www.DelphiDeveloper.hu

Delphi developer

1. évfolyam 5. szám
2005. május
5

Delphi programozók havilapja
megjelenik minden hónap első napján

9 771786 224805 00505
Ára: 1576 Ft

Windows mindenütt

Windows-os alkalmazásfejlesztés Delphi-vel

Kültéri hőmérséklet mérése programból

Ablak animáció

Egy számítógép felhasználói

Program futtatása meghatározott processzoron

USB eszközök csatlakoztatásának figyelése programból

Hálózati információk gyűjtése

Üresjárati idő figyelése

Üzenet küldése alkalmazások között

Színkódok meghatározása

♦ Egér műveletek követése ♦ Felhasználók egyedi azonosítója ♦
♦ Szolgáltatás státuszának lekérdezése ♦ Átkapcsolás programok között programból ♦
♦ Hangállományok párhuzamos lejátszása ♦ Form méretének korlátozása ♦
♦ Felhasználók felvétele, törlése programból ♦ Kódlap információk ♦
♦ Milyen levelezo kliensek vannak telepítve? ♦ Delphi alól fut-e a program? ♦

Végkiárusítás!

30-50% kedvezmény minden könyvre amíg a készlet tart

A megrendeléseket a beérkezésük sorrendjében teljesítjük, amíg a készlet tart.
A könyvekből utánnyomás már nem készül.

webshop.animare.hu

iroda@animare.hu (30) 566 7110

Delphi Developer

Delphi programozók havilapja
megjelenik minden hónap első napján

Főszerkesztő: Püspöki Zoltán
Főszerkesztő-helyettes: Ónozó Péter
Kiadóvezető: Szigetvári Stefánia
Projektvezető: Fehér Ferenc
Tördelőszerkesztő: Király Zita
Olvasószerkesztő: Keszthelyi Noémi
Grafika: Végvári Andrea
Fotó: Szentei András
Szerzők: Ábrahám Lajos , Alabárdos András, Ámbrás Péter, Bakonyi János, Borbély István, Delák Péter, Farkas Sámuel, Garami Fülöp, Győri Béla, Hoplár Péter, Károlyi Péter, Katus Gábor, Kendes Máté, Medve László, Miksa Előd, Pálfalvi Péter, Prethus Gábor, Tália Gábor

Szerkesztőség, hirdetésfelvétel:
Animare Software Kft.
Cím: 7625 Pécs, Surányi út 12.
Telefon: +36 (30) 566 7110
Web: http://www.animare.hu
E-mail: iroda@animare.hu

A kiadvány kereskedelmi forgalomban nem kapható. A kiadvány előfizethető a havilap webhelyén, illetve a fenti szerkesztőségi címeken, telefonszámon. Az előfizetett havilapot minden hónap elején díjmentesen postázzuk. A lap régebbi számai is megrendelhetők a fenti címeken, telefonszámon.

Szerkesztőségünk a havilapban közölt tartalmat és a mellékelt forráskódokat a legnagyobb körültekintéssel gondozza. Ennek ellenére sem a szerzők, sem a kiadó nem vállal semminemű felelősséget vagy garanciát, hogy a lapban foglaltak, illetve a mellékelt forráskódok nem tartalmaznak hibákat, így ezeket mindenki csak saját felelősségére használhatja. Szerkesztőségünk a lapban közölt hirdetések tartalmáért nem vállal felelősséget.

ISSN 1786-2248

Windows

Windows-os alkalmazásfejlesztés Delphi-vel

A Windows operációs rendszer telepítését követően számos DLL-t találunk a rendszerben. Ezekben sok olyan függvény bújik meg, melyeket nem deklaráltak a Delphi unit-jaiban, sokuknak még csak leírása sem áll rendelkezésre a Delphi súgójában. E függvény persze ettől még használhatók, csak magunknak kell deklarálni azokat. A rendszer DLL-ekben lévő függvények sokasága közül válogatunk most néhányat aktuális havilapunk hasábjaira és vizsgáljuk meg azokat közelebbről, hogy miként is használatók saját alkalmazásainkba integrálva.

A lapban számos érdekes cikk található az alábbi témakörökben:

- Form ideiglenes bezárása
- Melyik kiterjesztéshez melyik alkalmazás van rendelve?
- Delphi alól fut-e a program?
- Üzenet küldése alkalmazások között
- Kódlap információk
- Ablak animáció
- Üresjárati idő figyelése
- Milyen levelező kliensek vannak telepítve?
- Színkódok meghatározása
- USB eszközök csatlakoztatásának figyelése programból
- Egér müveletek követése
- Felhasználók egyedi azonosítója
- Szolgáltatás státuszának lekérdezése
- Átkapcsolás programok között programból
- Felhasználók felvétele,
- törlése programból
- Form méretének korlátozása
- Program futtatása meghatározott processzoron
- Hangállományok párhuzamos lejátszása
- Hálózati információk gyüjtése
- Felhasználók ki, be és átjelentkezéseinek figyelése
- Egy számítógép felhasználói

Püspöki Zoltán
Animare Software

Tartalom

Delphi

2005. május

Delphi

Letöltés

Az újság weboldaláról minden előfizető letöltheti egyetlen tömörített állomány formájában az aktuális havilap cikkeihez tartozó forráskódokat a webhely Letöltés menüpontja alatt elérhető weblapról.

Keresés

A www.DelphiDeveloper.hu oldalon a Keresés menüpont alatt lehetőség van minden már megjelent havilap tartalmában kulcsszavak alapján keresni. Így könnyen megtudható, hogy a keresett szó melyik havilap hányadik oldalán található.

Havilap

Minden megjelent havilap bármely oldala megtekinthető a Delphi Developer webhelyén is. Így az sem lehet gond, ha éppen nincs kéznél az újság, de szükség lenne abból valamilyen információra.

www.DelphiDeveloper.hu

Médiaajánlat

Hirdesse Ön is szolgáltatását, vagy termékét szoftverfejlesztők számára a Delphi Developer havilapban!

A Delphi Developer havilap célközönsége

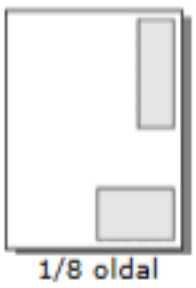
Elsősorban szoftverfejlesztők, szoftvermérnökök, rendszergazdák, magasan kvalifikált informatikai szakemberek, valamint mindazok köre, akik érdeklődnek az új programfejlesztési technológiák, új alkalmazások iránt.

Delphi Developer lapinformációk

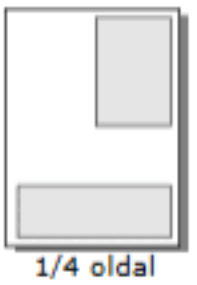
Megjelenik minden hónap első napján
Első megjelenése: 2005. január
Példányszám: 4 000 db
Méret: 200 x 270 mm
Terjedelem: 80-120 oldal
Formátum: 4+4 szín borító, fekete-fehér belív

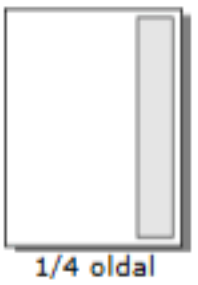
Hirdetésfelvétel

Animare Software Kft.
E-mail: iroda@animare.hu; Telefon: (30) 566 7110
A hirdetés megjelentetésének feltétele, hogy lapzártáig a hirdetési díj megérkezzen cégünk bankszámlájára. A hirdetési anyagot elektronikus formában tudjuk fogadni e-mailben, vagy CD lemezen. Technikai adatok: A hirdetés az alábbi méreteknek megfelelő 300 dpi felbontású BMP, TIF vagy CPT állomány. Kifutó méretű hirdetés esetén az alábbi megjelenési méreteket +10 mm-rel növelni kell. E-mail-ben történő küldés esetén kérjük a képeket tömöríteni (ZIP, RAR).

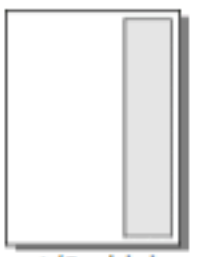

1/8 oldal
40 x 123 mm
80 x 59 mm
9 000 Ft

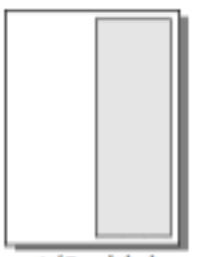

1/6 oldal
54 x 123 mm
175 x 38 mm
15 000 Ft

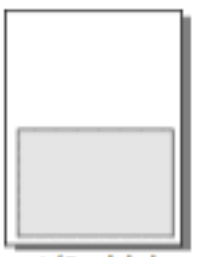

1/4 oldal
80 x 123 mm
175 x 59 mm
23 000 Ft


1/4 oldal
40 x 250 mm
22 000 Ft


1/3 oldal
175 x 79 mm
30 000 Ft


1/3 oldal
54 x 250 mm
30 000 Ft


1/2 oldal
80 x 250 mm
46 000 Ft


1/2 oldal
175 x 123 mm
46 000 Ft


2/3 oldal
175 x 165 mm
61 000 Ft


1 oldal
200 x 270 mm
108 000 Ft


2 oldal
2 x 200 x 270 mm
191 000 Ft


Borító (színes)
II. 200 x 270 mm 213 000 Ft
III. 200 x 270 mm 207 000 Ft
IV. 200 x 270 mm 219 000 Ft

Fenti hirdetési díjak a 25% ÁFÁ-t nem tartalmazzák!

Kis hírek a nagyvilágból

Visual Studio 2005 Beta 2

Elérhető a Visual Studio 2005 Beta 2, a Microsoft .NET Framework 2.0 Beta 2 és az SQL Server 2005 áprilisi Community Technology Preview (CTP).

Az új Visual Studio.NET 2005 és SQL Server 2005 újabb jelentős mérföldkő lesz a szoftverfejlesztés terén. A két termék szoros integrációja révén, valamint a .NET Framework 2.0 számos újdonsága segítségével a szoftverfejlesztés eddig megszokott menete sok tekintetben átalakul, jelentősen egyszerűsödik és ezáltal még inkább hatékonyabbá válik.

A Visual Studio.NET és SQL Server mostani béta verziója már Go Live licencprogram keretén belül jelenik meg. Ennek jelentősége, hogy az alkalmazásokat már éles helyzetben is alkalmazhatjuk. Így már most készíthetünk az új termékekkel ASP.NET-esemény, Windows Forms, Office alapú alkalmazásokat, sőt építhetünk a Windows Mobile alapú eszközöket támogató szoftvereket is a Compact Framework segítségével.

↳ 051

64 bit

Megjelentek a Windows XP és Windows Server 2003 operációs rendszerek 64-bites változatai.

↳ 050

Angol-Magyar szótár

Angol-Magyar és Magyar-Angol szótárral bővíthetjük ki Office alkalmazásainkat, sőt még az Internet Explorer-t is!

A kiegészítő programcsomag díjmentesen letölthető és telepíthető az alábbi webcímről:

↳ 052

Longhorn - Avalon

Ingyenesen letölthető egy 56 perces videofilm, melyben a Windows új verziójának a grafikus alrendszere, az Avalon kerül bemutatásra.

↳ 053

Windows Server 2003 SP1

Megjelent a Windows Server 2003 operációs rendszer első javítócsomagja. A javítás ingyenesen letölthető az alábbi webcímről, ahol számos dokumentum is megtalálható a javítócsomaggal kapcsolatban:

↳ 057

Windows Server 2003 SP1 Platform SDK

Megjelent a Windows Server 2003 SP1 Platform SDK 32 és 64 bitre egyaránt. Az SDK számos programozói dokumentációt, példaprogram forráskódot, header állományokat és segédprogramokat tartalmaz. A programcsomag 408 MB.

↳ 054

.NET Framework Version 2.0 Redistributable Package Beta 2

Letölthető az új .NET Framework terjeszthető csomagja 32 és 64 bites változatban egyaránt. E telepítővel a végfelhasználók számítógépére telepíthető a .NET Framework.

↳ 055
↳ 056

Windows regisztrációs adatbázis mentés, szerkesztés, visszaállítás

"A rendszerleíró adatbázis biztonsági mentése, szerkesztése és visszaállítása Windows XP és Windows Server 2003 rendszerben" címmel jelent meg egy magyarnyelvű dokumentum az alábbi webcímen:

↳ 058

Útiterv

A Windows Hardware Engineering Conference-en (WinHEC 2005) megjelent Microsoft útitervek láthatók az alábbi webcímen. A Seattle-ben megrendezett konferencia anyagából megtudhatjuk, hogy a Microsoft 2006-ra tervezi a Longhorn és a Windows Server klaszter változatának megjelenése mellett az Exchange szerver 12-es, a Virtual Server 2-es verziójának kiadását is.

↳ 059

MSN Messenger 7

Megjelent és szabadon letölthető az új MSN Messenger, mely immáron a 7.0-s verziószámmal rendelkezik.
↳060

.NET Framework 2.0

Új webhely jelent meg az MSDN-en belül, mely a .NET Framework 2.0 programozásával foglalkozik. A webhely számos leírást és letölthető forráskódokat tartalmaz, annak érdekében, hogy az új .NET Framework-re történő áttérést mielőbb elkezdhessék a programozók.
↳061

SQL Server 2005 – Áprilisi CTP

A Microsoft kiadta az SQL Server 2005 Community Technology Preview áprilisi változatát, ami a Visual Studio 2005 Beta 2-t követi. A Microsoft bejelentette, hogy nem lesz Beta 3-as változat, így a termék végleges verziója már igen közelinek mondható.
↳062

SQL Server 2005 Webcast sorozat

A Microsoft egy 22 részes webcast sorozatot indít a mai naptól az SQL Server 2005 témakörben.

Az SQL Server 2005 több újdonsággal is szolgál a .NET fejlesztők számára: bővített programozási modell, jobb biztonság, felhasználó által definiált típusok és aggregátumok, valamint a Visual Studio 2005-el integrált fejlesztőkörnyezet.

- .NET framework integráció: Avagy milyen módon használhatjuk ki a .NET keretrendszer osztálykönyvtárát alkalmazások fejlesztésekor.
- Transact-SQL és Felügyelt kód: Tudjunk dönteni, hogy mikor használjuk a tradicionális Transact-SQL és mikor a .NET keretrendszerrel kompatibilis programozási nyelvet. Tanuljuk meg mindkettő előnyeit és tudjuk őket alkalmazni.
- Web Services: Megtekinthetjük, hogy miként lehet XML webszolgáltatásokat fejleszteni az adatbázis rétegben.
- XML: Az SQL Server 2005 komoly XML feldolgozási lehetőségeket biztosít. Tanuljuk meg kihasználni ezeket az előnyöket XML manipulálással foglalkozó alkalmazásainkban is.
- Adatelérés: Avagy az ADO.NET 2.0 újdonságai.

↳063

Legfrissebb MSDN letöltések

- SQL Server 2005 CTP - April 2005 - Express Manager (English)
- Visual Studio 2005 Team Edition Test Load Agent Beta 2 (English)
- SQL Server 2005 CTP - April 2005 - Developer Edition - 64 bit Extended (English)
- SQL Server 2005 CTP - April 2005 - Developer Edition (English)
- SQL Server 2005 CTP - April 2005 - Express Edition (English)
- Windows XP Professional x64 Edition (Japanese)
- Visual Basic 2005 Express Beta 2 (English)
- Visual C# 2005 Express Beta 2 (English)
- Visual C++ 2005 Express Beta 2 (English)
- Visual J# 2005 Express Beta 2 (English)
- Visual Web Developer 2005 Express Beta 2 (English)
- Visual SourceSafe 2005 Beta 2 (English)
- Visual Studio 2005 Standard Beta 2 (English)
- Visual Studio 2005 Team Foundation Server Beta 2 (English)
- Visual Studio 2005 Team Suite Beta 2 (English)
- Windows Server 2003 with Service Pack 1 System Resource Manager (English)
- Windows Server 2003 Enterprise Edition with SP1 - VL (English)
- Windows Server 2003 Enterprise x64 Edition - VL (English)
- Windows Server 2003 Standard Edition with SP1 - VL (German)
- Windows Server 2003 Standard x64 Edition - VL (English)
- Windows Server 2003 Web Edition with SP1 - VL (English)
- Windows XP Professional x64 Edition (English)
- Visio 2002 Service Pack 2 (German)
- Windows SharePoint Services SDK (English)
- Log Parser 2.2 (English)
- Great Plains 8.0 Service Pack 2 (English)
- Great Plains 8.0 Service Pack 2 Installation Guide (English)

Kültéri hőmérséklet mérése programból

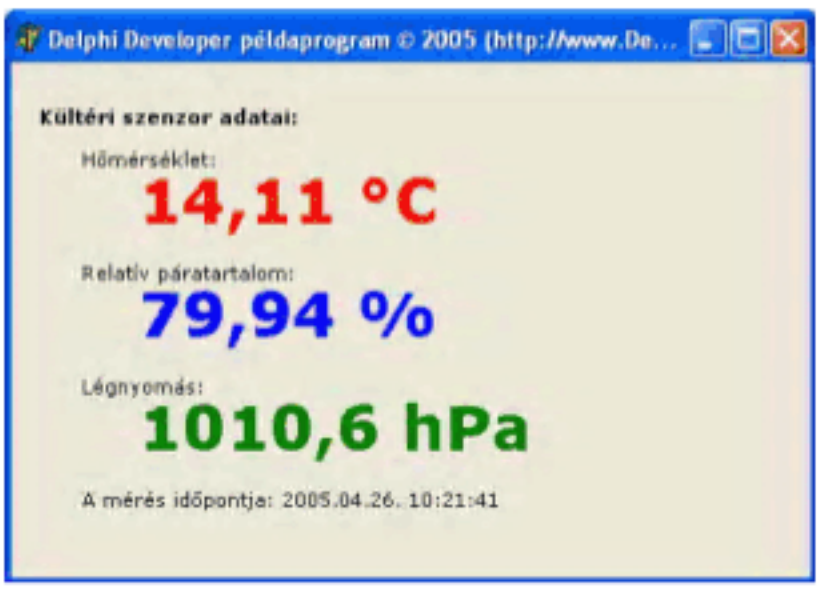
Cikk sorszáma	Szoftverfeltételek
084	➢ Delphi ➢ HOM
Forráskód	\Source\Temperature\

Hogyan jelenítsük meg, dolgozzuk fel egyedi módon az aktuális hőmérséklet, a páratartalom és légnyomás értékeket saját programmal? E kérdésre ad választ jelen cikk mellékleteként készített egyszerű kis példaprogram.

A lekérdezett adatokat csupán megjelenítjük az alkalmazás Form-ján, de persze ennél jóval bonyolultabb megoldásokat is megvalósíthatunk: az adatokat tárolhatjuk egyedi nyilvántartásunkban, feldolgozhatjuk tetszés szerint, vagy akár továbbíthatjuk az interneten keresztül saját webhelyünkre is publikálás céljából.

A Hardware Online Manager (HOM) alkalmazás telepítése szükséges a példaprogram használatához. A szoftvercsomag ingyenesen letölthető a cikk végén található webcímről.

Az aktuális mérés adatainak megjelenítése saját programban



Az aktuális mérés adatainak megjelenítése saját programban

Az aktuálisan mért adatok elérése igen egyszerű feladat: nincs másra szükségünk, mint futtatni egy select lekérdezést az SQL adatbázisra, ahol tárolásra kerülnek a mérési értékek.

A HardwareOnline adatbázis eléréséhez ellenőrizzük és szükség esetén módosítsuk a mellékelt példaprogramban található ADOQuery1 komponens ConnectionString tulajdonságát:

```
Provider=SQLOLEDB.1;Integrated
Security=SSPI;Persist Security
Info=False;Initial
Catalog=HardwareOnline;Data
Source=localhost
```

A legfrissebb mérési adatokat az alábbi select lekérdezéssel határozhatjuk meg:

```
SELECT      TOP 1 *
FROM          [Device.SensorTriple]
ORDER BY MeasurementDateTime DESC
```

A lekérdezés minden tárolt adatot visszaad.

Az eredményhalmazt rendezzük a mérési időpontra (MeasurementDateTime) csökkenő időrend szerint, így a legfrissebb mérés kerül a lista elejére.

A TOP 1 módosító kulcsszó használatával pedig elérjük, hogy listából csak az első sort kapjuk vissza. Így mindig a legfrissebben mért adatok állnak majd rendelkezésünkre.

Abban az esetben, ha több szenzor is található a rendszerben, akkor természetesen van lehetőség arra is, hogy egy-egy szenzornál külön-külön kérdezzük le a legfrissebb adatokat.

Ekkor a fenti select lekérdezés úgy módosul, hogy azt ki kell egészítenünk egy szűrő feltétellel.

A szűrést az adott szenzor sorszámára és típuskódjára kell elvégeznünk.

E két adat alapján bármilyen eszközt egyedileg azonosíthatunk.

Az eszköz sorszáma az Address, míg típuskódja a TypeCode adatoszlopban van tárolva.

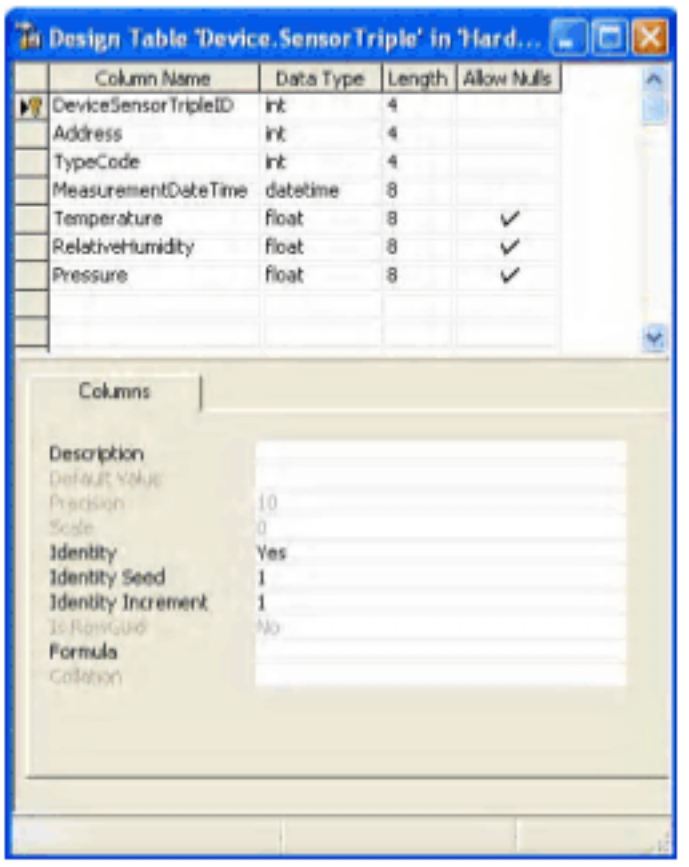
Minden eszköz hátoldalán található címkén szerepel az eszköz azonosítója, például: "5-16".

