a négy MCSA tanfolyamot.
- Az ingyenes vizsgabónt mindig a következő tanfolyam megrendelésekor adjuk ki (tehát az elsőt a második tanfolyam megrendelése után, a 2. bónt a 3. tanfolyam után stb.).
- **Jelentkezési határidő: 2004. szeptember 15.**

Az összeállítás csak egy a lehetőségek közül.
A teljes kínálatról tájékozódjon honlapunkon!

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®