Ebből tudhatjuk meg, hogy az 5 az azonosító száma, míg a 16 a típuskódja.

E két szám ismeretében már el tudjuk készíteni a szűréssel rendelkező lekérdezést, mely csak erre az egy eszközre adja meg a kérdés információt.

```
SELECT      TOP 1 *
FROM        [Device.SensorTriple]
WHERE       (Address = 5) AND (TypeCode = 16)
ORDER BY  MeasurementDateTime DESC
```

A lekérdezett adatok értelmezése



Device.SensorTriple adattábla szerkezete

A lekérdezett adatok értelmezéséhez ismernünk kell a tárolt adatok szerkezetét. A szenzorok által tárolt adatok a Device.SensorTriple nevű adattáblába kerülnek tárolásra. E táblának a szerkezete a következő:

Adatoszlop neve	Típusa	Leírása
DeviceSensorTripleID	int	elsődleges kulcs
Address	int	az eszköz egyedi azonosító száma
TypeCode	int	az eszköz típuskódja
MeasurementDateTime	datetime	a mérés dátuma, időpontja
Temperature	float	Mért hőmérséklet értéke
RelativeHumidity	float	Mért relatív páratartalom értéke
Pressure	float	Mért légnyomás értéke

Lekérdezés megvalósítása

Mivel a mérések percenként kerülnek végrehajtásra ezért célszerű egy Timer komponenst felhasználni és percenként futtatni újra és újra a lekérdezést. E módszer révén mindig a legfrissebben mért adatokat jeleníthetjük a Form-ra helyezett Label-eken.

```
procedure TForm1.Timer1Timer(
  Sender: TObject);
begin
```

A lekérdezés idejére átállítjuk a kurzort, ezzel is jelezve a felhasználó számára, hogy programunk éppen műveletet végez.

```
  Cursor:=crHourGlass;
```

Megnyitjuk a már előkészített adatbázis kapcsolatot és futtatjuk a select lekérdezést.

```
  ADOQuery1.Open;
```

Kiolvassuk a legfrissebb hőmérsékletet és megjelenítjük annak értékét a Label2 komponensen.

```
  Label2.Caption:=ADOQuery1Temperature.
    AsString + ' °C';
```

Kiolvassuk a legfrissebb relatív páratartalmat és megjelenítjük annak értékét a Label3 komponensen.

```
Label3.Caption:=ADOQuery1RelativeHumidity.
    AsString + ' %';
```

Végül kiolvassuk a legfrissebb légnyomást és megjelenítjük annak értékét a Label4 komponensen.

```
Label4.Caption:=ADOQuery1Pressure.AsString
    + ' hPa';
```

A Label5 komponensre kiírjuk, hogy mikor történt a fenti értékek mérése.

```
  Label5.Caption:='A mérés időpontja: ' +
    ADOQuery1MeasurementDateTime.AsString;
```

Végül zárjuk az adatbázis kapcsolatot és visszaállítjuk a kurzort az alapértelmezettre.

```
  ADOQuery1.Close;
  Cursor:=crDefault;
end;
```

Mivel a Timer komponensbe 60 másodperces időzítést adtunk meg, így a program indulását követően csak az egy perc letelte után jelenne meg adat a Form-on. Ez nyilván nem megfelelő megoldás, így amikor a Form megjelenik, annak OnShow eseményénél szintén aktivizáljuk a Timer komponens OnTimer eseményhez rendelt Timer1Timer nevű eseménykezelő eljárást. Ennek révén a program indulását követően a Form-ra helyezett Label-ek azonnal mutatják a legfrissebb mérések eredményeit.

```
procedure TForm1.FormShow(Sender: TObject);
begin
  Timer1Timer(nil);
end;
```

Prethus Gábor

Kapcsolódó webhelyek

HOM alkalmazás letöltése ↳ 007

Form ideiglenes bezárása

Cikk sorszáma	Szoftverfeltételek
085	➢ Delphi
Forráskód	\Source\HitTest\

E példában megvalósítandó feladatunk az lesz, hogy amikor a felhasználó a Form fejlécén jobb egérgombbal kattint, akkor a Form kliens területének magasságát animálva állítsuk nullára. Egy újabb kattintásnál pedig állítsuk vissza az eredeti állapotot.

A feladat megoldásához tudnunk kell kezelni azt az eseményt, hogy a felhasználó mikor és milyen egérgombbal kattint a Form fejlécének területére. Mivel a Form OnMouse... kezdetű eseménye csak a Form kliens területére vonatkozik, így e feladathoz ezeket nem tudjuk alkalmazni, más megoldást kell hát keresni.

A mellékelt példaprogram kipróbálásához, annak futtatása után kattintson egymás után többször is jobb egérgombbal a Form fejlécén.

A megvalósítás alapja

WM_NCRBUTTONDOWN üzenet

A feladatunk megvalósításának alapját a WM_NCRBUTTONDOWN üzenet megérkezésének figyelése adja. Ez az üzenet akkor érkezik a Form-unkhoz, ha a felhasználó annak kliens területen kívüli részére kattint az egér jobb gombjával.

Tudnunk kell erről az üzenetről még azt is, hogy amennyiben ez egy olyan terület lenne, mely képes birtokolni a fókuszt, akkor az üzenet nem kerülne elküldésre. Mivel a Form fejléce erre nem képes, így az üzenet megérkezik a Formhoz.

Mint minden üzenet, a WM_NCRBUTTONDOWN is tartalmaz két paramétert az alábbiak szerint:

```
WPARAM wParam
LPARAM lParam;
```

- wParam

A nem kliens területen történt kattintás pontos helyét leíró érték. A wParam lehetséges értékeire az alábbiakban visszatérünk.

- lParam

Az lParam paraméter egy POINTS struktúrában tárolja azt az X, Y koordinátát, mely az adott egér eseményhez köthető. A koordináta képernyő koordinátarendszerben értelmezendő, vagyis annak bal felső sarka a 0, 0 pont. Az lParam alsó szava az X, a felső szava az Y koordinátát adja.

wParam lehetséges értékei

Érték	Leírás
HTBORDER	Az egér művelet az ablak keretén történt, annak is azon részén, melyen átméretezés nem végezhető.
HTBOTTOM	Az egér művelet az ablak alsó szélén történt, ahol az ablak függőlegesen átméretezhető.
HTBOTTOMLEFT	Az egér művelet az ablak bal alsó sarkában történt, ahol az ablak átlósan átméretezhető.
HTBOTTOMRIGHT	Az egér művelet az ablak jobb alsó sarkában történt, ahol az ablak átlósan átméretezhető.
HTCAPTION	Az ablak fejlécén.
HTCLIENT	Az ablak kliens területén.
HTCLOSE	Az ablak Close gombján.
HTERROR	A képernyő hátterén.
HTGROWBOX	Az ablak jobb alsó méretező dobozán.
HTHELP	Az ablak Help gombján.
HTHSCROLL	A vízszintes görgetősávon.
HTLEFT	Az egér művelet az ablak bal szélén történt, ahol az ablak vízszintesen átméretezhető.
HTMENU	Az ablak menüjén.
HTMAXBUTTON	A maximalizáló gombon.
HTMINBUTTON	A minimalizáló gombon.
HTNOWHERE	A képernyő hátterén.
HTREDUCE	A minimalizáló gombon.
HTRIGHT	Az egér művelet az ablak jobb szélén történt, ahol az ablak vízszintesen átméretezhető.
HTSIZE	Lásd: HTGROWBOX
HTSYSMENU	Az ablak rendszer menüjén, vagy gyerek ablak esetén a Close gombján.
HTTOP	Az egér művelet az ablak felső szélén történt, ahol az ablak függőlegesen átméretezhető.
HTTOPLEFT	Az egér művelet az ablak bal felső sarkában történt, ahol az ablak átlósan átméretezhető.
HTTOPRIGHT	Az egér művelet az ablak jobb felső sarkában történt, ahol az ablak átlósan átméretezhető.
HTVSCROLL	A függőleges görgetősávon.
HTZOOM	A maximalizáló gombon.

TWMNCRButtonDown rekord

A WM_NCRBUTTONDOWN üzenet paramétereinek feldolgozásához kapunk segítséget is a TWMNCRButtonDown rekord által. Az üzenet kezelésekor ezt használhatjuk fel a paraméterként kapott adatok kezeléséhez.

```
TWMNCRButtonDown = packed record
  Msg: Cardinal;
  HitTest: Longint;
  XCursor: Smallint;
  YCursor: Smallint;
  Result: Longint;
end;
```

- Msg: Cardinal

Az üzenet azonosítója.

- HitTest: Longint

A fenti táblázat lehetséges értékei közül valamelyik.

- XCursor: Smallint

Az egér X koordinátája.

- YCursor: Smallint

Az egér Y koordinátája.

- Result: Longint

Az üzenet feldolgozásának eredménye.

A mellékelt példaprogram

A mellékelt példaprogram

A feladat megvalósításához tehát deklarálnunk kell egy üzenet kezelő eljárást a TForm1 osztályon belül, mely képes lesz a WM_NCRBUTTONDOWN üzenet fogadására és feldolgozására.

Ezen kívül létrehozunk még egy FOldHeight változót is, melyben tároljuk a Form eredeti magasságát. Erre akkor lesz szükségünk, mikor a Form fejlécén ismételten kattint a felhasználó és ekkor helyre kell állítanunk a Form eredeti magasságát.

```
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    MonthCalendar1: TMonthCalendar;
    MonthCalendar2: TMonthCalendar;
  private
    FOldHeight: Integer;
    procedure WMNCRButtonDown(var Msg:
      TWMNCRButtonDown);
      message WM_NCRBUTTONDOWN;
  public
    { Public declarations }
  end;
```

WMNCRButtonDown eljárás

Amikor a Form-hoz befut a WM_NCRBUTTONDOWN üzenet, akkor aktivizálódik a WMNCRButtonDown eljárás.

```
procedure TForm1.WMNCRButtonDown(var
  Msg: TWMNCRButtonDown);
var
  speed,  i: integer;
begin
```

Ebben az esetben ellenőrizzük, hogy az egér jobb gombjával történt kattintás a Form fejlécén történt-e.

```
  if (Msg.HitTest = HTCAPTION) then
```

Amennyiben igen, úgy azt ellenőrizzük, hogy a Form kliens magassága egyezik-e nullával.

```
    if (ClientHeight = 0) then
    begin
```

Ha igen, akkor visszaállítjuk a Form eredeti magasságát.

```
      ClientHeight := FOldHeight;
      Application.ProcessMessages;
    end
```

Amennyiben a Form magassága még nem nulla, akkor egy ciklus segítségével folyamatosan csökkentjük a magasságot, amíg az a nullával nem lesz egyenlő. A csökkenés sebessége, vagyis a for ciklus lépés száma megadható a speed változóba. Ezzel elérhetjük, hogy a Form úgy látszik, mintha animálva becsukódna.

```
    else
    begin
      speed:=8;
      FOldHeight   := ClientHeight;
      for i:=FOldHeight div speed downto 0
        do begin
        ClientHeight := i * speed;
        Application.ProcessMessages;
      end;
    end;
end;
```

Delák Péter

Kapcsolódó webhelyek

WM_NCRBUTTONDOWN ↳ 006

Melyik kiterjesztéshez melyik alkalmazás van rendelve?

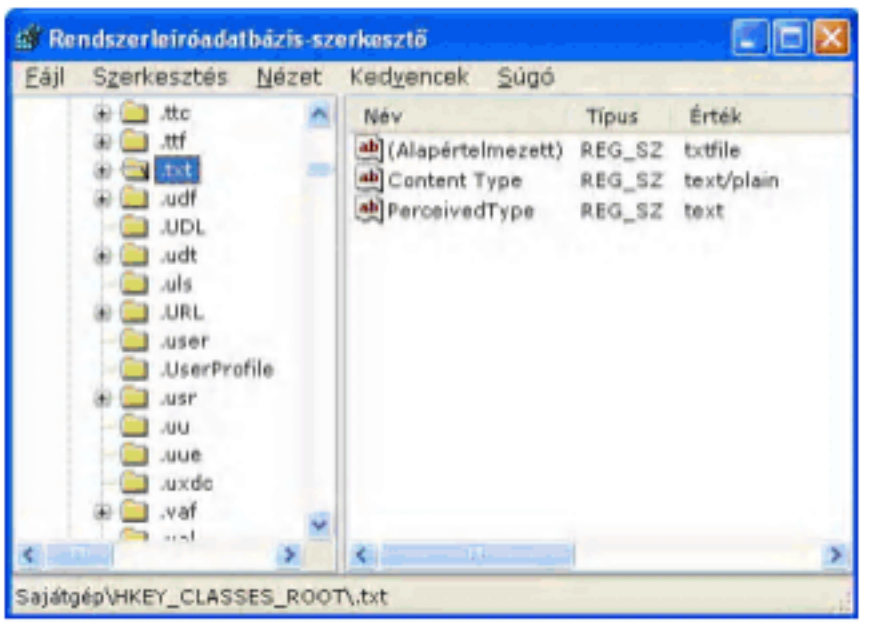
Cikk sorszáma	Szoftverfeltételek
086	➢ Delphi
Forráskód	\Source\Extensions\

Réges-régen még egy DOC állományt úgy nyithattunk meg a Word-del, hogy elindítottuk az alkalmazást, majd a megnyitás menüpontot kiválasztva elő kellett keresnünk a kérdéses dokumentumot.

Ma már teljesen hétköznapi dolog, hogy egy-egy állomány típust felismer a Windows és automatikusan elindítja az adott állományt kezelő programot, mely rögtön be is tölti a kívánt állományt. Így már nyugodtan kattinthatunk a DOC kiterjesztésű állományokra: a Word azonnal indul és a dokumentum megnyitásra kerül.

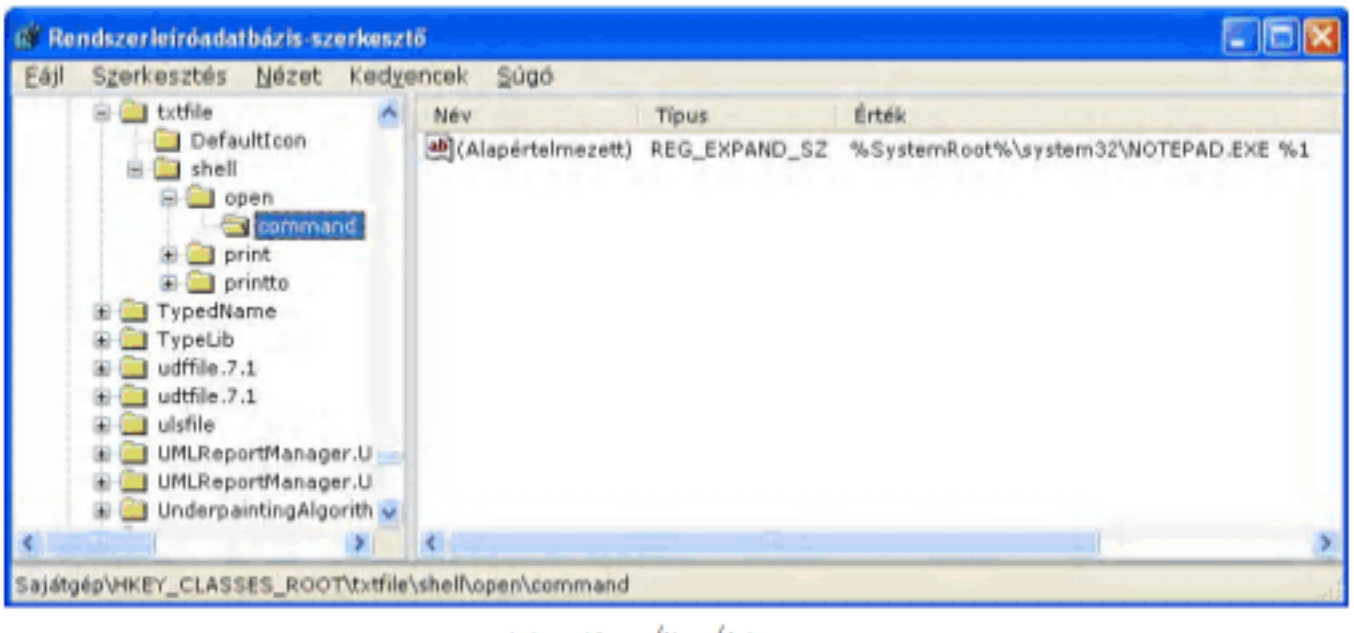
De honnan is tudja vajon a Windows, hogy milyen kiterjesztésű állományhoz milyen alkalmazás tartozik, amit ilyen esetben el kell indítani? E cikkben erre keressük meg a választ.

Windows regisztrációs adatbázis



txt kiterjesztés leírása

A Windows regisztrációs adatbázisban a HKEY_CLASSES_ROOT főkulcs alatt találjuk meg felsorolva az összes olyan állomány kiterjesztést, melyet a Windows ismer. Minden kiterjesztés leírásánál az Alapértelmezett értéknél találunk egy elnevezést. A .txt kiterjesztés esetén a txtfile elnevezés található a Windows regisztrációs adatbázisban. Ez alapján tudjuk meg azt, hogy melyik lesz a következő kulcs, amit elő kell keresnünk. Jelen esetben tehát nézzük mit is találunk a txtfile bejegyzésnél.



txt után indítandó program

Itt találjuk azokat az alapvető műveleteket, melyeket egy-egy dokumentummal, állománnyal végezhetünk. A txtfile-on belül találjuk a shell nevű bejegyzést, ahol az alapműveletek leírása található. Az open kulcsnál lesz a megnyitásra vonatkozó parancs, míg a print bejegyzésnél a dokumentum nyomtatásához szükséges értékeket találjuk. Az open kulcsra visszatérve és azt kinyitva megtaláljuk a command nevű bejegyzést. Ennek alapértelmezett értékénél lesz leírva az, hogy a dokumentum kezeléséhez mely programot kell aktiválni. Ez jelen esetben a Notepad.exe lesz. A program megadását követő %1 helyére a Windows behelyettesíti az aktuálisan kiválasztott állományt. Így amikor a Windows Intézőben egy TXT állományon kattintunk, akkor a Windows a regisztrációs adatbázisból előkeresi a TXT-hez tartozó programot, amit elindít, paraméterként megadva számára, hogy mely állományon történt a kattintás. A Notepad indulása után vizsgálja, hogy kapott-e indítási paraméterként állományt és ha igen, akkor azt automatikusan betölti.

Ehhez hasonlóan zajlik a nyomtatás. Amikor egy TXT állományon jobb gombbal kattintunk, akkor elérhető a Nyomtatás menüpont. Ezt választva a Windows regisztrációs adatbázis txtfile/shell/print/command kulcsán leírt program indul, mely ismét csak a Notepad.exe lesz, de most az állomány nevén kívül a /p kapcsoló is megadásra kerül. Ez arra utasítja a Notepad programot, hogy ne csak megnyissa, hanem rögtön ki is nyomtassa a dokumentumot:

```
%SystemRoot%\system32\NOTEPAD.EXE /p %1
```

Saját kiterjesztés kezelése

Fenti ismeretek birtokában saját programunk által kezelt állományokhoz mi is készíthetünk saját Windows regisztrációs adatbázis bejegyzést.

Ehhez tegyük a következőt: legyen mondjuk egy programunk abc.exe néven, mely képes kezelni xy kiterjesztésű állományokat. Ekkor a Windows regisztrációs adatbázisba a HKEY_CLASSES_ROOT főkulcsa alá hozzunk létre egy .xy nevű bejegyzést és alapértelmezett értékénél adjuk meg mondjuk az xyfile értéket. Hozzunk létre szintén e főkulcs alá egy xyfile bejegyzést. Ez alá pedig vegyük fel a shell/open/command kulcsot. Ez utóbbi alapértelmezett értékénél adjuk meg az abc.exe nevű alkalmazásunkat, szükséges esetén teljes elérési úttal és ne feledkezzünk meg a %1 paraméterről sem, mely az aktuális állomány nevét, elérési útját tartalmazza majd, amit meg kell nyitnunk.

Az abc.exe programunkat persze fel kell készítenünk arra, hogy indítási paraméterként kap egy állomány nevet. Ebben az esetben ezt meg kell nyitnia és kezelnie kell. A Windows regisztrációs adatbázis módosítását követően, annak hatása azonnal él, amint egy xy állományon kattintunk.

A példaprogram

Térjünk most vissza fenti kis kitérő után eredeti feladatunkhoz, melynek célja, hogy minden a Windows által ismert állomány kiterjesztéshez kilistázzuk a hozzájuk tartozó kezelő programokat.

A példaprogram futás közben

E feladathoz egy ListBox-ot helyezünk a Form-ra, melyben soronként megjelenítjük a Windows által ismert állomány kiterjesztéseket.

Form OnCreate eseménye

A listát a program indulásakor készítjük el, a Form OnCreate eseményének futásakor.

```
procedure TForm1.FormCreate(
  Sender: TObject);
var
  reg: TRegistry;
  keys: TStringList;
  i: Integer;
  typename, displayname, server: string;
begin
```

Itt elsőként létrehozunk egy új példányt a TRegistry osztályból.

```
  reg := TRegistry.Create;
  try
```

Beállítjuk a főkulcsot a HKEY_CLASSES_ROOT értékre, így az ezen belül található adatokhoz férhetünk majd hozzá.

```
    reg.rootkey := HKEY_CLASSES_ROOT;
```

Megnyitjuk a főkulcsot.

```
    if reg.OpenKey('', False) then
    begin
```

Létrehozunk egy TStringList osztály példányát az adatok átmeneti tárolásához.

```
      keys := TStringList.Create;
      try
```

Ebbe a listába lekérdezzük a főkulcs alatt található össze alkulcs nevét.

```
        reg.GetKeyNames(keys);
```

Zárjuk a főkulcsot.

```
        reg.CloseKey;
```

Egy ciklust kezdeményezünk, mely végigmegy a lekérdezett lista összes elemén.

```
        for i := 0 to keys.Count - 1 do
        begin
```

Ha a kulcs nevének első karaktere pont, akkor tudhatjuk, hogy aktuálisan egy állomány kiterjesztés leírása következik.

```
          if keys[i][1] = '.' then
          begin
```

Ebben az esetben megnyitjuk az itt leírt kulcsot.

```
            if reg.OpenKey(keys[i],
              False) then
            begin
```

Beolvassuk annak alapértelmezett értékét. Az alapértelmezett érték olvasásához a ReadString függvénynek üres sztringet kell megadnunk.

```
              typename := reg.
                ReadString('');
```

A beolvasást követően zárjuk ezt a kulcsot.

```
              reg.CloseKey;
```

Amennyiben nem üres sztringet kaptunk, úgy elérhetjük az adott állomány kiterjesztés további leírását is a Windows regisztrációs adatbázisban.

```
if typename <> '' then
begin
```

Megnyitjuk tehát a megadott kulcsot.

```
if reg.OpenKey(typename,
  False) then
begin
```

Innen kiolvassuk a szöveges leírását az állomány kiterjesztésnek, melyet szintén az adott kulcs alapértelmezett értékénél találunk.

```
displayname := reg.
  ReadString('');
```

Zárjuk ezt a kulcsot is.

```
  reg.CloseKey;
end;
```

Most megnyitjuk a \shell\open\command\ alkulcsot, annak érdekében, hogy meghatározható legyen, hogy az adott állomány kiterjesztést mely program kezeli.

```
if reg.OpenKey(typename +
  '\shell\open\command',
  False) then
begin
```

Ezt az információt szintén e kulcs alapértelmezett értéke tárolja.

```
server := reg.
  ReadString('');
```

Az eddig gyűjtött információkat megjelenítjük a ListBox-ban.

```
ListBox1.Items.Add(keys[i]
                   + ' - ' + typename + '
-
                   '  + displayname + ' -
'
                    + server);
```

Végül zárjuk ezt a kulcsot is és folytatódhat tovább a for ciklusunk a következő állomány kiterjesztéssel.

```
  reg.CloseKey;
end;
```

```
          end;
        end;
      end;
    end;
  finally
    keys.Free;
  end;
end;
```

A ciklus végén megszüntetjük a létrehozott TRegistry objektum példányát.

```
  finally
    reg.Free
  end;
end;
```

Borbély István

Kapcsolódó webhelyek

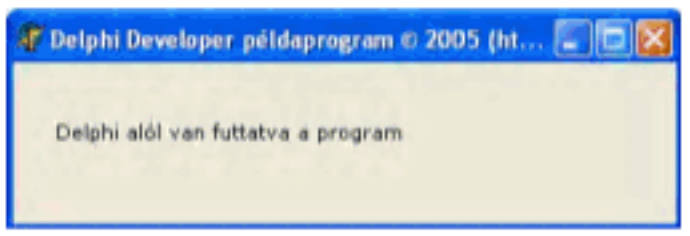
Windows regisztrációs adatbázis ↳ 005

Delphi alól fut-e a program?

Cikk sorszáma	Szoftverfeltételek
087	➢ Delphi
Forráskód	\Source\Delphi\

Arra a kérdésre, hogy egy program a Delphi fejlesztői környezete alól lett-e elindítva vagy sem, igen egyszerűen választ adhatunk. Mindössze ismernünk kell egy függvényt ehhez a Kernel32.dll-ben.

Delphi alól indítva



A program a Delphi fejlesztői környezete alól lett indítva

A kérdés megválaszolására az IsDebuggerPresent függvényt kell meghívnunk. Használata rendkívül egyszerű: a függvény paramétert nem igényel, visszatérési érték pedig igaz, ha fejlesztői környezet alól futtatjuk programunkat, míg hamis, ha nem.

Ezt a lekérdezési lehetőséget jól felhasználhatjuk olyan funkciók megvalósítására melyekre csak a fejlesztés folyamán van szükségünk, amikor alkalmazásunkat még a Delphi alól futtatva teszteljük, fejlesztjük.

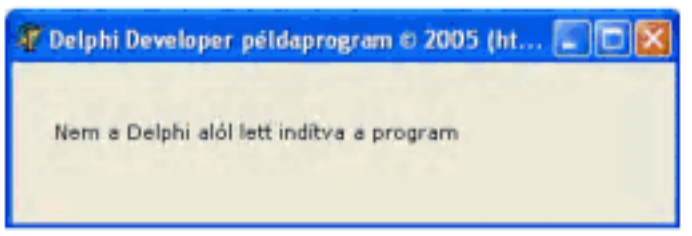
Az IsDebuggerPresent függvényt a Kernel32.dll-ben találjuk, így ennek megfelelően kell deklarálnunk:

```
function IsDebuggerPresent: boolean;
  stdcall; external kernel32
  name 'IsDebuggerPresent';
```

A függvény által szolgáltatott érték alapján elágazásokat tehetünk az alkalmazásunk futásának menetébe, így a mindenkori körülményekhez igazítva futtathatjuk kódrészleteinket.

```
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  if IsDebuggerPresent then begin
    Label1.Caption:='Delphi alól van
      futtatva a program';
  end else begin
    Label1.Caption:='Nem a Delphi alól lett
      indítva a program';
  end;
end;
```

Nem Delphi alól indítva



A program nem a Delphi fejlesztői környezete alól lett indítva

Indítsuk most el a példa alkalmazást a Windows Intézőjéből. Ekkor jól látható az üzenetből is, hogy a program érzékelte, hogy most nem a Delphi alól lett elindítva.

Bakonyi János

Kapcsolódó webhelyek

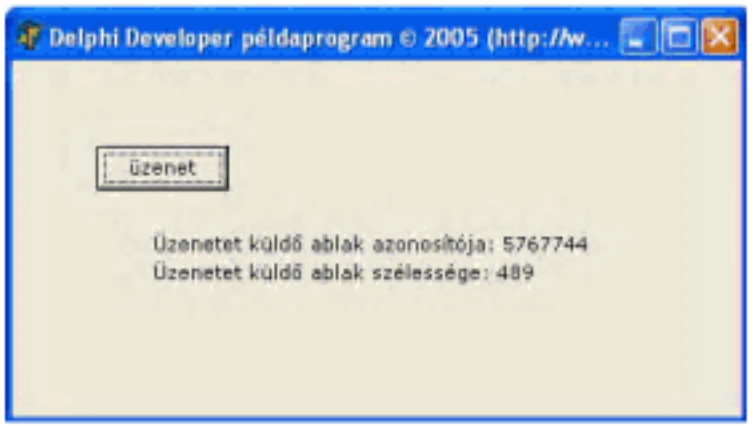
IsDebuggerPresent ↳ 004

Üzenet küldése alkalmazások között

Cikk sorszáma	Szoftverfeltételek
088	➢ Delphi
Forráskód	\Source\Communicate\

Számos eset fordul elő a programozói gyakorlatban, amikor két vagy több azonos, vagy esetleg különböző alkalmazásnak egymás között kell kapcsolatot teremteni és adatokat átadni, fogadni. Az alkalmazás közötti kapcsolattartásnak számos módszere ismert. Vegyük példánkban talán a legegyszerűbben megvalósítható esetet, az üzenetküldést. Üzenetek küldéséhez csupán egy egyszerű függvényhívásra van szükségünk. Az üzeneteket küldhetjük kimondottan egy alkalmazás számára, de küldhetjük akár az összes futó programnak is. Utóbbi esetben nyilvánvaló, hogy az üzenetet csak az az alkalmazás képes feldolgozni, melyet felkészítettek erre az üzenetre, vagyis nem származhat gond abból, hogy az üzenetünket olyan alkalmazás is megkapja, mely nem tud vele mit kezdeni, hiszen annak nem is feladata ez.

A mellékelt példaprogram kipróbálásához indítson el minimum három példányt az alkalmazásból, majd kattintson az "Üzenet" gombra. Ekkor a program üzenetet küld a többi példánynak, melyek megjelenítik azt a Form-ra helyezett Label-eken.



Üzenet fogadása és adatainak megjelenítése

A megvalósítás elve

Az üzenetküldés alapja a SendMessage függvény lesz. Ennek működtetéséhez szükségünk lesz egy olyan üzenet azonosítóra, melyet még egyetlen működő alkalmazás sem használ. Ez igen kritikus pontja a programunknak, hiszen, ha olyan azonosító számot használnánk, melyet más program is alkalmaz, akkor előfordulhatna olyan nem kívánt helyzet, hogy több program használja ugyanazt a számot üzenet küldésre. Ebből pedig igen sok nem kívánt helyzet és hibás működés származna. Amennyiben adott az egyedi azonosító szám, akkor már csak egy egyedi üzenet struktúrát kell létrehoznunk, mely az adott feladatainknak megfelel. Ha ez is adott, akkor máris használhatjuk a SendMessage függvényt. Fel kell még készítenünk programunkat arra is, hogy ne csak küldje, hanem fogadja is az üzeneteket. Az üzenetek érkezésekor ki kell válogatnunk, hogy melyik az, ami a mi alkalmazásunknak szól.

Üzenet struktúra létrehozása

Kezdjük a megvalósítást az új üzenet struktúra létrehozásával. A példaprogramban a többi al-

kalmazás számára küldendő üzenetünkben a Form aktuális szélességét kívánjuk tovább adni. Ennek megfelelően az üzenet struktúra tartalmaz egy Width nevű mezőt, melyben ezt az értéket tárolhatjuk. Ezen felül megtaláljuk még a szokásos elemeket: Msg - az üzenet azonosítója, Handle - a küldő ablak azonosítója, Result - a feldolgozás eredménye.

```
type
  TWMMYMessage = record
    Msg: Cardinal;
    Handle: HWND;
    Width: Longint;
    Result: Longint;
  end;
```

Egyedi üzenet azonosító

Az egyedi üzenet azonosítószám beszerzéséhez a RegisterWindowMessage függvényen keresztül vezet az út. E függvénynek egy tetszőleges tartalmú szöveges paramétert kell megadni. Ha a függvény talál már ilyen szöveggel regisztrált üzenetet, akkor annak azonosítószámát adja vissza. Ha még nincs ilyen, akkor létrehoz egy újat, melyet még egyetlen alkalmazás sem használ és azt adja eredményül és egyúttal tárolja ehhez a szöveghez társítva a kiosztott számot. Így ha alkalmazásunkból újabb példányt indítunk el és az is ugyanezzel a szöveges paraméterrel hívja a RegisterWindowMessage függvényt, akkor megkapja a már létrehozott egyedi azonosítót, így biztosítva van, hogy alkalmazásunk különböző példányai képesek lesznek kommunikálni egymással, hiszen mindegyik ismeri majd a programunk számára létrehozott egyedi számot.

Ezt a számot a programunk futásának kezdetén regisztráljuk és értékét a WM_MESSAGE globális változóba tároljuk.

```
var
  WM_MESSAGE: DWORD;
initialization
  WM_MESSAGE := RegisterWindowMessage(
    'Delphi Developer message');
end.
```

Üzenet küldése

Ezt követően az üzenet küldése igen egyszerű feladat a Button1-re történő kattintáskor: meghívjuk a SendMessage függvényt, melynek első paraméterként ablak azonosító helyett a HWND_BROADCAST konstanst adjuk meg. Ennek az lesz a hatása, hogy az elküldött üzenetet nem csupán egy megadott ablak, hanem minden program megkapja. Így az üzenet eljut minden alkalmazáshoz, közte saját programjainkhoz is.

A SendMessage függvény második paraméterében az üzenet azonosítóját adjuk meg, melyet a WM_MESSAGE változónk tárol. Ezt követi az ablakunk azonosítója, mint az üzenet "feladója", valamint a Form1 aktuális szélessége. Ez utóbbi csupán a teszt kedvéért. Próbáljuk ki, hogy változtatjuk a Form1 szélességét és újra meg újra elküldjük az üzenetet. Ekkor megfigyelhető lesz, hogy a programunk további példányainak Formján megjelenik a küldő Form szélessége.

```
procedure TForm1.Button1Click(
  Sender: TObject);
begin
  SendMessage(HWND_BROADCAST, WM_MESSAGE,
    Handle, Width);
end;
```

TForm1 eljárásai

Szükségünk lesz a beérkező üzenetek kezeléséhez két eljárás létrehozására is. Az egyik a DefaultHandler, mely megtalálható a TForm-ban, mint ős osztályban, így ezt most felülírva kell deklarálnunk. Ez az eljárás kerül meghívásra minden olyan esetben, amikor a Form1-hez bármilyen üzenet érkezik. Ez lesz majd az a pillanat, amikor a sokféle üzenet közül ki kell választanunk a WM_MESSAGE változóban tárolt azonosítószámmal takart üzenetet. Amikor bekövetkezik ennek az üzenetnek a megérkezése, akkor hívjuk meg a másodikként létrehozott eljárásunkat, melyet WMMYMessage néven hozunk létre. Ennek feladata lesz a beérkezett üzenet feldolgozása.

```
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender:
TObject);
  private
    procedure DefaultHandler(var
      Message); override;
    procedure WMMYMessage(var
      Msg: TWMMYMessage);
  public
    { Public declarations }
  end;
```

DefaultHandler eljárás

Ha tehát a DefaultHandler eljárás futására kerül a sor, akkor tudhatjuk, hogy ablakunk új üzenetet kapott. Elsőként azt kell ellenőriznünk, hogy az üzenet egyedi azonosítószáma egyezik-e a WM_MESSAGE változóban tárolt értékkel, vagy sem.

```
procedure TForm1.DefaultHandler(
  var Message);
var
  ee: TWMMYMessage;
begin
  with TMessage(Message) do
  begin
    if (Msg = WM_MESSAGE) then
    begin
```

Ha egyezik, akkor feltöltünk egy TWMMYMessage típusú változót értékekkel, melyet a paraméterként kapott Message-ből olvashatunk ki.

```
      ee.Msg    := Msg;
      ee.Handle := wParam;
      ee.Width   := lParam;
```

Következő lépésben szűrést végzünk: mivel a HWND_BROADCAST miatt minden program - közte az éppen futó programunk is - megkapja az üzenetet, így szeretnénk kiszűrni azt, hogy a program saját magának is üzenjen. Ezért ellenőrizzük az üzenetben lévő ablak azonosító számot. Ha ez egyezik a Form1 azonosítójával, akkor nem dolgozzuk fel tovább a kapott üzenetet. Ha különböző ez a szám, akkor tudhatjuk, hogy az üzenet egy másik alkalmazástól származik és ez esetben meghívjuk a WMMYMessage eljárásunkat, mely feldolgozza a kapott üzenetet.

```
      if ee.Handle <> Handle then
        WMMYMessage(ee);
    end
```

Bármilyen egyéb üzenet érkezik alkalmazásunkhoz, azt tovább adjuk a felülírt DefaultHandler eljárásnak feldolgozásra.

```
    else
      inherited DefaultHandler(Message);
  end;
end;
```

WMMYMessage eljárás

A WMMYMessage eljárás futására tehát már csak akkor kerülhet sor, ha a kapott üzenetet ténylegesen fel kell dolgoznia alkalmazásunknak.

Ebben az esetben igény szerinti funkciót valósíthatunk meg. A mellékelt példaprogramban az egyszerűség kedvéért most csupán megjelenítjük a Form-on két Label segítségével az üzenetet küldő ablak azonosítóját, valamint annak szélességét.

```
procedure TForm1.WMMYMessage(var
  Msg: TWMMYMessage);
begin
  Label1.Caption := 'Üzenetet küldő ablak
    azonosítója: ' + IntToStr(Msg.Handle);
  Label2.Caption := 'Üzenetet küldő ablak
    szélessége: '+ IntToStr(Msg.Width);
end;
```

Továbbfejlesztés

Fentiekben minden alkalmazás elküldi az ablak azonosítóját a többi program számára. Ha ezt tároljuk, akkor minden programunk "ismeri" majd a másikat és ezt kihasználva célzott üzeneteket is küldhetnek egymásnak. Így nem szükséges a HWND_BROADCAST típusú üzenet küldés, sőt az üzenetet küldhetjük csupán egy-egy programunknak helyzettől függően.

Ambrás Péter

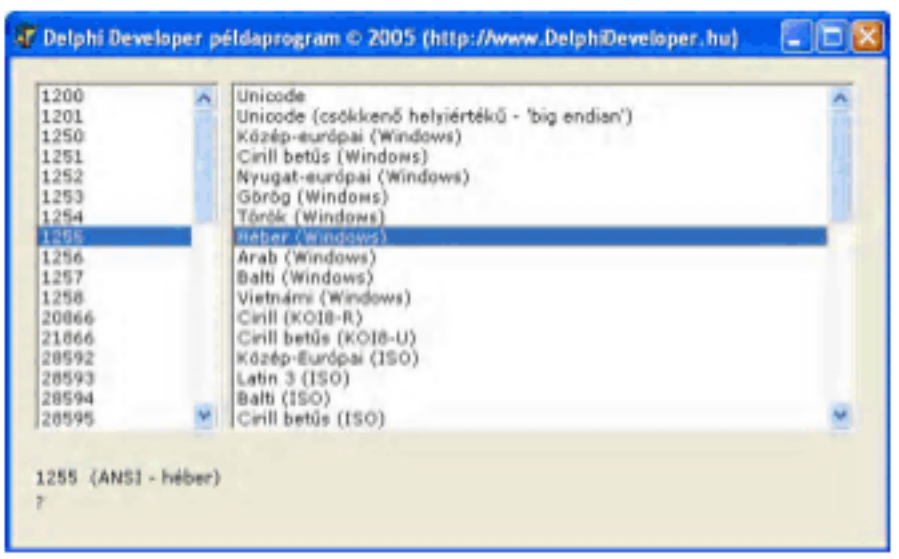
Kapcsolódó webhelyek

SendMessage ↳ 003

Kódlap információk

Cikk sorszáma	Szoftverfeltételek
089	➢ Delphi
Forráskód	\Source\CodePage\

E példaprogram feladata, hogy segítségével felderíthessük az adott számítógépre telepített, illetve az ott elérhető kódlapok információit.



Kódlap információk

Az elmélet

A feladat megvalósításában a GetCPInfoEx függvény lesz segítségünkre. E függvényt a Kernel32.DLL-ben találjuk és a Windows 98 óta létezik. Deklarációja az alábbi:

```
BOOL GetCPInfoEx (
  UINT CodePage,
  DWORD dwFlags,
  LPCPINFOEX lpCPInfoEx
);
```

A függvény paraméterei

- CodePage

Itt kell megadnunk annak a kódlapnak az azonosító számát, melynek információira kíváncsiak vagyunk. A kódszám helyett használhatjuk azonban az alábbi konstansok egyikét is:

CP_ACP - a rendszer által alapértelmezettként használt ANSI kódlap

CP_MACCP - Windows NT/2000/XP/2003 esetén az alapértelmezett Macintosh kódlap

CP_OEMCP - a rendszer által alapértelmezettként használt OEM kódlap

CP_THREAD_ACP - Windows 2000/XP/2003 esetén az aktuális folyamat által használt ANSI kódlap

- dwFlags

Fenntartott paraméter. Értéke mindig nulla.

- lpCPInfoEx

Egy CPINFOEX struktúra, melyben a lekérdezett adatokat kapjuk meg a kiválasztott kódlapról.

Visszatérési érték

Ha az adatlekérdezés művelete sikeres, akkor visszatérési értékként igazat, különben hamisat kapunk.

Sikertelen végrehajtás esetén a GetLastError függvény hívásával további információt tudhatunk meg a hiba okáról.

Abban az esetben, ha a paraméterként megadott kódlap nem elérhető, illetve nincs telepítve az adott számítógépre, akkor a GetLastError az ERROR_INVALID_PARAMETER értéket adja vissza.

CPINFOEX struktúra

A CPINFOEX struktúra tárolja az adott kódlap információit, melyet a GetCPInfoEx függvény kérdez le.

A struktúra deklarációja:

```
typedef struct _cpinfoex {
  UINT    MaxCharSize;
  BYTE    DefaultChar[MAX_DEFAULTCHAR];
  BYTE    LeadByte[MAX_LEADBYTES];
  WCHAR   UnicodeDefaultChar;
  UINT    CodePage;
  TCHAR   CodePageName[MAX_PATH];
} CPINFOEX, *LPCPINFOEX;
```

- MaxCharSize

A kódlap egy-egy karaktere ennyi bájton kerül tárolásra.

- DefaultChar

Alapértelmezett karakter, melyet a WideCharToMultiByte függvény használ fel.

- UnicodeDefaultChar

Alapértelmezett karakter, melyet a MultiByteToWideChar függvény használ fel.

- CodePage

A kódlap azonosítója.

- CodePageName

A kódlap teljes megnevezése az aktuális nemzetközi beállításoknál érvényben lévő nyelven.

A példaprogram megvalósítása

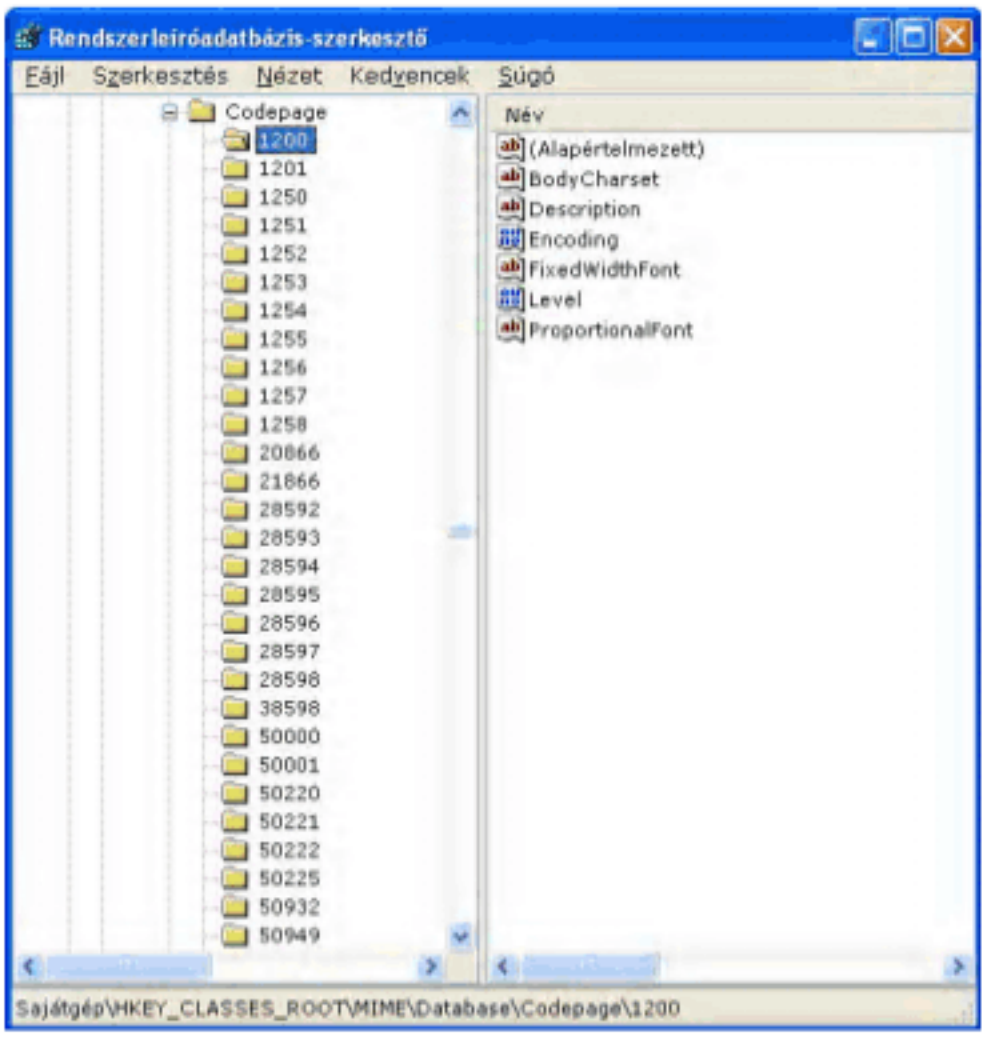
A GetCPInfoEx függvény felhasználásához szükségünk lesz annak deklarálására, mivel a Delphi unit-jaiban ezt nem találjuk meg. Ez a helyzet a függvény által használt CPINFOEX struktúrával is, így ennek leírásáról is nekünk kell gondoskodnunk:

```
TCPInfoEx = record
    MaxCharSize: UINT;
    DefaultChar: array[0..MAX_DEFAULTCHAR -
      1] of Byte;
    LeadByte: array[0..MAX_LEADBYTES - 1]
      of Byte;
    UnicodeDefaultChar: WCHAR;
    CodePage: UINT;
    CodePageName: array[0..MAX_PATH]
      of char;
  end;
```

A GetCPInfoEx-t külső függvényként deklaráljuk, mely a Kernel32.DLL-ben található.

```
  function GetCPInfoEx(CodePage: UINT;
    dwFlags: DWORD; var lpCPInfoEx:
    TCPInfoEx): BOOL; stdcall;
  function GetCPInfoEx; external kernel32
    name 'GetCPInfoExA';
```

Form OnCreate esemény



Windows regisztrációs adatbázisban tárolt adatok

A példaprogram Form-jára helyezünk egy ListBox komponenst. Ennek tartalmát feltöltjük az adott számítógépen elérhető kódlapok azonosító számaival. Hogy az adott környezetben mely kódlapok állnak rendelkezésre, azt a Windows regisztrációs adatbázis segítségével határozhatjuk meg. A HKEY_CLASSES_ROOT főkulcson belül, a 'MIME\Database\Codepage' címen találunk egy listát az elérhető kódlapokról.

Így nincs más tennivalónk, mint a szükséges adatokat kiolvasni a program indulásakor és megjeleníteni azokat.

```
procedure TForm1.FormCreate(
  Sender: TObject);
const
  REGKEY = 'MIME\Database\Codepage';
  DESCRIPTION = 'Description';
begin
  inherited;
```

Elsőként megnyitjuk a Windows regisztrációs adatbázis megadott kulcsát.

```
  reg:=TRegistry.Create;
  with reg do begin
    RootKey:=HKEY_CLASSES_ROOT;
    if OpenKey(REGKEY, false) then begin
```

Majd beolvassuk az adott kulcson található neveket egy TStringList listába.

```
      l:=TStringList.Create;
      GetKeyNames(l);
```

Zárjuk ezt a Windows regisztrációs adatbázis kulcsot.

```
      CloseKey;
```

Ezek után egy while ciklust kezdeményezünk, mely végigmegy az imént kiolvasott lista minden elemén.

```
      i:=0;
      while i<l.Count do begin
```

A lista minden eleme egy-egy újabb kulcsot takar, melyet olvasásra megnyitunk.

```
        if OpenKey(REGKEY+'\'+l[i], false)
          then begin
```

Ellenőrizzük, hogy a kódlap leírása megtalálható-e. Ha igen, akkor kiolvassuk az s változóba.

```
          if ValueExists(DESCRIPTION)
            then begin
            s:=ReadString(DESCRIPTION);
          end else begin
            s:='?';
          end;
```

Végezetül megjelenítjük a Form-on a kiolvasott adatokat és zárjuk a megnyitott Windows regisztrációs adatbázis kulcsot.

```
          ListBox1.Items.Append(l[i]);
          ListBox2.Items.Append(s);
          CloseKey;
        end;
        inc(i);
      end;
      l.Free;
    end;
    Free;
  end;
end;
```

ListBox1 OnClick esemény

Amikor a program futása közben a felhasználó a ListBox1 valamely elemén kattint, akkor hívjuk meg a GetCPInfoEx függvényünket. Ennek feladata lesz, hogy a ListBox1-ben kiválasztott kódlapról információt gyűjtsön és jelenítsen meg.

A függvény első paraméterének a kódlap számát adjuk meg, mely most a ListBox1 aktuálisan kijelölt eleme lesz. Ezt persze számmá kell konvertálnunk. A második paraméter nem használatos, így ide nullát írunk. A harmadik paraméter lesz a TCPInfoEx struktúrából létrehozott változó, melybe a lekérdezett adatokat írja a függvény.

```
procedure TForm1.ListBox1Click(
  Sender: TObject);
var
  cp: TCPInfoEx;
begin
```

Ha a GetCPInfoEx függvény igaz értékkel tér vissza, akkor az adott kódlap információit sikerült lekérdezni és azok megtalálhatók a TCPInfoEx struktúrában.

```
  if GetCPInfoEx(StrToInt(ListBox1.
    Items[ListBox1.ItemIndex]), 0, cp)
    then begin
```

Most már tetszés szerint felhasználhatjuk a struktúra adatait. Ezekből mi csupán két értéket jelenítünk most meg a Form-on, egy-egy Label komponensen.

```
    Label1.Caption:=cp.CodePageName;
    Label2.Caption:=cp.UnicodeDefaultChar;
  end;
end;
```

Alabárdos András

Kapcsolódó webhelyek

GetCPInfoEx ↳002

Ablak animáció

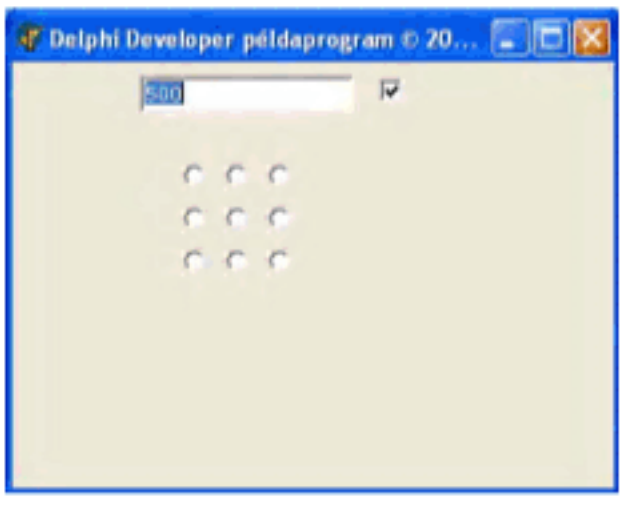
Cikk sorszáma	Szoftverfeltételek
090	➢ Delphi
Forráskód	\Source\Animate\

Programjaink ablakainak megjelenítésére általában nincs sok gondunk. Meghívjuk a megfelelő eljárást és az ablak látható lesz a képernyőn.

Ha azonban a megjelenítést egy kis animációval ötvözzük, akkor igen egyszerűen látványossá tehetjük ablakaink megjelenítését és elrejtését. Már a Windows 98 óta adott a lehetőség arra, hogy az ablakokat animálva jelenítsük meg, illetve rejtsük el.

E példában annak járunk utána, hogy az animációt megvalósító függvényt miként használhatjuk fel saját programunkban.

A mellékelt példaprogram használatához minimálisan Windows 98, illetve Windows 2000, vagy ezeknél magasabb verziószámú operációs rendszer szükséges.



A működő példaprogram

AnimateWindow függvény

Az ablak animációt megvalósító függvény az AnimateWindow lesz, mely a User32.DLL-ben található. Ennek segítségével nem csak egyszerűen megjeleníthetünk, illetve elrejthetünk egy-egy ablakot, hanem ezt animálva tehetjük meg. Az animáció végrehajtásához négy különböző effektet használhatunk fel:

- roll
- slide
- collapse illetve expand
- alpha-blended fade

AnimateWindow szintaxisa

```
BOOL AnimateWindow(
    HWND hwnd,
    DWORD dwTime,
    DWORD dwFlags
);
```

Paraméterek

- hwnd

Annak az ablaknak a leírója, melyet animálva kívánunk megjeleníteni, illetve elrejteni. Fontos, hogy csak olyan ablakot adhatunk meg, melynek programunk a tulajdonosa.

- dwTime

Az animáció időtartama ezredmásodpercben mérve.

- dwFlags

Az animáció típusát adhatjuk meg a dwFlags paraméterben. Ennek értéke az alábbi konstansok logikai vagy művelet eredményeként határozható meg.

dwFlags paraméter konstansai

- AW_SLIDE

Slide típusú animáció végrehajtása. Ez az érték nem használható az AW_CENTER konstanssal együtt.

- AW_ACTIVATE

Az animáció az ablak megjelenítésére irányul. Ez az érték nem használható az AW_HIDE konstanssal együtt.

- AW_BLEND

Alpha-blended fade típusú animáció végrehajtása. Ez a lehetőség csak ún. top-level ablakok esetén alkalmazható.

- AW_HIDE

Az animáció az ablak elrejtésére irányul.

- AW_CENTER

Az animáció az ablak széleitől annak közepe felé irányul, ha az AW_HIDE konstans is meg van adva. Ha az AW_HIDE konstans nincs megadva, akkor az animáció az ablak közepétől indulva annak szélei felé irányul.

- AW_HOR_POSITIVE

Az animáció az ablak bal szélétől indulva annak jobb széle felé irányul. Ez az érték a Roll és Slide típusú animációk esetén használható. Ez az érték figyelmen kívül lesz hagyva az AW_CENTER vagy az AW_BLEND konstansok használata esetén.

- AW_HOR_NEGATIVE

Az animáció az ablak jobb szélétől indulva annak bal széle felé irányul. Ez az érték a Roll és Slide típusú animációk esetén használható. Ez az érték figyelmen kívül lesz hagyva az AW_CENTER vagy az AW_BLEND konstansok használata esetén.

- AW_VER_POSITIVE

Az animáció az ablak tetejéről indulva annak alja felé irányul. Ez az érték a Roll és Slide típusú animációk esetén használható. Ez az érték figyelmen kívül lesz hagyva az AW_CENTER vagy az AW_BLEND konstansok használata esetén.

- AW_VER_NEGATIVE

Az animáció az ablak aljáról indulva annak teteje felé irányul. Ez az érték a Roll és Slide típusú animációk esetén használható. Ez az érték figyelmen kívül lesz hagyva az AW_CENTER vagy az AW_BLEND konstansok használata esetén.

Visszatérési érték

A függvény igaz értékkel tér vissza, ha a kért művelet sikeresen lezajlott.

Sikertelen futás esetén a visszatérési értéke hamis. Ennek oka az alábbi helyzetek lehetnek:

- ha az ablak használ régiót. (Windows XP esetén ez nem lesz probléma!)
- ha az ablak már látható és azt az AW_ACTIVATE konstanssal meg akarjuk jeleníteni.
- ha az ablak nem látható és azt az AW_HIDE konstanssal el akarjuk rejteni.
- Slide vagy Roll típusú animáció esetén nem adunk meg irányt.
- gyerek ablakra kívánjuk alkalmazni az AW_BLEND konstanst.
- a programunk nem tulajdonosa annak az ablaknak, melyre az animációt végre kívánjuk hajtani.

Hiba esetén további információt a GetLastError függvény hívásával kaphatunk.

Megjegyzések

Ha nem kívánunk animációt használni, akkor az ablakok megjelenítésére, elrejtésére használjuk inkább a ShowWindow függvényt az AnimateWindow helyett.

A Slide vagy Roll animáció esetén legalább az egyik konstanst adjuk meg a következők közül: AW_HOR_POSITIVE, AW_HOR_NEGATIVE, AW_VER_POSITIVE, AW_VER_NEGATIVE.

Ha kombináljuk az AW_HOR_POSITIVE vagy AW_HOR_NEGATIVE konstansokat az AW_VER_POSITIVE vagy AW_VER_NEGATIVE konstansokkal, akkor elérhetjük, hogy az animáció iránya átlós legyen.

A gyakorlat

A függvény gyakorlati alkalmazása előtt két tennivalónk akad: az egyik, hogy deklaráljuk azokat a konstansokat, melyeket a függvényben használhatunk. Mivel a Delphi unit-jai nem tartalmazzák ezeket az értékeket, így magunknak kell gondoskodni róla:

```
const
  AW_HOR_POSITIVE = $00000001;
  AW_HOR_NEGATIVE = $00000002;
  AW_VER_POSITIVE = $00000004;
  AW_VER_NEGATIVE = $00000008;
  AW_CENTER = $00000010;
  AW_HIDE = $00010000;
  AW_ACTIVATE = $00020000;
  AW_SLIDE = $00040000;
  AW_BLEND = $00080000;
```

A másik lépés maga az AnimateWindow függvény deklarációja lesz. Ezt külső függvényként adjuk meg, mely a User32.DLL-ben található.

```
function AnimateWindow(hWnd: HWND; dwTime:
  DWORD; dwFlags: DWORD): BOOL; stdcall;
  external 'user32.dll' name
'AnimateWindow';
```

Fenti előkészületek után máris használható lesz a Delphi-s programjainkban az AnimateWindow függvény.

Példaprogram

A mellékelt példaprogramban elhelyezünk a Form-on 9 + 1db RadioButton-t. Ezek segítségével választhatjuk ki az animáció irányát, illetve a középen álló +1 db RadioButton segítségével végezhetünk az ablak közepe felé irányuló animációt. Egy Edit komponensbe írhatjuk be az animáció időtartamát ezredmásodpercben, míg egy CheckBox segítségével jelölhetjük ki a Slide effektet.

OnCreate esemény

Az OnCreate eseménynél a RadioButton komponensek Tag tulajdonságaiba tároljuk el a végrehajtandó animáció irányát. Így bármely RadioButton-ra történő kattintás esetén egyszerűen meghatározhatjuk, hogy milyen konstans értéket kell megadni az AnimateWindow függvény harmadik paraméterébe.

```
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  RadioButton1.
    Tag:=AW_HOR_NEGATIVE+AW_VER_NEGATIVE;
  RadioButton2.Tag:=AW_VER_NEGATIVE;
  RadioButton3.
    Tag:=AW_VER_NEGATIVE+AW_HOR_POSITIVE;
  RadioButton4.Tag:=AW_HOR_NEGATIVE;
  RadioButton5.Tag:=AW_CENTER;
  RadioButton6.Tag:=AW_HOR_POSITIVE;
  RadioButton7.
    Tag:=AW_HOR_NEGATIVE+AW_VER_POSITIVE;
  RadioButton8.Tag:=AW_VER_POSITIVE;
  RadioButton9.
    Tag:=AW_VER_POSITIVE+AW_HOR_POSITIVE;
end;
```

RadioButton Click esemény

A Form-on lévő összes RadioButton komponens Click eseményeihez ugyanazt az esemény kezelő eljárást kötjük hozzá.

```
procedure TForm1.RadioButton1Click(
  Sender: TObject);
const
  s: array[boolean] of integer=
    (0, AW_SLIDE);
begin
```

Két animációt végzünk: az első segítségével elrejtjük a Form-ot. Ennek érdekében a harmadik paraméterben megadjuk az AW_HIDE konstanst, melyhez hozzáadjuk a RadioButton komponens Tag tulajdonságában tárolt értékét. Az így kiszámított értékhez még egy szám kerül hozzáadásra attól függően, hogy a CheckBox kijelölt-e vagy sem. Ha kijelölt, akkor az AW_SLIDE értékét is hozzáadjuk.

```
  AnimateWindow(Handle, StrToInt(Edit1.
    Text), AW_HIDE+(Sender as
    TRadioButton).
    Tag+s[CheckBox1.Checked]);
```

A második AnimateWindow hívásnál a Form megjelenítését végezzük el, az AW_ACTIVATE konstans megadásával. A további paraméterek egyeznek a fenti függvény hívásánál megadott értékekkel.

```
  AnimateWindow(Handle, StrToInt(Edit1.
    Text), AW_ACTIVATE+(Sender as
    TRadioButton).
    Tag+s[CheckBox1.Checked]);
end;
```

CloseQuery esemény

A Form bezárásakor, annak elrejtését animálva végezzük. Így amikor a felhasználó bezárja programunk ablakát, akkor az egy előre kiválasztott animáció végrehajtásával tűnik el a képernyőről.

```
procedure TForm1.FormCloseQuery(Sender:
  TObject; var CanClose: Boolean);
begin
  AnimateWindow(Handle,
    300, AW_HIDE+AW_VER_NEGATIVE);
end;
```

Button1 gomb Click esemény

Nem foglalkoztunk eddig még az alpha-blended fade típusú animációval. Ezt a Form-ra helyezett Button1 gombra történő kattintáskor próbálhatjuk ki. Ez az animáció fokozatosan átlátszóvá teszi az ablakot, amíg az teljesen el nem tűnik.

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  AnimateWindow(Handle,
StrToInt(Edit1.Text), AW_HIDE+AW_BLEND);
  AnimateWindow(Handle,
StrToInt(Edit1.Text),
AW_ACTIVATE+AW_BLEND);
end;
```

Ábrahám Lajos

Kapcsolódó webhelyek

AnimateWindow ↳ 001

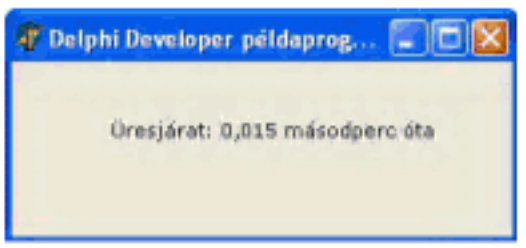
Üresjárati idő figyelése

Cikk sorszáma	Szoftverfeltételek
091	➢ Delphi
Forráskód	\Source\Idle\

Számítógépünk processzora bizony ideje nagy részében csak "unatkozik". A processzor számára még két billentyű leütés között is rengeteg idő telik el, hiszen a mai számítógépek feldolgozási sebessége már igen nagy. Sokszor előfordul azonban az is, hogy percekre, órákra magára hagyjuk a gépet és azon nem fut semmi olyan alkalmazás, mely terhelné a processzort. Ezt a kihasználatlan időt nevezzük "üresjárati" időnek. Egy ügyesen megírt alkalmazás lehetővé teszi ezt az üresjárati idő kihasználását. Erre akkor lehet szükségünk, ha olyan műveleteket futtatnánk, melyek akár órákig, napokig eltarthat. Ilyenkor nyilván nem szeretnénk, ha gépünk teljesen használhatatlan lenne az adott alkalmazás miatt. Ezért célszerű ezt az alkalmazást úgy megírnunk, hogy az a háttérben futva, csak akkor végezze a dolgát, ha amúgy a számítógép üresjárati időben van. Ezért számítógépünket továbbra is úgy használhatjuk, mint azelőtt és észre sem vesszük, hogy a háttérben futó alkalmazás mikor is teszi a dolgát, hiszen az mindig csak akkor jut szerephez, ha más alkalmazás nem terheli a számítógépet.

E példában annak járunk utána, hogy miként figyelhetjük programból az üresjárati idő bekövetkeztét, illetve mikor értesülhetünk arról, ha a felhasználó ismét használja a gépet (billentyűt, egeret kezel).

Példaprogram használata



Üresjárati idő mérése

A mellékelt példaprogramot futtatva, az egy Label komponens területén megjeleníti az aktuális üresjárati időt. Amint megmozdítjuk az egeret, vagy lenyomunk bármilyen billentyűt, akkor ezt érzékeli a programunk és a számlálót lenullázva elölről kezdi a mérést.

Működés elmélete

A feladat megvalósításához ismernünk kell néhány Windows API függvényt közelebbről. A legfontosabb a GetLastInputInfo függvény lesz, melynek segítségével lekérdezhetjük, hogy a felhasználó mióta nem használta sem a billentyűzetet, sem az egeret, vagy egyéb beviteli eszközt. Ez alapján meghatározható, hogy a számítógépünk mióta tétlenkedik. Ezt a függvényt a Windows 2000 verziója óta vehetjük igénybe.

A függvény deklarációja az alábbi:

```
BOOL GetLastInputInfo(PLASTINPUTINFO plii);
```

A függvény egyetlen paramétert tartalmaz:

- plii

Ez egy LASTINPUTINFO struktúra lesz, melyben a lekérdezett eredmény található. E struktúra felépítése igen egyszerű, mindössze két mezőt tartalmaz: a cbSize-nél kell megadnunk a struktú-

ra méretét, míg a dwTime értéke adja meg számunkra azt az időt, amióta a felhasználó nem használta a gépet. Bár a dwTime-ban megadott időpont ezredmásodpercben van, azt mégsem tudjuk közvetlenül értelmezni. Ehhez még szükségünk lesz a GetTickCount függvény által adott értékre is.

Visszatérési érték

A függvény visszatérési értéke igaz lesz, ha a lekérdezés sikeres volt, különben hamis.

GetTickCount függvény

A GetTickCount függvény egy olyan számláló aktuális értékét adja vissza, mely akkor indul, amikor az operációs rendszer, és mindaddig fut, amíg a számítógépet le nem állítjuk. E számláló értéke ezredmásodpercenként eggyel nő. Mivel az értéket DWORD típusban tárolja, így 49.7 napig képes folyamatosan számlálni, majd a nullától kezdi újra. Persze, ha ez idő alatt újraindítjuk az operációs rendszert, akkor a számláló is nullázódik. A függvény deklarációja:

```
DWORD GetTickCount(void);
```

Visszatérési érték

A függvény ezredmásodpercben adja vissza a rendszer indítása óta eltelt időt. Ezt könnyedén átszámíthatjuk nap, óra, perc és másodperc formára az alábbiak szerint:

```
   tickCount := GetTickCount();
   seconds := tickCount / 1000 % 60;
   minutes := tickCount / 1000 / 60 % 60;
   hours := tickCount / 1000 / 60 / 60 %
24;
   days := tickCount / 1000 / 60 / 60 / 24
% 7;
```

Megjegyzés

Ha a 49.7 nap kevés lenne feladataink megvalósításához, akkor a Windows regisztrációs adatbázisban találunk egy 8 bájton tárolt System Up Time nevű számlálót a HKEY_PERFORMANCE_DATA főkulcsnál.

Példaprogram megvalósítása

A példaprogramban egy Timer komponenst helyezünk a Form-ra. Ennek Interval tulajdonságát 100-ra állítjuk, így tized másodpercenként lefut az OnTimer eseménye.

Ezt használjuk fel a Label1 értékének aktualizálására. A pillanatnyi üresjárati időt lekérdezzük a LastInput függvény hívásával, mely másodpercben mérve adja vissza ezt az értéket számunkra.

```
procedure TForm1.Timer1Timer(
  Sender: TObject);
begin
  Label1.Caption := 'Üresjárat: ' +
    FloatToStr(LastInput) + ' másodperc
óta';
end;
```

LastInput függvény

A függvényben meg kell hívnunk a GetLastInputInfo függvényt. Ehhez szükségünk lesz egy TLastInputInfo struktúrára, melyben a cbSize mezőnek értékül adjuk a struktúra méretét.

```
function TForm1.LastInput: double;
var
  LInput: TLastInputInfo;
begin
  LInput.cbSize := SizeOf(TLastInputInfo);
  GetLastInputInfo(LInput);
```

Az eredmény másodpercben történő kiszámításához a GetTickCount függvény által visszaadott pillanatnyi értékből a TLastInputInfo struktúra dwTime mezőjében tárolt értéket kell kivonnunk. Ekkor az eredmény már rendelkezésünkre áll, de még ezredmásodpercben mérve. Ezt könnyedén átalakíthatjuk másodperc alapú értékké, csupán osztanunk kell ezerrel.

```
  Result := (GetTickCount - LInput.dwTime)
    / 1000;
end;
```

Üresjárati idő felhasználása

Fentiek alapján most már figyelni és mérni tudjuk az üresjárati időt. Ezt kihasználni háttérben futó műveletek végzésére úgy tudjuk, ha alkalmazzuk az Application osztály OnIdle nevű eseményét. Ez akkor kerül aktivizálásra, amikor a gép üresjárati időben van, így a processzort kihasználhatjuk háttérben futó folyamatok elvégzéséhez.

Farkas Sámuel

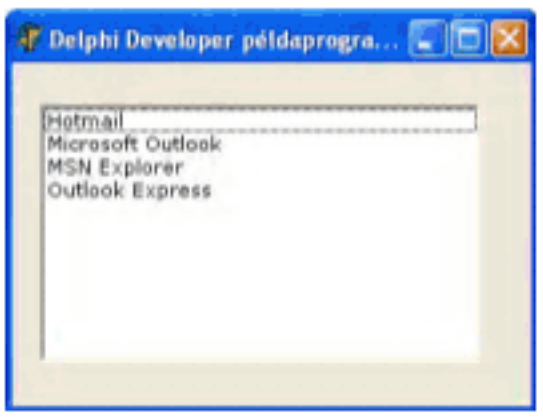
Kapcsolódó webhelyek

GetLastInputInfo ↳ 008

Milyen levelező kliensek vannak telepítve?

Cikk sorszáma	Szoftverfeltételek
092	➢ Delphi
Forráskód	\Source\MailClients\

Ebben a példában annak járunk utána, hogy egy számítógépen miként állapítható meg saját programból az, hogy milyen levelező kliensek vannak telepítve.



Telepített levelező kliensek listája

Form OnCreate esemény

A programunk indulásakor, a Form OnCreate eseményénél meghívjuk a GetInstalledMailClients függvényünket, melynek paraméterként megadjuk a ListBox1 Items tulajdonságát. Az összegyűjtött levelező kliensek megnevezéseit majd itt tárolja el ez a függvény.

```
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  GetInstalledMailClients(ListBox1.Items);
end;
```

GetInstalledMailClients függvény

A kérdéses adatokat a Windows regisztrációs adatbázisában találjuk meg, így onnan kiolvasva kell a paraméterként megadott TStrings-ből származó objektumba tárolnunk.

```
function GetInstalledMailClients(AList:
  TStrings): Boolean;
```

A Windows regisztrációs adatbázis címeit konstansokba írjuk le.

```
const
  RegClientsRoot = '\SOFTWARE\Clients';
  RegClientsMail = '\Mail';
  RegClientsOpenCmd =
'\shell\open\command';
var
  reg: TRegistry;
begin
  Result := True;
  try
```

Töröljük a paraméterként kapott lista esetleges elemeit.

```
    AList.Clear;
```

Megnyitjuk a Windows regisztrációs adatbázist.

```
    reg := TRegistry.Create;
    try
      with reg do
      begin
```

A főkulcsot a HKEY_LOCAL_MACHINE-ra állítjuk.

```
        RootKey := HKEY_LOCAL_MACHINE;
```

Megnyitjuk azt a címet, ahol a mail kliensek leírása található.

```
        if OpenKeyReadOnly(RegClientsroot +
          RegClientsMail) then
```

Ellenőrizzük, hogy ezen a címen található-e albejegyzés.

```
        if HasSubKeys then
```

Ha igen, akkor annak neveivel feltöltjük a paraméterként kapott listát. Ez a lista egyben az e-mail kliensek neveit adja meg, így máris elértük a kitűzött célunkat.

```
          GetKeyNames(AList);
    end;
```

Végezetül felszabadítjuk a lefoglalt erőforrást.

```
    finally
      if Assigned(reg) then reg.Free;
    end;
  except
    Result := False;
  end;
end;
```

Garami Fülöp

Felhasználók felvétele, törlése programból

Cikk sorszáma	Szoftverfeltételek
093	➢ Delphi
Forráskód	\Source\ManageUser\

Windows operációs rendszereken programból is van lehetőségünk új felhasználók felvételére, illetve törlésére. Ennek mikéntjére keressük a megoldást e cikkben.

Elméleti ismeretek

Először ismerkedjünk meg két Windows API függvénnyel, melyek kulcsszerepet játszanak a felhasználók létrehozásában és törlésében. Új felhasználót a NetUserAdd nevű függvénnyel hozhatunk létre, míg ha törölni kívánunk egy meglévő felhasználót a nyilvántartásból, akkor a NetUserDel függvényt kell alkalmaznunk.

NetUserAdd

A NetUserAdd függvény képes egy új felhasználó felvételére a helyi, vagy akár egy távoli gépen is. A függvény szintaxisa az alábbi:

```
NET_API_STATUS NetUserAdd(
  LMSTR servername,
  DWORD level,
  LPBYTE buf,
  LPDWORD parm_err
);
```

Paraméterek

- servername

DNS, vagy NetBIOS formában megadott számítógép név. A felhasználó az itt megadott számítógépen jön létre. Ha ez a paraméter NULL, akkor a helyi gépen jön létre az új felhasználó.

Windows NT esetén ezt a sztringet a \\ karakterekkel kell kezdeni.

- level

E paraméternél adható meg, hogy a felhasználói adatokat leíró buf nevű paraméter hányas szintű struktúrát használ. Lehetséges értékeke 1-4 között lehet.

1 - felhasználói információk megadása. A buf paraméter ebben az esetben egy USER_INFO_1 struktúrát kell, hogy tartalmazzon.

2 - felhasználói információk és további attribútumok megadása. A buf paraméter ebben az esetben egy USER_INFO_2 struktúrát kell, hogy tartalmazzon.

3 - a kettes szint adatain felül, további információk megadására van lehetőség. A buf paraméter ebben az esetben egy USER_INFO_3 struktúrát kell, hogy tartalmazzon.

4 - a kettes szint adatain felül, további információk megadására van lehetőség. Ez a szint csak szerver operációs rendszerek esetén használható. A buf paraméter ebben az esetben egy USER_INFO_4 struktúrát kell, hogy tartalmazzon.

- buf

A felhasználói adatokat leíró USER_INFO_1, USER_INFO_2, USER_INFO_3 vagy USER_INFO_4 struktúra a level paraméterben megadott számtól függően.

- parm_err

Egy egész számra mutató pointer, melyben eredményül kapunk egy index számot. További információ a NetUserSetInfo függvénynél található.

Visszatérési érték

A művelet sikeres végrehajtása esetén NERR_Success érték. Hiba esetén az alábbi hibakódok egyike:

- ERROR_ACCESS_DENIED - hozzáférés megtagadva
- NERR_InvalidComputer - a megadott számítógépnév nem helyes
- NERR_NotPrimary - a kért művelet csak elsődleges tartomány vezérlőn végezhető el
- NERR_GroupExists - a megadott csoport már létezik
- NERR_UserExists - a megadott felhasználó már létezik
- NERR_PasswordTooShort - a megadott jelszó nem fogadható el az érvényben lévő házirend szerint. Adjon meg hosszabb, erősebb jelszót.

Megjegyzések

Windows NT esetén a függvényt csak Administrators illetve Account Operators csoportba tartozó felhasználók hívhatják meg sikeresen. Administrators jogkörrel rendelkező felhasználót pedig csak Administrators csoportba tartozó személy vehet fel.

A felhasználói név hossza 20, a csoport neve 256 karakter hosszú lehet. A szöveg nem tartalmazhatja a", /, \, [,], :, |, <, >, +, =, ;, ?, * karaktereket, valamint a 1-31 kódú karaktereket, illetve minden egyéb nem nyomtatható karaktert sem.

Szoftverfeltételek

A függvény Windows NT, Windows 2000, Windows XP és Windows 2003 operációs rendszereken érhető el.

A függvény a Netapi32.dll-ben található.

NetUserDel függvény

Egy felhasználó törléséhez a NetUserDel függvényt alkalmazhatjuk. A függvény szintaxisa az alábbi:

```
NET_API_STATUS NetUserDel(
  LPCWSTR servername,
  LPCWSTR username
);
```

Paraméterek

- servername

DNS, vagy NetBIOS formában megadott számítógép név. A felhasználó az itt megadott számítógépen jön létre. Ha ez a paraméter NULL, akkor a helyi gépen jön létre az új felhasználó.

Windows NT esetén ezt a sztringet a \\ karakterekkel kell kezdeni.

- username

A törlendő felhasználói név.

Visszatérési érték

A művelet sikeres végrehajtása esetén NERR_Success érték. Hiba esetén az alábbi hibakódok egyike:

- ERROR_ACCESS_DENIED - a hozzáférés nem engedélyezett
- NERR_InvalidComputer - a megadott számítógép nem elérhető
- NERR_NotPrimary - a kért művelet csak elsődleges tartomány vezérlőn végezhető el
- NERR_UserNotFound - a megadott felhasználói név nem található

Megjegyzések

Windows NT esetén csak Administrators illetve Account Operators csoportba tartozó felhasználók hívhatják meg sikeresen ezt a függvényt. Administrators jogkörrel rendelkező felhasználót pedig csak Administrators csoportba tartozó személy törölhet.

Szoftverfeltételek

A függvény Windows NT, Windows 2000, Windows XP és Windows 2003 operációs rendszereken érhető el.

A függvény a Netapi32.dll-ben található.

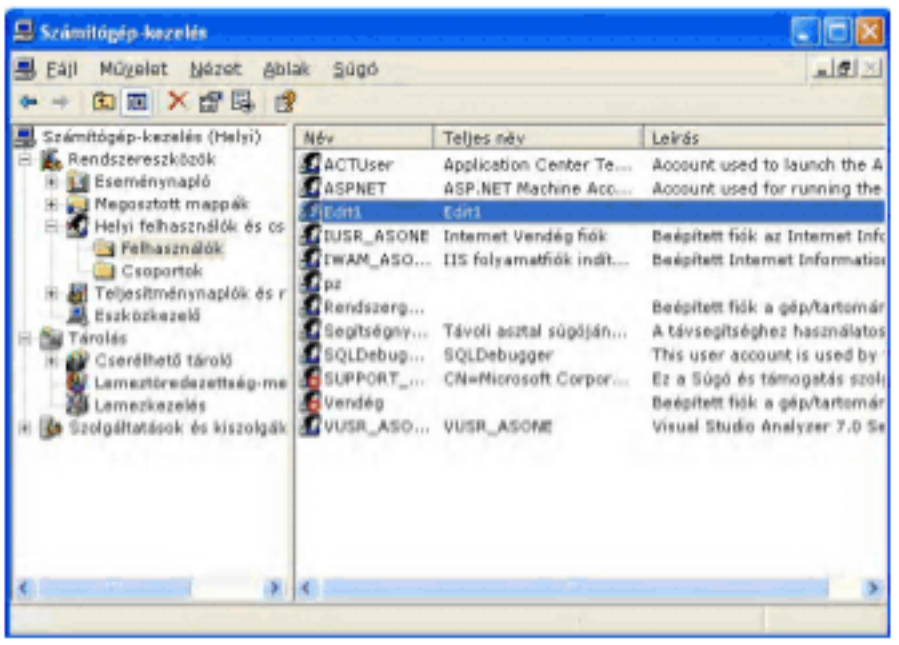
Gyakorlati megvalósítás

Példaprogram használata

Műveletek felhasználókkal

A mellékelt példaprogramban két Edit beviteli mezőt helyeztünk el. Egyikbe a létrehozandó, vagy törlendő felhasználó nevét, míg a másikba a létrehozandó felhasználóhoz tartozó jelszót kell megadnunk.

A Létrehozás gombra kattintva a helyi gépen létrejön a megadott nevű felhasználó. A Törlés gombra kattintva pedig eltávolíthatjuk a megadott nevű felhasználót.



Létrehozás ellenőrzése

A létrehozást követően ellenőrizhetjük is a Számítógép kezelés alkalmazás segítségével, a Helyi felhasználók és csoportok - Felhasználók kategórián belül látható az adott számítógép összes felhasználója, közte az újonnan felvett felhasználóval.

Példaprogram megvalósítása

A példában a USER_INFO_1 struktúrát használjuk a felhasználói adatok kezeléséhez:

```
type
  USER_INFO_1 = record
   usri1_name:pwidechar;
   usri1_password:pwidechar;
   usri1_password_age:dword;
   usri1_priv:dword;
   usri1_home_dir:pwidechar;
   usri1_comment:pwidechar;
   usri1_flags:dword;
   usri1_script_path:pwidechar;
  end;
  Buffer = ^USER_INFO_1;
```

A NetUserAdd és NetUserDel függvényeket külsőként kell deklarálnunk. Mindként függvény a Netapi32.dll-ben található:

```
function NetUserAdd(Server: PWideChar;
  Level: DWord; Buf: Pointer; ParmError:
  PDWord): LongInt; stdcall; external
  'netapi32.dll';
function NetUserDel(Server, UserName :
  PWideChar): LongInt; stdcall; external
  'netapi32.dll';
```

Felhasználó létrehozása

A Button1 gombra kattintva végezhetjük el az új felhasználó létrehozását.

```
procedure TForm1.Button1Click(
  Sender: TObject);
var
 Buf: Buffer;
 Err: Integer;
 user : PWideChar;
 password : PWideChar;
begin
```

E műveletre csak akkor kerülhet sor, ha a felhasználói név és jelszó is meg lett adva.

```
 if (edit1.Text <> '') and (edit2.Text <>
   '') then begin
```

Mind a két sztringet PWideChar típusra kell konvertálnunk.

```
   user := PWideChar(WideString(
     edit1.Text));
   password := PWideChar(WideString(
     edit2.Text));
```

Ezt követően memóriaterületet foglalunk a USER_INFO_1 struktúra számára.

```
   GetMem(Buf, SizeOf(USER_INFO_1));
```

Feltöltjük adatokkal a USER_INFO_1 struktúrát.

```
   with Buf^ do
     begin
       usri1_name:= user;
       usri1_password:= password;
       usri1_password_age:= 0;
       usri1_priv:= 1;
       usri1_home_dir:= nil;
       usri1_comment:= nil;
       usri1_flags:= 1;
       usri1_script_path:= nil;
     end;
```

Majd meghívjuk a NetUserAdd függvényt, oly módon paraméterezve, hogy az új felhasználó a helyi számítógépen kerüljön létrehozásra.

```
   NetUserAdd(nil, 1, Pointer(Buf), @Err);
```

Felszabadítjuk a lefoglalt memória területet.

```
   FreeMem(Buf);
```

Végül tudatjuk a felhasználóval az új felhasználó létrehozását.

```
   Application.MessageBox(PChar(
     'Felhasználó létrehozva'),'',0)
 end;
end;
```

Felhasználó törlése

A Button2-re kattintva törölhetjük az Edit1-ben megadott nevű felhasználót.

```
procedure TForm1.Button2Click(
  Sender: TObject);
var i : longint;
    user : PWideChar;
begin
```

A műveletet csak akkor hajtjuk végre, ha van megadva felhasználói név.

```
  if edit1.Text <> '' then begin
```

A felhasználó nevet tartalmazó sztringet PWideChar típusra kell konvertálnunk.

```
    user := PWideChar(WideString(
      edit1.Text));
```

Töröljük a helyi számítógépen található, megadott nevű felhasználót.

```
    i := NetUserDel(nil,user);
```

A művelet eredményességét jelezzük a felhasználó felé.

```
    if i = 0 then Application.MessageBox(
      PChar('Felhasználó törölve!'),'',0)
    else Application.MessageBox(PChar('A
      törlés nem sikerült!'),'',0)
  end;
end;
```

Győri Béla

Kapcsolódó webhelyek

NetUserAdd ↳ 009
NetUserDel ↳ 010

Form méretének korlátozása

Cikk sorszáma	Szoftverfeltételek
094	➢ Delphi
Forráskód	\Source\MinMax\

Alapértelmezett esetben a felhasználó szabadon átméretezheti Form-jaink szélességét, magasságát. Vannak azonban olyan esetek, amikor bizonyos mérték alá, esetleg fölé nem szeretnénk menni. Ilyen esetben korlátozhatjuk a szélesség, magasság minimális, illetve maximális értékét.

A mellékelt példaprogramban elhelyezünk a Form-on egy ListBox-ot, majd korlátozzuk a Form magasságát és szélességét úgy, hogy azt ne lehessen lekicsinyíteni oly mértékben, hogy a ListBox ne legyen látható teljesen.

WM_GETMINMAXINFO

A Form minimális szélessége, magassága

Amikor egy Form átméretezése vagy mozgatása történik, akkor kap az ablakunk egy WM_GETMINMAXINFO üzenetet. Amennyiben ezt feldogozzuk, akkor lehetőségünk van korlátozni a felhasználót abban, hogy a Form-nál mekkora lehet a minimális és maximális szélesség, valamint magasság, amit még beállíthat.

Továbbá arról is rendelkezhetünk, hogy mely területen mozoghat a Form, vagyis mi lehet a Left és Top tulajdonságainak maximális értéke.

Az WM_GETMINMAXINFO üzenet az alábbiak szerint értelmezendő:

```
WM_GETMINMAXINFO
    WPARAM wParam
    LPARAM lParam;
```

Paraméterek

- wParam

A wParam ennél az üzenetnél nem használatos.

- lParam

Egy mutatót kapunk a MINMAXINFO struktúrára, melyben megadhatók a korlátozó értékek.

Visszatérési érték

Amennyiben az alkalmazásunk feldolgozta az üzenetet, a visszatérési értéket nullára kell állítanunk.

MINMAXINFO struktúra

A WM_GETMINMAXINFO üzenet lParam paraméterében kapjuk meg a MINMAXINFO struktúrát, melyen keresztül szabályozhatjuk a minimális, maximális értékeket.

A struktúra deklarációja az alábbi:

```
typedef struct tagMINMAXINFO {
   POINT ptReserved;
   POINT ptMaxSize;
   POINT ptMaxPosition;
   POINT ptMinTrackSize;
   POINT ptMaxTrackSize;
} MINMAXINFO;
```

Paraméterek

- ptReserved

Belső használatra fenntartott.

- ptMaxSize

Az ablak maximális szélességét a ptMaxSize.X, míg a maximális magasságát a ptMaxSize.Y értéknél szabályozatjuk.

- ptMaxPosition

Az ablak Left tulajdonságának maximális értékét a ptMaxPosition.X, míg a Top tulajdonságának maximális értékét a ptMaxPosition.Y elemnél adhatjuk meg.

- ptMinTrackSize

Az ablak átméretezés közbeni minimális szélessége és magassága a ptMinTrackSize.X és ptMinTrackSize.Y értékeknél szabályozható.

- ptMaxTrackSize

Az ablak átméretezés közbeni maximális szélessége és magassága a ptMinTrackSize.X és ptMinTrackSize.Y értékeknél szabályozható.

TWMGetMinMaxInfo rekord

A WM_GETMINMAXINFO üzenet kezeléséhez a Delphi-ben rendelkezésünkre áll a TWMGetMinMaxInfo nevű rekord:

```
TWMGetMinMaxInfo = packed record
   Msg: Cardinal;
   Unused: Integer;
   MinMaxInfo: PMinMaxInfo;
   Result: Longint;
 end;
```

TMinMaxInfo rekord

A TWMGetMinMaxInfo rekord MinMaxInfo nevű eleme egy újabb rekordot tartalmaz, mely a MINMAXINFO struktúra Delphi-s megfelelője:

```
 PMinMaxInfo = ^TMinMaxInfo;
 tagMINMAXINFO = packed record
   ptReserved: TPoint;
   ptMaxSize: TPoint;
   ptMaxPosition: TPoint;
   ptMinTrackSize: TPoint;
   ptMaxTrackSize: TPoint;
 end;
 TMinMaxInfo = tagMINMAXINFO;
```

Példaprogram megvalósítása

A WM_GETMINMAXINFO üzenet kezeléséhez létrehozunk egy üzenet kezelő eljárást a Form1-en belül. Ez az eljárás a GetMinMaxInfo nevet kapja.

```
TForm1 = class(TForm)
    ListBox1: TListBox;
  private
     procedure GetMinMaxInfo(var Msg:
       TWMGETMINMAXINFO);
       message WM_GETMINMAXINFO;
  public
    { Public declarations }
  end;
```

GetMinMaxInfo eljárás

A GetMinMaxInfo eljárásban korlátozzuk a Form minimális szélességét, így a felhasználó nem lesz képes futási időben az itt megadott 200, 200 értéknél kisebb méretűre állítani a Form-ot, így az azon elhelyezett ListBox minden esetben látható marad teljes terjedelmében.

```
procedure TForm1.GetMinMaxInfo(var
  Msg: TWMGETMINMAXINFO);
begin
inherited;
  with Msg.MinMaxInfo^ do
  begin
    ptMinTrackSize.X := 200;
    ptMinTrackSize.Y := 200;
  end;
end;
```

Hoplár Péter

Kapcsolódó webhelyek

WM_GETMINMAXINFO ↳ 011

Program futtatása meghatározott processzoron

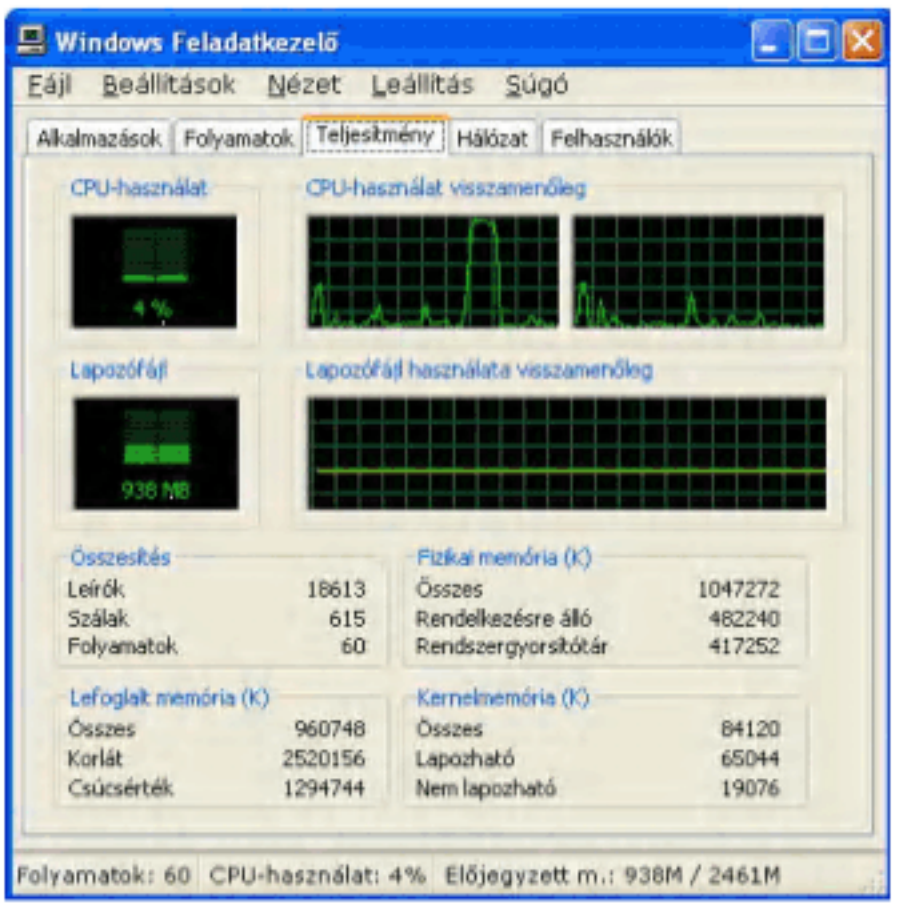
Cikk sorszáma	Szoftverfeltételek
095	➢ Delphi
Forráskód	\Source\MultiProc\

Ha számítógépünk több processzort, vagy többmagos processzort, esetleg Hyper Threading képességgel rendelkező processzort tartalmaz, akkor lehetőségünk van arra, hogy programból szabályozzuk azt, hogy a futtatandó folyamataink melyik processzort használják és melyiket ne.

Több folyamatot indítva így mi magunk is befolyásolhatjuk az egyes processzorok közötti feladat elosztást.

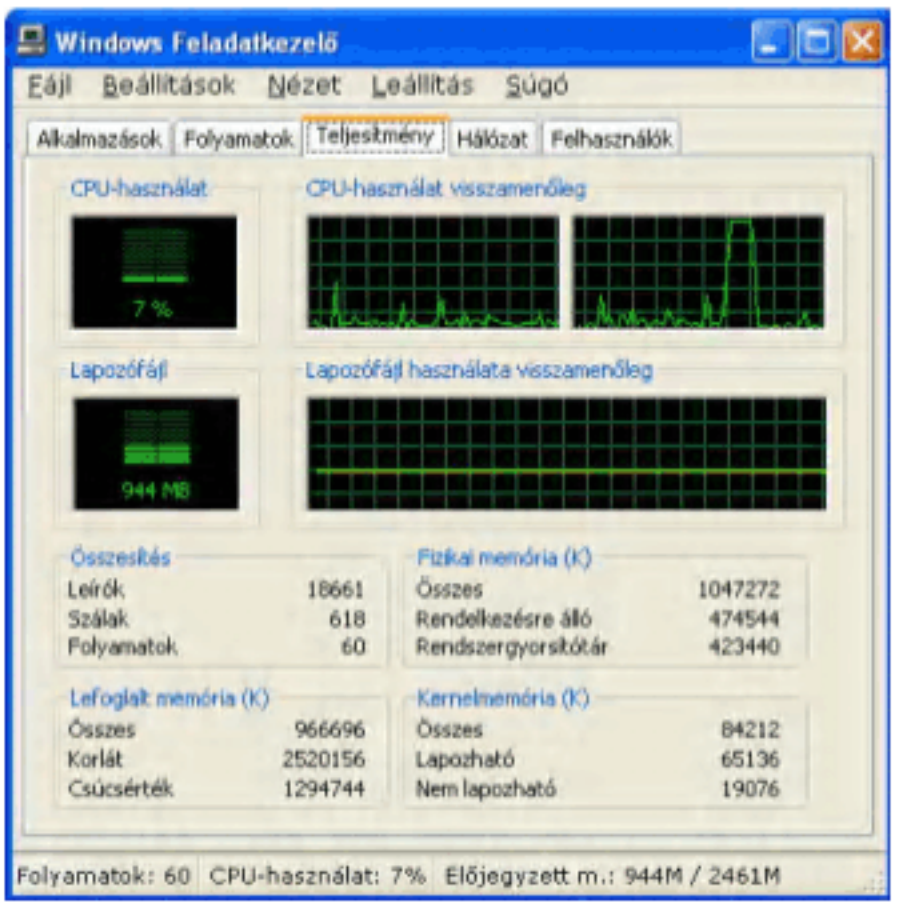
A cikk mellékletét képező példa két darab példaprogramból áll. A kipróbáláshoz először fordítsa le a \MultiProc\Progress\ mappában lévő alkalmazást, majd ezt követően futtassa a \MultiProc\ mappában található programot.

A próba



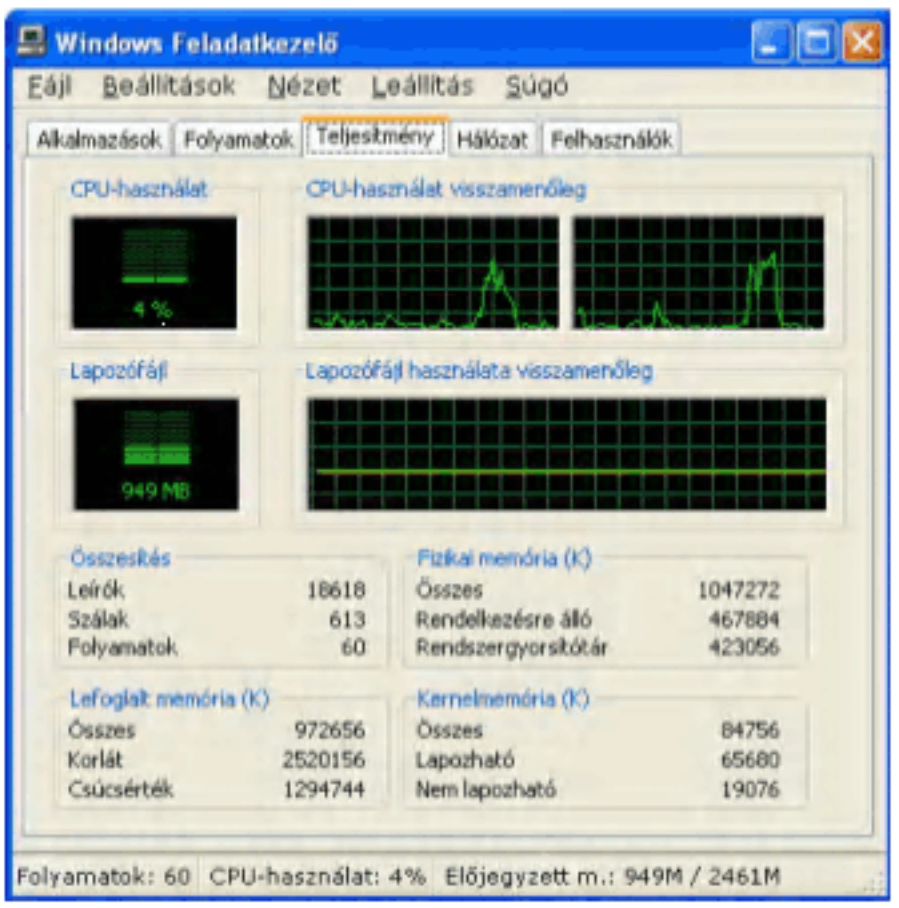
Folyamat futtatása az első processzoron

Ha a folyamatot az első processzoron futtatjuk le, akkor jó megfigyelhető a Windows Feladatkezelőben, hogy jelentős mértékben csupán az első processzor kihasználtsága nő meg.



Folyamat futtatása a második processzoron

Elvégezve a tesztet a másik processzorra is a Feladatkezelő ablakában megfigyelve az eseményeket egyértelműen látható, hogy most a második processzor terheltsége nő.



Folyamat futtatása mindkét processzoron

Abban az esetben, ha nem rendelkezünk arról, hogy a folyamat melyik processzoron fusson, akkor a feladaton egyaránt osztozik a két processzor, a Windows Feladatkezelő grafikonjain megjelenő görbe alapján.

A teszt folyamat

A \MultiProc\Progress\ mappában található programban valósítunk meg egy olyan folyamatot, mely jelentős mértékben terheli a processzszort, de csupán pár másodpercig. E folyamat nem lesz más, mint egy ProgressBar komponens aktuális értékének növelése a minimum értékétől a maximum értékéig. Ezt a folyamatos növelést egy ciklussal végeztetjük el, melyben meghívjuk minden alkalommal az Application osztály ProcessMessages eljárását is. Ez utóbbi feldolgoztatja az alkalmazáshoz időközben beérkezett üzeneteket.

A ciklus végén bezárjuk az alkalmazást.

```
procedure TForm1.Timer1Timer(
  Sender: TObject);
var
  i: integer;
begin
  Timer1.Enabled := false;
  for i:=0 to ProgressBar1.Max do begin
    ProgressBar1.StepIt;
    Application.ProcessMessages;
  end;
  Close;
end;
```

Elméleti alapok

Mielőtt folytatnánk a feladat megvalósítását, meg kell ismernünk néhány kulcsszerepet játszó függvényt, melyekkel megvalósítható az, hogy az egyes folyamatokat egy általunk előírt processzor futtassa le.

CreateProcess függvény

A CreateProcess függvényt használhatjuk arra, hogy egy programból egy másik folyamatot indítsunk el. A függvénynek egy futtatható program állományát kell megadni. Ezt indítja el és ennek befolyásolhatjuk azt a tulajdonságát, hogy melyik processzort használja a feladatai elvégzéséhez.

A futtatandó programot akár másik felhasználó nevében is elindíthatjuk, ha ismerjük annak bejelentkezési adatait. Ekkor azonban a CreateProcessAsUser, illetve

CreateProcessWithLogonW függvények valamelyikére lesz szükségünk.

A CreateProcess függvény deklarációja az alábbi:

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,
  LPTSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES
lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL bInheritHandles,
  DWORD dwCreationFlags,
  LPVOID lpEnvironment,
  LPCTSTR lpCurrentDirectory,
  LPSTARTUPINFO lpStartupInfo,
  LPPROCESS_INFORMATION
lpProcessInformation
);
```

Paraméterek

- lpApplicationName

Itt kell megadnunk sztringként azt a programot elérési úttal együtt, melyet indítani kívánunk. Ha nem adunk meg meghajtót, vagy elérési utat, akkor a keresés az aktuális meghajtó, aktuális mappájában történik a megadott program után.

Amennyiben nem adunk meg kiterjesztést, akkor a függvény az EXE kiterjesztésű állományt feltételezi.

Ha nem adunk meg elérési útvonalat, akkor a következők szerint történik a keresés:

Először abban a mappában keresi a CreateProcess függvény a megadott programot, melyből az alkalmazásunk is el lett indítva. Ha itt nem találja, akkor az aktuális mappában keres. Ha itt sincs, akkor 32 bites Windows esetén a Windows mappa System32 mappájában végez keresést. E mappa pontos helyét a GetSystemDirectory függvény adja vissza. 16 bites Windows esetén szintén a GetSystemDirectory függvény által megadott helyen keres. Ha továbbra sincs találat, akkor a GetWindowsDirectory függvény által adott elérési úton keresi a futtatandó programot. Ezek után a PATH környezeti változóba megadott elérési utakat nézi sorra. Ha még itt sincs találat, akkor hibajelzést kapunk.

Az lpApplicationName paraméternél adhatunk meg akár üres sztringet is. Ebben az esetben az lpCommandLine paraméternél kell megadnunk az indítandó alkalmazást oly módon, hogy az átadott sztring első szóközig tartó szakasza ennek a programnak az elérési útját, nevét írja le.

Például:

```
c:\Mappa\abc.exe /u /j /t=75
```

Ebben az esetben viszont ügyeljünk arra, hogy a mappák, állományok nevei ne tartalmazzanak szóközt, mert ez esetben hibásan értelmezheti a függvény a megadott értéket. Például a "c:\program files\sub dir\program name" elérési út megadása értelmezhető az alábbiak szerint:

```
c:\program.exe files\sub dir\program name
c:\program files\sub.exe dir\program name
c:\program files\sub dir\program.exe name
c:\program files\sub dir\program name.exe
```

- lpCommandLine

Szükség esetén a futtatandó alkalmazás számára itt adhatunk meg parancssori paramétereket. Az átadható sztring hossza maximum 32 786 karakter lehet. Windows 2000 esetén a maximális hosszt a MAX_PATH konstans adja meg.

Amennyiben Unicode karakter megadására is szükségünk lenne, akkor használjuk a CreateProcessW függvényt.

E paraméter értéke lehet NULL, de az lpApplicationName és lpCommandLine paraméterek közül legalább az egyiknek minden esetben kell értéket tartalmaznia.

- lpProcessAttributes

Ebben a paraméterben egy SECURITY_ATTRIBUTES struktúrát adhatunk meg, melyben leírhatjuk az indítandó modul biztonsági tulajdonságait. Null érték esetén az alapértelmezett biztonsági leíró kerül hozzárendelésre.

- lpThreadAttributes

Ebben a paraméterben szintén egy SECURITY_ATTRIBUTES struktúrát adhatunk meg, melyben leírhatjuk az indítandó modul fő szálához rendelendő biztonsági tulajdonságokat. Null érték esetén az alapértelmezett biztonsági leíró kerül hozzárendelésre.

- bInheritHandles

Ha e paraméter igaz, akkor az indítandó alkalmazás örökli az azonosítókat (Handle), ellenkező esetben nem. Öröklődés esetén az elindított alkalmazásnak ugyanolyan jogai lesznek, mint annak a programnak mely elindította.

- dwCreationFlags

A futtatandó alkalmazás főszálának prioritását állíthatjuk be ennél a paraméternél. Az érték az alábbiak egyike lehet.

Érték	Leírás
ABOVE_NORMAL_PRIORITY_CLASS	A NORMAL_PRIORITY_CLASS szintjénél nagyobb, de a HIGH_PRIORITY_CLASS szintjénél kisebb prioritás.
BELOW_NORMAL_PRIORITY_CLASS	Az IDLE_PRIORITY_CLASS szintjénél nagyobb, de a NORMAL_PRIORITY_CLASS kisebb prioritás.
HIGH_PRIORITY_CLASS	Magas szintű prioritás. Olyan alkalmazások esetén szükséges, ahol fontos a pontos időzítés, a gyors rendelkezésre állás. Ilyen például a Task List, melynek gyorsan kell reagálnia minden helyzetben a felhasználói kérések kiszolgálásához.
IDLE_PRIORITY_CLASS	Alacsony szintű prioritás. Olyan programok esetén alkalmazandó, melyeknek feladatunk időbeli végrehajtása nem fontos tényező. Lassú, hosszú folyamatokat érdemes ezzel a prioritással futtatnunk, ha az elvégzendő folyamat eredménye nem sürgős. Az így indított folyamat csak akkor terheli a processzort, ha az amúgy nem végezne semmilyen különös munkát. Ez által kihasználhatjuk az üresjárati időt hasznos tevékenységre, de amint más programot is használnánk, annak futási sebességét nem befolyásolja az IDLE_PRIORITY_CLASS prioritással futó folyamat.
NORMAL_PRIORITY_CLASS	Normál prioritás. Alapértelmezett esetben alkalmazható prioritás.
REALTIME_PRIORITY_CLASS	Legmagasabb szintű prioritás. Valósidejű alkalmazások számára felhasználható prioritási szint, ahol az idő és rendelkezésre állás kritikus tényező a folyamat futása során. Ilyen például az egér mozgását a képernyőn megjelenítő folyamat prioritása.

- lpEnvironment

Környezeti változókat adhatunk át az indítandó folyamatnak az lpEnvironment paraméterben. Minden környezeti változó az alábbi formában adandó meg:

```
name=value
```

Minden megadott környezeti változó sztringjét egy nulla karakter kell hogy zárja. Így több környezeti változó is megadható egymás után. A teljes blokkot egy újabb nulla karakterrel kell zárnunk.

A sztring tartalmazhat Unicode és ANSI karaktereket. Unicode karakterek használata esetén a dwCreationFlags paraméternek tartalmaznia kell a CREATE_UNICODE_ENVIRONMENT konstanst is.

ANSI karakterek esetén tehát a megadott blokkot két darab nulla bájt zárja: egyik az utolsó sztringet, a másik pedig a teljes blokkot. Ugyanennél a helyzetnél Unicode karakterek esetén négy darab nulla bájtra lesz szükségünk, hiszen ott minden karaktert két-két bájt ír le.

- lpCurrentDirectory

Kijelölhetünk a futtatandó program számára egy aktuális könyvtárat. Amennyiben ezt nem adjuk meg, akkor a hívó alkalmazás aktuális könyvtára kerül átadásra.

- lpStartupInfo

E paraméterben egy STARTUPINFO struktúrát adhatunk meg, melyben leírhatjuk, hogy az indítandó program legyen-e látható, milyen módon jelenjen meg.

- lpProcessInformation

A CreateProcess függvény sikeres futása esetén az lpProcessInformation paraméterben kapunk vissza információkat az elindított alkalmazásról.

Visszatérési érték

Ha a CreateProcess függvény futása sikeres, akkor a függvény visszatérési értéke igaz, különben hamis lesz. Utóbbi esetben a GetLastError függvény hívása ad pontosabb tájékoztatást a hiba okáról.

SetProcessAffinityMask függvény

Példaprogramunk fő feladata lesz, hogy az elindított alkalmazást egy megadott processzoron futtassa le. Ennek megadásához használhatjuk fel a SetProcessAffinityMask függvényt.

A függvény deklarációja az alábbi:

```
BOOL SetProcessAffinityMask(
  HANDLE hProcess,
  DWORD_PTR dwProcessAffinityMask
);
```

Paraméterek

- hProcess

Itt kell megadnunk annak a folyamatnak az azonosítóját, melynél a beállítást el kívánjuk végezni. E művelet elvégzésére csak akkor lesz lehetőségünk, ha a folyamat rendelkezik PROCESS_SET_INFORMATION joggal.

- dwProcessAffinityMask

Itt adható meg, hogy a folyamat melyik processzort, vagy processzorokat használja.

A DWORD érték minden egyes bitje egy-egy processzort jelent. Azok a bitek, melyek 1-es értéket tartalmaznak, jelölik meg azokat a processzorokat, ahol a folyamat futhat.

Visszatérési érték

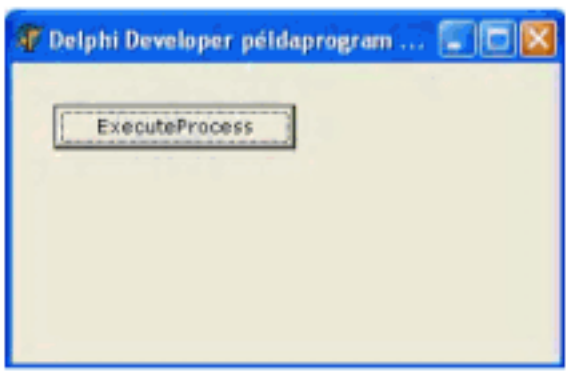
Ha a beállítási művelet sikeres, akkor a függvény visszatérési értéke igaz, különben hamis lesz. Utóbbi esetben a GetLastError függvény hívása ad pontosabb tájékoztatást a hiba okáról.

Megjegyzések

A függvényhez minimálisan Windows NT, vagy újabb verziójú Windows szükséges.

A függvény a Kernel32.dll-ben található.

Gyakorlati megvalósítás



Mellékelt példaprogram

Az elméleti részben áttekintett függvény egyszerűbb használatához készítünk egy ExecuteProcess nevű függvényt, mellyel könnyedén indíthatunk más alkalmazásokat előírva számukra, hogy melyik processzoron fussanak.

```
function ExecuteProcess(FileName: string;
  Visibility: Integer; BitMask: Integer;
  Synch: Boolean): Longword;
```

A függvény FileName paraméterében kell megadnunk a futtatandó program nevét, elérési útját. A program láthatóságát a Visibility paraméter írja le. Ennek értéke az alábbiak egyike lehet:

- SW_HIDE - az elindított program ablaka nem lesz látható, így az a felhasználó elől rejtve fut le.
- SW_MAXIMIZE - az elindított program ablaka maximalizálva lesz.
- SW_MINIMIZE - az elindított program minimalizálva lesz.
- SW_SHOW - az elindított program ablaka az alapértelmezett méretében jelenik meg.
- SW_SHOWMAXIMIZED - az elindított program ablaka maximalizálva jelenik meg és a program aktivizálva is lesz.
- SW_SHOWMINIMIZED - az elindított program ablaka minimalizálva jelenik meg és a program aktivizálva is lesz.
- SW_SHOWNORMAL - az elindított program ablaka az alapértelmezett méretében jelenik meg és aktivizálva is lesz.

A függvény Bitmask paraméterében írhatjuk elő, hogy az elindított alkalmazás mely processzorokon fusson. A megadott szám minden bitje egy-egy processzort jelképez. Ha egy bit egy, akkor az adott sorszámú processzor futtatja az alkalmazást.

A Synch paraméter igazra állításával az alkalmazásunk megvárja, míg az elindított alkalmazás futása véget nem ér. Erre az időtartamra az alkalmazásunk futása megszakad. Hamis érték esetén azonnal fut tovább a mi programunk is, amint az újat elindítottuk.

```
function ExecuteProcess(FileName: string;
  Visibility: Integer; BitMask: Integer;
  Synch: Boolean): Longword;
var
  zAppName: array[0..512] of Char;
  zCurDir: array[0..255] of Char;
  WorkDir: string;
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
  Closed: Boolean;
begin
```

Első lépésként összeállítjuk a program indításához szükséges adat struktúrákat.

```
  Closed := True;
  StrPCopy(zAppName, FileName);
  GetDir(0, WorkDir);
  StrPCopy(zCurDir, WorkDir);
  FillChar(StartupInfo, SizeOf(
    StartupInfo), #0);
  StartupInfo.cb := SizeOf(StartupInfo);
  StartupInfo.dwFlags
    := STARTF_USESHOWWINDOW;
  StartupInfo.wShowWindow := Visibility;
```

Ezt követően megpróbáljuk elindítani a megadott folyamatot.

```
  if not CreateProcess(nil,
    zAppName,
    nil,
    nil,
    False,
    CREATE_NEW_CONSOLE or
    NORMAL_PRIORITY_CLASS,
    nil,
    nil,
    StartupInfo,
    ProcessInfo)
    then Result := WAIT_FAILED
  else
  begin
```

Sikeres indítás esetén a SetProcessAffinityMask függvény hívásával beállítjuk azt, hogy a folyamat mely processzorokon fusson. Az újonnan indult alkalmazás azonosítóját a CreateProcess függvény a ProcessInfo paraméter hProcess változójába adja vissza.

```
    SetProcessAffinityMask(ProcessInfo.
      hProcess, BitMask);
```

Amennyiben szükséges megvárnunk az elindított folyamat végét, úgy egy ciklust kezdeményezünk, melyben folyamatosan hívjuk a WaitForSingleObject függvényt. Ezzel ellenőrizhetjük, hogy az elindított alkalmazás még fut-e.

```
    if (Synch = True) then
      begin
        Closed:= False;
        repeat
          case WaitForSingleObject(
            ProcessInfo.hProcess, 100) of
              WAIT_OBJECT_0 : Closed:=
True;
              WAIT_FAILED
                : RaiseLastWin32Error;
          end;
          Application.ProcessMessages;
        until (Closed);
```

A futásának végén lekérdezzük a kilépési kódját a GetExitCodeProcess függvény segítségével. A kapott értéket a függvény visszatérési értékeként szolgáltatjuk.

```
        GetExitCodeProcess(ProcessInfo.
          hProcess, Result);
```

Végezetül zárjuk a megnyitott erőforrásokat.

```
        CloseHandle(ProcessInfo.hProcess);
        CloseHandle(ProcessInfo.hThread);
      end
    else  begin
        Result := 0;
      end;
  end;
end;
```

Button1 OnClick esemény

Fenti előkészületek után a Button1-re történő kattintáskor már igen egyszerű a dolgunk: meghívjuk az ExecuteProcess függvényünket, paraméterként átadva számára a Progress mappában lévő alkalmazást indításra.

```
procedure TForm1.Button1Click(
  Sender: TObject);
begin
  ExecuteProcess(ExtractFilePath(
    Application.ExeName) +
    '\Progress\Progress.exe', SW_SHOW,
    1, false);
end;
```

Kendes Máté

Kapcsolódó webhelyek

CreateProcess ↳ 012
SetProcessAffinityMask ↳ 013

Hangállományok párhuzamos lejátszása

Cikk sorszáma	Szoftverfeltételek
096	➢ Delphi
Forráskód	\Source\Wavs\

Számítógépünk segítségével lehetőségünk van több hangállomány egyidejű lejátszására is. Ezt kihasználva például egy folyamatos háttérzene közben is adhat programunk különféle hangjelzéseket, bizonyos események bekövetkezésekor.

A mellékelt példaprogram futtatása előtt ellenőrizze, hogy az alábbi állományok adottak-e számítógépén, ha nem, akkor módosítsa a forráskódot, két másik, ám de létező állományt választva:
C:\WINDOWS\Media\tada.wav
C:\WINDOWS\Media\ringin.wav

Elméleti alapok

A feladataink megvalósításához két multimédiás függvényt használunk fel: az mciSendString segítségével küldjük a különféle üzeneteket, míg a mciGetErrorString függvényt használva kérdezzük le az esetlegesen fellépő hibák üzeneteinek szövegét.

mciSendString függvény

Az mciSendString függvény használható arra, hogy MCI eszközöknek parancsokat küldjünk. A parancsokat sztringként adhatjuk meg.

A függvény szintaxisa az alábbi:

```
MCIERROR mciSendString(
  LPCTSTR lpszCommand,
  LPTSTR lpszReturnString,
  UINT cchReturn,
  HANDLE hwndCallback
);
```

Paraméterek

- lpszCommand

Ebben a paraméterben adhatjuk meg szövegesen a végrehajtani kívánt parancsot. E parancsok az alábbiak egyike lehet:

break
capability
capture
close
configure
copy
cue
cut
delete
escape
freeze
index
info
list
load
mark
monitor
open
paste
pause
play
put
quality
realize
record
reserve

restore
resume
save
seek
set
setaudio
settimecode
settuner
setvideo
signal
spin
status
step
stop
sysinfo
undo
unfreeze
update
where
window

- lpszReturnString

Az lpszReturnString paraméterben visszakapunk egy szöveges információt a végrehajtott paranccsal kapcsolatban. Ha olyan parancsot hívunk, mely nem ad vissza értéket, akkor e paraméter a hívás után NULL lesz.

- cchReturn

Az lpszReturnString paraméter hossza bájtokban. E paraméterből tudhatjuk meg, hogy kaptunk-e vissza sztringet az lpszReturnString paraméterben és ha igen, akkor az hány karaktert tartalmaz.

- hwndCallback

Callback ablak leírója.

Visszatérési érték

Ha a függvény visszatérési értéke nulla, akkor a parancs futása sikeresen befejeződött. Ellenkező esetben egy hibakódot kapunk, melyet szöveges üzenetté alakíthatunk a mciGetErrorString függvény hívásával.

Szoftverfeltételek

Windows 95, illetve Windows NT 3.1 óta használható függvény, azonban a 64 bites Windows verziókban már nem használható.

mciGetErrorString függvény

Hiba esetén a hibakód alapján az mciGetErrorString függvény keresi elő a hibaüzenet szöveges leírását.

A függvény szintaxisa:

```
BOOL mciGetErrorString(
  DWORD fdwError,
  LPTSTR lpszErrorText,
  UINT cchErrorText
);
```

Paraméterek

- fdwError

Az mciSendCommand függvény vagy az mciSendString függvény által visszaadott hibakód.

- lpszErrorText

A hibaüzenet szövegét ebbe a paraméterbe kapjuk vissza.

- cchErrorText

Az lpszErrorText karaktereinek számát ebbe a paraméterbe kapjuk vissza.

Visszatérési érték

Ha a hibaüzenet szövegének lekérdezése sikeres, akkor igaz, különben hamis.

Megjegyzés

Bármely e függvény által visszaadott hibaüzenet hossza maximum 128 karakter lehet.

Példaprogram elkészítése

Mellékelt példaprogram

A mellékelt példaprogram kipróbálása egyszerű: kattintsunk a Button1 gombra. Ennek hatására párhuzamosan lejátszásra kerül két WAV állomány.

SendMCICommand függvény

Első lépésként készítünk egy SendMCICommand nevű függvényt, melyet arra használunk, hogy parancsokat küldjünk és hiba esetén kezeljük a hibaüzeneteket is.

```
procedure TForm1.SendMCICommand(
  Cmd: string);
var
  RetVal: Integer;
  ErrMsg: array[0..254] of char;
begin
```

Elsőként az mciSendString függvényt hívjuk kiadva a kért parancsot.

```
  RetVal := mciSendString(PChar(Cmd), nil,
    0, 0);
```

Ha hiba történik a parancs feldolgozásánál, akkor az mciGetErrorString függvény segítségével lekérdezzük a hibaüzenetet.

```
  if RetVal <> 0 then begin
    mciGetErrorString(RetVal, ErrMsg, 255);
```

A lekérdezett hibaüzenetet a MessageDlg eljárással jelenítjük meg a programunkban.

```
    MessageDlg(StrPas(ErrMsg), mtError,
      [mbOK], 0);
  end;
end;
```

Button1 OnClick esemény

Hang állomány lejátszásához először az open parancsot kell kiküldenünk. Mivel WAV állományokat kívánunk lejátszani, ezért a megnyitáshoz szükséges megadnunk a waveaudio paramétert is. Arra is szükségünk lesz, hogy párhuzamosan több állományt lejátszhassunk. Ennek érdekében nincs más teendő, mint a shareable paraméter megadása.

```
procedure TForm1.Button1Click(
  Sender: TObject);
begin
  SendMCICommand('open
    waveaudio shareable');
```

Hang állományt lejátszani a play paranccsal tudunk. Paraméterként a lejátszandó állomány nevét, elérési útját kell megadnunk.

```
  SendMCICommand('play
    "C:\WINDOWS\Media\tada.wav"');
```

Ezt követően rögtön indíthatjuk a másik állomány lejátszását is.

```
  SendMCICommand('play
    "C:\WINDOWS\Media\ringin.wav"');
```

Végül zárjuk a kapcsolatot a close paranccsal. A hang állományok természetesen teljes terjedelmükben lejátszásra kerülnek.

```
  SendMCICommand('close waveaudio');
end;
```

Miksa Elöd

Kapcsolódó webhelyek

mciSendString	↳ 016
Multimedia Commands	↳ 017

Színkódok meghatározása

Cikk sorszáma	Szoftverfeltételek
097	➢ Delphi
Forráskód	\Source\Zoom\

Készítünk egy kis grafikai alkalmazást, mely képes tetszőleges mértékben felnagyítva megjeleníteni a képernyőnek egy területét, amely felett épp az egér tartózkodik. A terület közepén lévő pixel RGB színkódjait decimális és hexadecimális formában is megjelenítjük a Form-on. Ezt felhasználva könnyedén meghatározhatjuk egy-egy pont színét és azt tetszés szerint felhasználhatjuk. Például egy weblap készítésénél a hexadecimális módon megjelenített színkódot közvetlenül felhasználhatjuk a HTML kódba történő beillesztéshez.

A példaprogram megvalósítása

Felnagyított képernyő

A feladat megvalósításához folyamatosan kell figyelnünk az egér mozgását a teljes képernyőn attól függetlenül, hogy melyik program az aktív. Ehhez egy Timer komponenst használunk fel, mellyel szinte folyamatosan futtatjuk az egér aktuális pozíciójának lekérdezését, amelyhez a GetCursorPos függvényt kell meghívnunk. E függvény az egér pillanatnyi X, Y koordinátáját adja vissza TPoint típusban. A koordináták a képernyő koordináta rendszerében értelmezendők.

```
procedure TForm1.Timer1Timer(
  Sender: TObject);
var
  Srect,Drect:TRect;
  zoom, iWidth,iHeight,DmX,DmY:Integer;
  iTmpX,iTmpY:Real;
  C:TCanvas;
  hDesktop: Hwnd;
  cur:TPoint;
  co: TColor;
begin
  GetCursorPos(cur);
```

Amint adott egy új koordináta, nincs más dolgunk, mint az adott pont körüli képernyőképet felnagyítva megjeleníteni a programunk Formjának területén. E művelet elvégzéséhez szükségünk lesz egy TCanvas objektumra.

```
  c:=TCanvas.Create;
```

A TCanvas objektumhoz egy új azonosítót rendelünk a Handle tulajdonságán keresztül. Tesszük ezt azért, hogy a TCanvas képes legyen a teljes képernyő területet elérni. Ehhez a GetWindowDC függvényt hívjuk meg, mely a GetDeskTopWindow által adott érték alapján képes egy olyan grafikai leírót szolgáltatni, melyet már felhasználhatunk a TCanvas osztály Handle tulajdonságának történő értékadáshoz.

Az így beállított TCanvas objektummal akár rajzolhatnánk is a Form teljes területén bárhová, bármit.

```
  c.Handle:=GetWindowDC(GetDeskTopWindow);
```

Elsőként az egér koordinátájában lévő pixel színét olvassuk ki egy TColor változóba. Ehhez a TCanvas osztály Pixels tulajdonságát használjuk, melynek mint egy két dimenziós tömbnek megadva a kérdéses X, Y koordinátákat, a megadott pontban lévő pixel színét kapjuk eredményül.

```
  co:=c.Pixels[cur.X, cur.Y];
```

Panel2 háttérszínét beállítjuk erre a színre, így nem csak szövegesen olvasható a szín értéke, de meg is tekinthető a Panel2 által.

```
  Panel2.Color:=co;
```

Következő lépésként felbontjuk az adott színt RGB összetevőire. Ehhez a GetRValue, GetGValue és a GetBValue függvényt hívjuk meg. Eredményül megkapjuk az R, G és B színösszetevők 0-255 közötti értékeit, melyeket három Label komponens segítségével jelenítünk meg a Form-on.

```
  Label1.Caption:=IntToStr(GetRValue(co));
  Label2.Caption:=IntToStr(GetGValue(co));
  Label3.Caption:=IntToStr(GetBValue(co));
```

Most hexadecimális kódokkal is megjelenítjük ezt a szín értéket, annak megkönnyítésére, hogy az közvetlenül felhasználható legyen HTML kódban. Ezt az értéket szintén egy Label komponens segítségével jelenítjük meg a Form-on.

```
  Label4.Caption:='#'+IntToHex(GetRValue(
    co), 2)+IntToHex(GetGValue(co),
    2)+IntToHex(GetBValue(co), 2);
```

Utolsóként megjelenítendő adatunk a Form-on az egér pillanatnyi X, Y koordinátája lesz.

```
  Label5.Caption:=IntToStr(cur.X) + ' - ' +
    IntToStr(cur.Y);
```

Ezzel a szín értékeit feldolgoztuk, így a lefoglalt erőforrásokat felszabadíthatjuk.

```
  ReleaseDC(GetDeskTopWindow, c.Handle);
  c.Free;
```

Következhet az egér által kijelölt terület nagyított megjelenítése. A nagyítás mértékét a zoom változó tárolja. Ezt a számot átírva megváltoztathatjuk azt, hogy a program milyen mértékben nagyítsa fel a képernyő képet, mielőtt azt megjeleníti a Form-on.

```
  zoom:= 4;
```

A kirajzolás előtt ellenőrizzük, hogy az alkalmazás nincs minimalizálva.

```
If not IsIconic(Application.Handle)
  then begin
```

Ha nincs, lekérjük a képernyő eléréséhez szükséges leírót.

```
    hDesktop:= GetDesktopWindow;
```

Változókba tároljuk a Form-on lévő Image komponens szélességét, magasságát.

```
    iWidth:=Image1.Width;
    iHeight:=Image1.Height;
```

Ezeket felhasználva készítünk egy TRect típusú változót, melyet a kép másolásánál használunk majd fel.

```
    Drect:=Rect(0,0,iWidth,iHeight);
    iTmpX:=iWidth / (zoom * 4);
    iTmpY:=iHeight / (zoom * 4);
    Srect:=Rect(cur.x,cur.y,cur.x,cur.y);
    InflateRect(Srect,Round(
      iTmpX),Round(iTmpY));
```

Ellenőrzést végzünk, hogy az egér nincs-e a képernyő széléhez közel. Ez esetben az Image-en megjelenített képen a középpontot jelző keresztet is mozgatnunk kell.

```
    If Srect.Left<0 then OffsetRect(
```

```
      Srect,-Srect.Left,0);
    If Srect.Top<0 then OffsetRect(
      Srect,0,-Srect.Top);
    If Srect.Right>Screen.Width then
      OffsetRect(Srect,-(Srect.
      Right-Screen.Width),0);
    If Srect.Bottom>Screen.Height then
      OffsetRect(Srect,0,-(Srect.
      Bottom-Screen.Height));
```

Létrehozunk egy TCanvas objektumot, melyet a képernyőre állítunk be. Ez lesz az adatforrás. Az Image1 Canvas objektumán keresztül elérhető CopyRect eljárás segítségével elvégezzük a képernyőkép nagyított másolását.

```
  C:=TCanvas.Create;
  try
    C.Handle:=GetDC(GetDesktopWindow);
    Image1.Canvas.CopyRect(Drect,C,Srect);
```

Végezetül felszabadítjuk a lefoglalt erőforrásokat.

```
  finally
    ReleaseDC(hDesktop, C.Handle);
    C.Free;
  end;
```

Utolsó lépésként kirajzoljuk a nagyított képre azt a szálkeresztet, mely jelöli az egér koordináta pontját.

```
      with Image1.Canvas do begin
        DmX:=zoom * 2 * (cur.X-Srect.Left);
        DmY:=zoom * 2 * (cur.Y-Srect.Top);
        MoveTo(DmX - (iWidth div 4),DmY);
        LineTo(DmX + (iWidth div 4),DmY);
        MoveTo(DmX,DmY - (iHeight div 4));
        LineTo(DmX,DmY + (iHeight div 4));
      end;
    Application.ProcessMessages;
  end;
end;
```

Miksa Előd

USB eszközök csatlakoztatásának figyelése programból

Cikk sorszáma	Szoftverfeltételek
098	➢ Delphi
Forráskód	\Source\Usb\

Manapság már számos olyan eszköz kapható, melyek az USB porton keresztül csatlakoznak számítógépünkhöz. Ezek között sok olyan is található, melyek nem állandó jelleggel kapcsolódnak, hanem csak bizonyos időszakonként. Ilyen például egy digitális fényképezőgép, melyet csak arra az időszakra kapcsolunk a géphez, ameddig új fényképeinket át nem töltjük.

E példában egy olyan kis programot készítünk, mely képes figyelni és érzékelni azt a helyzetet, ha a számítógép bármely USB portjához csatlakoztat a felhasználó bármilyen eszközt, illetve eltávolítja azt.

WM_DEVICECHANGE üzenet

Az USB eszközök csatlakoztatásának, eltávolításának eseményéhez a WM_DEVICECHANGE üzenet érkezését kell figyelnünk.

Az üzenetet wParam paraméter értéke árulja el számunkra, hogy milyen esemény történt a rendszerben. Ez az alábbiak egyike lehet:

- 0x0019 - DBT_CONFIGCHANGECANCELED
- 0x0018 - DBT_CONFIGCHANGED
- 0x8006 - DBT_CUSTOMEVENT
- 0x8000 - DBT_DEVICEARRIVAL
- 0x8001 - DBT_DEVICEQUERYREMOVE
- 0x8002 - DBT_DEVICEQUERYREMOVEFAILED
- 0x8004 - DBT_DEVICEREMOVECOMPLETE
- 0x8003 - DBT_DEVICEREMOVEPENDING
- 0x8005 - DBT_DEVICETYPESPECIFIC
- 0x0007 - DBT_DEVNODES_CHANGED
- 0x0017 - DBT_QUERYCHANGECONFIG
- 0xFFFF - DBT_USERDEFINED

Az üzenet lParam paraméter a bekövetkezett eseménytől függő adatstruktúrát adja meg szá-

munkra, melyből kiolvashatók a további információk.

Amennyiben az üzenet visszatérési értékének igaz értéket adunk, abban az esetben engedélyeztük a művelet végrehajtását. Ha azonban BROADCAST_QUERY_DENY értéket adunk visszatérési értékként, akkor azzal a folyamatban lévő műveletet megszakítjuk.

Példaprogram elkészítése

Üzenet megjelenése a Form-on új eszköz érkezésekor

Az USB eszközök érzékeléséhez szükségünk lesz két adatstruktúrára. Első lépésként ezeket deklaráljuk:

```
type
  PDevBroadcastHdr  = ^DEV_BROADCAST_HDR;
  DEV_BROADCAST_HDR = packed record
    dbch_size: DWORD;
    dbch_devicetype: DWORD;
    dbch_reserved: DWORD;
  end;
  PDevBroadcastDeviceInterface
     = ^DEV_BROADCAST_DEVICEINTERFACE;
  DEV_BROADCAST_DEVICEINTERFACE = record
    dbcc_size: DWORD;
    dbcc_devicetype: DWORD;
    dbcc_reserved: DWORD;
    dbcc_classguid: TGUID;
    dbcc_name: short;
  end;
```

Ezt követően létrehozzuk a szükséges konstans értékeket.

```
const
  GUID_DEVINTERFACE_USB_DEVICE: TGUID
    = '{A5DCBF10-6530-11D2-901F-
00C04FB951ED}';
  DBT_DEVICEARRIVAL            = $8000;
  DBT_DEVICEREMOVECOMPLETE     = $8004;
  DBT_DEVTYP_DEVICEINTERFACE = $00000005;
```

A Form1-ben készítünk egy üzenetkezelő eljárást, mely akkor aktivizálódik, ha a Form a WM_DEVICECHANGE üzenetet kapja.

```
type
 TForm1 = class(TForm)
   Label1: TLabel;
   procedure FormCreate(Sender: TObject);
 private
 protected
  procedure WM_DEVICECHANGE(
     var Msg: TMessage);
     message WM_DEVICECHANGE;
 public
    { Public declarations }
 end;
```

Form OnCreate esemény

A Form OnCreate eseményénél regisztráljuk az ablakunkat a Windows-nál olyan célból, hogy értesítést kérünk abban az esetben, ha új eszköz csatlakozik a géphez. E kérelem nélkül a WM_DEVICECHANGE üzenetet nem kapná meg programunk.

```
procedure TForm1.FormCreate(
  Sender: TObject);
var
  dbi: DEV_BROADCAST_DEVICEINTERFACE;
  Size: Integer;
  r: Pointer;
begin
```

A kérelem elküldéséhez a RegisterDeviceNotification függvényt kell meghívnunk. A függvény első paraméterében az ablak leíróját kell átadni, a következőben egy feltöltött DEV_BROADCAST_DEVICEINTERFACE adatstruktúrát, melyben leírjuk, hogy milyen eszközök csatlakoztatásakor kérünk értesítést. Ez természetesen most az USB eszközök kategóriája lesz.

```
  Size :=
     SizeOf(DEV_BROADCAST_DEVICEINTERFACE);
  ZeroMemory(@dbi, Size);
  dbi.dbcc_size := Size;
  dbi.dbcc_devicetype
    := DBT_DEVTYP_DEVICEINTERFACE;
  dbi.dbcc_reserved := 0;
  dbi.dbcc_classguid
     := GUID_DEVINTERFACE_USB_DEVICE;
  dbi.dbcc_name := 0;
  r := RegisterDeviceNotification(Handle,
    @dbi, DEVICE_NOTIFY_WINDOW_HANDLE);
end;
```

WM_DEVICECHANGE üzenet

Amikor a WM_DEVICECHANGE üzenetet megkapja alkalmazásunk, akkor biztosak lehetünk abban, hogy egy új eszköz lett csatlakoztatva, vagy épp egy meglévő eltávolítva.

```
procedure TForm1.WM_DEVICECHANGE(var
  Msg: TMessage);
var
  devType: Integer;
  Datos: PDevBroadcastHdr;
begin
```

Elsőként ellenőrizzük, hogy az általunk várt események egyike következett-e be.

```
  if (Msg.wParam = DBT_DEVICEARRIVAL) or
```

```
    (Msg.wParam =
    DBT_DEVICEREMOVECOMPLETE) then
  begin
```

Lekérdezzük az események adatait.

```
    Datos := PDevBroadcastHdr(Msg.lParam);
    devType := Datos^.dbch_devicetype;
```

Majd vizsgáljuk, hogy az érintett eszköz éppen csatlakoztatva, vagy eltávolítva lett-e. Ennek megfelelő üzenetet jelenítünk meg a Form-on.

```
    if devType =
      DBT_DEVTYP_DEVICEINTERFACE then
    begin
      if Msg.wParam = DBT_DEVICEARRIVAL
then
      begin
          Label1.Caption:='usb
            eszköz csatlakoztatva';
      end
      else
      begin
          Label1.Caption:='usb
            eszköz eltávolítva';
      end;
    end;
  end;
end;
```

Pálfalvi Péter

Kapcsolódó webhelyek

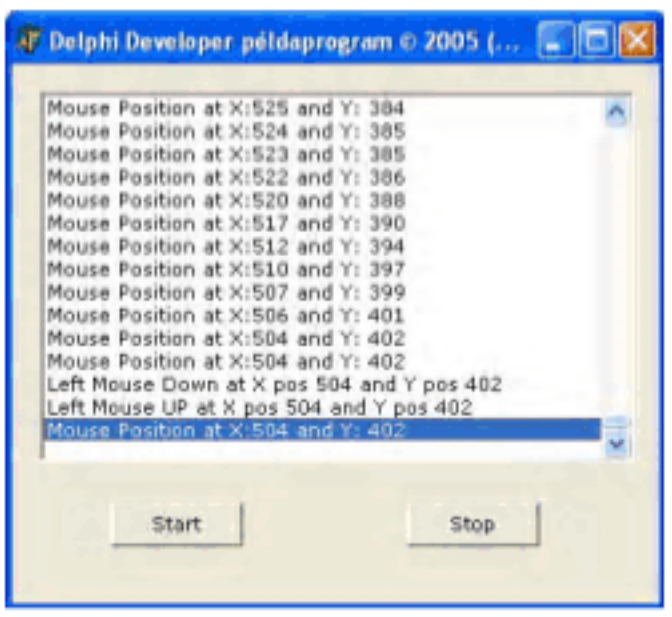
WM_DEVICECHANGE ↳ 021

Egér műveletek követése

Cikk sorszáma	Szoftverfeltételek
099	➢ Delphi
Forráskód	\Source\Trap\

Készítünk most egy olyan alkalmazást, melynek segítségével bármilyen egér műveletet nyomon lehet követni és naplózni még akkor is, ha éppen egy másik alkalmazással dolgozik a felhasználó.

A példaprogram



A példaprogram futás közben

A nyomkövetés indítása

A Start gombra kattintva indíthatjuk el az egér események nyomkövetését.

```
procedure TForm1.Button1Click(
  Sender: TObject);
begin
```

Első lépésként ellenőrizzük, hogy e folyamat nem fut-e még. Ezt egy globális változó értékének ellenőrzésével tehetjük meg.

```
  if not FHookStarted then
  begin
```

Ha még nem lett elindítva a folyamat, akkor meghívjuk a SetWindowsHookEx függvényt.

Ennek paraméterként megadunk egy JournalProc nevű függvényt. Amikor felhasználói input következik be, akkor aktivizálva lesz a JournalProc nevű függvényünk. Ebben vizsgálhatjuk, hogy milyen egér művelet történt.

A SetWindowsHookEx függvény visszatérési értékként ad egy azonosító számot, melyet globális változóba tárolunk.

```
    JHook := SetWindowsHookEx(
      WH_JOURNALRECORD, @JournalProc,
      hInstance, 0);
```

Amennyiben a művelet indítása sikeres volt, úgy átállítjuk a változónkat igaz értékre, mely jelzi, hogy a folyamat fut.

```
    if JHook > 0 then
    begin
      FHookStarted := True;
    end
  end;
end;
```

JournalProc függvény

A JournalProc függvényben, mielőbb bármit is teszünk, meg kell hívnunk a CallNextHookEx függvényt, mely továbbadja a kapott eseményt más alkalmazások számára.

```
function JournalProc(Code, wParam: Integer;
  var EventStrut: TEventMsg):
  Integer; stdcall;
var
  Char1: PChar;
  s: string;
begin
  Result := CallNextHookEx(JHook, Code,
    wParam, Longint(@EventStrut));
  if Code < 0 then Exit;
  if Code = HC_SYSMODALON then Exit;
  if Code = HC_ACTION then
  begin
    {
```

A kapott üzenet adatai alapján ellenőrizzük, hogy milyen esemény következett be. WM_LBUTTONUP esemény az egér bal gombjának felengedésekor keletkezik.

```
    s := '';
    if EventStrut.message =
      WM_LBUTTONUP then
      s := 'Left Mouse UP at X pos ' +
        IntToStr(EventStrut.paramL) + ' and
          Y pos ' + IntToStr(
          EventStrut.paramH);
```

WM_ LBUTTONDOWN esemény az egér bal gombjának lenyomásakor keletkezik.

```
    if EventStrut.message =
      WM_LBUTTONDOWN then
      s := 'Left Mouse Down at X pos ' +
        IntToStr(EventStrut.paramL) + ' and
          Y pos ' + IntToStr(
          EventStrut.paramH);
```

WM_RBUTTONDOWN esemény az egér jobb gombjának lenyomásakor keletkezik.

```
    if EventStrut.message =
      WM_RBUTTONDOWN then
      s := 'Right Mouse Down at X pos ' +
        IntToStr(EventStrut.paramL) + ' and
          Y pos ' + IntToStr(
          EventStrut.paramH);
```

WM_RBUTTONUP esemény az egér jobb gombjának felengedésekor keletkezik.

```
    if (EventStrut.message =
      WM_RBUTTONUP) then
      s := 'Right Mouse Up at X pos ' +
        IntToStr(EventStrut.paramL) + ' and
          Y pos ' + IntToStr(
          EventStrut.paramH);
```

WM_MOUSEWHEEL esemény az egér görgőjének mozgatásakor keletkezik.

```
    if (EventStrut.message =
      WM_MOUSEWHEEL) then
      s := 'Mouse Wheel at X pos ' +
        IntToStr(EventStrut.paramL) + ' and
          Y pos ' + IntToStr(
          EventStrut.paramH);
```

WM_MOUSEMOVE esemény az egér mozgatásakor keletkezik.

```
    if (EventStrut.message =
      WM_MOUSEMOVE) then
      s := 'Mouse Position at X:' +
        IntToStr(EventStrut.paramL) + ' and
          Y: ' +
IntToStr(EventStrut.paramH);
    if s <> '' then
       Form1.ListBox1.ItemIndex :=  Form1.
         ListBox1.Items.Add(s);
  end;
end;
```

ApplicationEvents komponens OnMessage esemény

Amikor Ctrl - Alt - Del billentyű kombinációt, vagy Ctrl - Esc billentyű kombinációt nyom le a felhasználó, akkor alkalmazásunk egér esemény figyelése megszakad. Ilyen esetben újra kell indítanunk a folyamatot, ha az már futott. Erről a tényről egy WM_CANCELJOURNAL eseményből értesülhetünk. Az esemény figyelésére egy ApplicationEvents komponenst helyezünk a Form-ra. Ennek OnMessage eseményét felhasználva minden beérkező üzenetet ellenőrizhetünk.

```
procedure TForm1.ApplicationEvents1Message(
  var Msg: tagMSG;
  var Handled: Boolean);
begin
```

Ha a beérkező üzenet a WM_CANCELJOURNAL és a figyelés folyamata már el lett indítva, akkor a SetWindowsHookEx függvény hívásával újraindítjuk a megszakadt folyamatot.

```
  Handled := False;
  if (Msg.message = WM_CANCELJOURNAL) and
    FHookStarted then
    JHook := SetWindowsHookEx(
      WH_JOURNALRECORD, @JournalProc, 0,
0);
end;
```

Figyelés leállítása

Amikor a Stop gombra kattintunk, akkor a már elindított figyelést le kell állítania alkalmazásunknak. Ehhez nincs más teendőnk, mint meghívni az UnhookWindowsHookEx függvényt, melynek paraméterként azt az azonosítót kell megadnunk, amelyet a SetWindowsHookEx függvény adott visszatérési értékként.

```
procedure TForm1.Button2Click(
  Sender: TObject);
begin
  FHookStarted := False;
  UnhookWindowsHookEx(JHook);
  JHook := 0;
end;
```

Form OnClose esemény

Programunk futásának végén, amikor a felhasználó bezárja a Form-ot, akkor fut le az OnClose esemény. Ekkor ellenőriznünk kell, hogy a figyelési folyamat el lett-e már indítva. Ha igen, akkor le kell állítanunk az UnhookWindowsHookEx függvény használatával.

```
procedure TForm1.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  if FHookStarted then
    UnhookWindowsHookEx(JHook);
end;
```

Tália Gábor

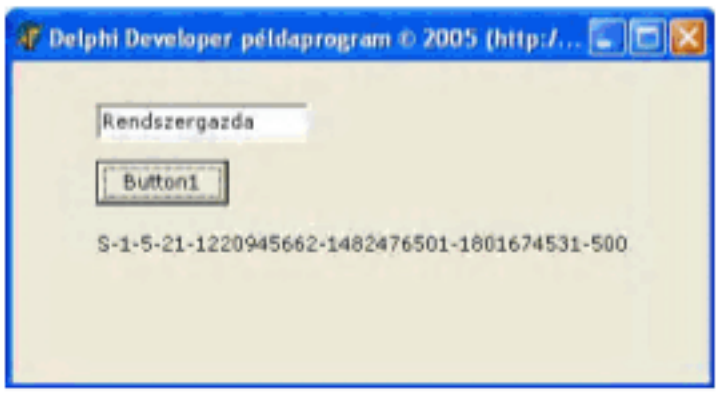
Kapcsolódó webhelyek

SetWindowsHookEx ↳ 022

Felhasználók egyedi azonosítója

Cikk sorszáma	Szoftverfeltételek
100	➢ Delphi ➢ Windows 2000
Forráskód	\Source\Sid\

Az operációs rendszer minden felhasználóhoz egy egyedi azonosítót (SID) rendel hozzá. Ezt az azonosítót mi magunk is lekérdezhetjük és tetszés szerint felhasználhatjuk saját alkalmazásainkban.



Egyedi azonosító megjelenítése

Az egyedi azonosítót a LookupAccountName függvény hívásával tudjuk lekérdezni. Amennyiben ez sikeresen megtörténik, a kapott értéket a ConvertSidToStringSidA függvénnyel tudjuk szövegesen olvashatóvá konvertálni.

A ConvertSidToStringSidA függvény az ADVAPI32.DLL-ben található. Ezt külön kell deklarálnunk, a további függvények deklarációja már rendelkezésre áll a Delphi unit-jaiban.

```
  function ConvertSidToStringSidA(sid:
PSID;
    var StringSid: LPSTR): BOOL; stdcall;
    external 'ADVAPI32.DLL';
```

Button1 OnClick

A Form-on lévő Edit1-be adjunk meg szövegesen egy felhasználói nevet, majd kattintsunk a Button1 gombra. Ekkor a program meghatározza a megadott felhasználóhoz tartozó egyedi azonosító számot és megjeleníti egy Label-en.

```
procedure TForm1.Button1Click(
  Sender: TObject);
const
  SID_REVISION=1;
var
  buf: array [0..MAX_PATH - 1] of char;
  p: PAnsiChar;
  domain: array [0..MAX_PATH - 1] of char;
  snu: SID_NAME_USE;
  sid: PSid;
  a, b: dword;
begin
  Label1.Caption:='';
```

A lekérdezés előtt ellenőrizzük, hogy az adott operációs rendszer verziója megfelelő-e a szükséges függvények futtatásához.

```
  if (Win32Platform=VER_PLATFORM_WIN32_NT)
    and (Win32MajorVersion>=5) then begin
    a:=MAX_PATH;
    b:=0;
    sid:=nil;
```

Elsőként nulla bájtokkal töltjük fel a domain struktúrát.

```
FillChar(domain, SizeOf(domain), 0);
```

Majd a LookupAccountName függvény hívásával lekérdezzük, hogy mekkora memória területre lesz szükség a SID tárolásához. Abban az esetben,ha a sid változó értéke nil, akkor nem történik meg a feltöltése adatokkal, viszont eredményül kapjuk a szükséges bájtok számát.

```
LookupAccountName(nil, PChar(Edit1.
  Text), sid, b, domain, a, snu);
```

Most, hogy már ismert a szükséges méret, az AllocMem függvénnyel képesek vagyunk memóriát foglalni a kívánt mennyiségben.

```
sid:=AllocMem(b);
```

Ismét meghívjuk a LookupAccountName függvényt, de most már van memóriaterület foglalva, így a szükséges adatokat a sid változóba visszakapjuk.

```
LookupAccountName(nil, PChar(Edit1.
  Text), sid, b, domain, a, snu);
```

Az adatok helyességét ellenőrizhetjük az IsValidSid függvény segítségével.

```
if IsValidSid(sid) then begin
```

Most már csak a szöveges átalakítás marad hátra, melyhez a ConvertSidToStringSidA függvényt hívjuk meg, az eredményt pedig megjelenítjük a Label1-en.

```
      FillChar(buf, SizeOf(buf), 0);
      if ConvertSidToStringSidA (sid, p)
        then begin
        Label1.Caption:=p;
      end;
      LocalFree(LongWord(p));
    end;
    FreeMem (sid, b);
  end;
end;
```

Győri Béla

Kapcsolódó webhelyek

ConvertSidToStringSid ↳ 023

Szolgáltatás státuszának lekérdezése

Cikk sorszáma	Szoftverfeltételek
101	➢ Delphi ➢ Microsoft SQL
Forráskód	\Source\Service\

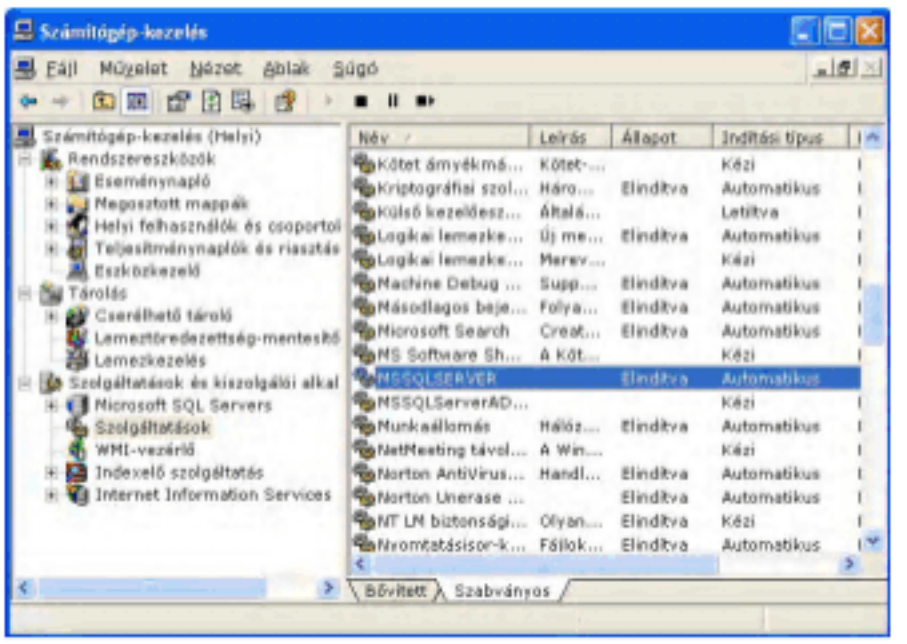
Egy számítógépen számos szolgáltatás, más néven szerviz alkalmazás fut, vagy éppenséggel nem fut. Hogy mi is a pontos státusza ezeknek az alkalmazásoknak, az igen fontos lehet alkalmazásaink számára, hiszen sok esetben a működőképességük függhet ezektől.

Például, ha alkalmazásunk SQL adatbázist használ és az adott számítógépen valamilyen okból kifolyólag nem fut az SQL szerver szolgáltatás, akkor alkalmazásunk sem tud működni céljának megfelelően.

Ha nem készítjük fel programunkat alaposan, akkor bizony súlyos működési hibába futhatunk. Ezért célszerű, hogy alkalmazásunk indulásakor az első SQL szerveres művelet előtt ellenőrizzük, hogy fut-e a szükséges szolgáltatás.

A mellékelt példaprogramban bármely szolgáltatás aktuális állapotának ellenőrizésére alkalmas eljárást készítünk.

Szolgáltatások manuális ellenőrzése



Számítógép szolgáltatásai

Az egyes szolgáltatások aktuális állapotát ellenőrizhetjük manuálisan is. Ehhez indítsuk el a Vezérlőpult - Felügyeleti eszközök - Szolgáltatások, vagy a Számítógép-kezelés nevű alkalmazását. A Szolgáltatások kategórián belül látható az adott számítógép összes szolgáltatása és aktuális állapotuk. E segédprogram használatával akár le is állíthatunk, el is indíthatunk szolgáltatásokat. Nézzük azonban, miként tehetjük meg ezt programból.

Elméleti alapok

Egy szolgáltatás aktuális állapotának meghatározása saját programból, feltételez néhány függvény ismeretet. Kezdjük most ezek áttekintésé-

vel, mielőtt hozzálátnánk a gyakorlati megvalósításhoz.

OpenSCManager függvény

Először is el kell érnünk a szolgáltatás menedzsert. Ehhez az OpenSCManager függvényt kell meghívnunk.

A függvény deklarációja:

```
SC_HANDLE OpenSCManager(
  LPCTSTR lpMachineName,
  LPCTSTR lpDatabaseName,
  DWORD dwDesiredAccess
);
```

Paraméter

- lpMachineName

Annak a számítógépnek a neve, amelyiken futó szolgáltatáskezelőt meg kívánjuk nyitni. Ha NULL értéket adunk meg, akkor a helyi számítógép szolgáltatáskezelője kerül megnyitásra.

- lpDatabaseName

A szolgáltatáskezelő által megnyitandó adatbázis. Ha NULL értéket adunk meg, akkor a SERVICES_ACTIVE_DATABASE adatbázis kerül megnyitásra, mely az aktív szolgáltatásokat tartalmazza.

- dwDesiredAccess

A hozzáférési jogok megadására szolgál a dwDesiredAccess paraméter. Általában a SC_MANAGER_CONNECT konstans használatos.

Visszatérési érték

Ha a függvény sikeresen lefut, akkor a szolgáltatáskezelő leíróját kapjuk meg. Ellenkező esetben NULL értéket.

OpenService függvény

Egy konkrét szolgáltatás megnyitásához szükséges az OpenService függvény.

A függvény deklarációja:

```
SC_HANDLE OpenService(
  SC_HANDLE hSCManager,
  LPCTSTR lpServiceName,
  DWORD dwDesiredAccess
);
```

Paraméterek

- hSCManager

A szolgáltatáskezelő leírója, melyet az OpenSCManager függvény adott vissza.

- lpServiceName

A megnyitandó szolgáltatás megnevezése.

- dwDesiredAccess

Hozzáférési jogok.

Visszatérési érték

Sikeres megnyitás esetén a szolgáltatás leírója, különben NULL érték.

QueryServiceStatus függvény

Egy megnyitott szolgáltatás státuszát kérdezhetjük le a QueryServiceStatus függvény által.

A függvény deklarációja:

```
BOOL QueryServiceStatus(
  SC_HANDLE hService,
  LPSERVICE_STATUS lpServiceStatus
);
```

Paraméterek

- hService

A megnyitott szolgáltatás leírója, melyet az OpenService, illetve a CreateService függvény ad vissza.

- lpServiceStatus

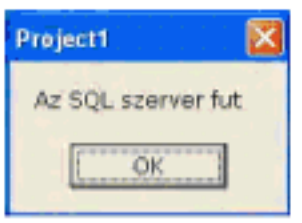
SERVICE_STATUS struktúra, melybe a lekérdezés eredményét kapjuk meg. A szolgáltatás aktuális státuszát a dwCurrentState értékéből tudjuk meg. Ennek értéke az alábbiak egyike lehet:

1 = SERVICE_STOPPED
2 = SERVICE_START_PENDING
3 = SERVICE_STOP_PENDING
4 = SERVICE_RUNNING
5 = SERVICE_CONTINUE_PENDING
6 = SERVICE_PAUSE_PENDING
7 = SERVICE_PAUSED

Visszatérési érték

A függvény igaz értékkel tér vissza, ha a lekérdezés művelete sikeres volt.

Példaprogram elkészítése



Futó példaprogram

A mellékelt példaprogramban a Microsoft SQL szerver szolgáltatás státuszát kérdezzük le gyakorlati példaként. Ehhez készítünk egy olyan függvényt, mellyel bármely szolgáltatás aktuális állapota lekérdezhető. E függvény a ServiceGetStatus lesz.

ServiceGetStatus függvény

E függvény első paramétereként annak a számítógépnek a nevét kell megadnunk, amelyen ellenőrizni szeretnénk, hogy fut-e a második paraméterben megadott nevű szolgáltatás.

```
function ServiceGetStatus(sMachine,
  sService: PChar): DWORD;
var
  SCManHandle, SvcHandle: SC_Handle;
  SS: TServiceStatus;
  dwStat: DWORD;
begin
  dwStat := 0;
```

Elsőként megnyitjuk a szolgáltatáskezelőt, a megadott számítógépen.

```
  SCManHandle := OpenSCManager(sMachine,
    nil, SC_MANAGER_CONNECT);
```

Ha ez sikeres volt, akkor léphetünk tovább.

```
  if (SCManHandle > 0) then
  begin
```

Következőként megnyitjuk a tesztelendő szolgáltatást.

```
    SvcHandle := OpenService(SCManHandle,
      sService, SERVICE_QUERY_STATUS);
```

Ha ez is sikerült, akkor következhet a státusz lekérdezés.

```
    if (SvcHandle > 0) then
    begin
```

Az aktuális státuszt a QueryServiceStatus függvény hívásával tudhatjuk meg.

```
      if (QueryServiceStatus(SvcHandle,
        SS)) then
```

Sikeres hívás esetén a TServiceStatus dwCurrentState értékéből tudjuk meg a kiválasztott státuszt.

```
        dwStat := ss.dwCurrentState;
```

Lekérdezés után zárjuk a szolgáltatást.

```
      CloseServiceHandle(SvcHandle);
    end;
```

Végül zárjuk a szolgáltatáskezelőt is.

```
    CloseServiceHandle(SCManHandle);
  end;
  Result := dwStat;
end;
```

ServiceRunning függvény

Készítünk egy másik függvényt is ServiceRunning néven, mely kifejezetten arra szolgál, hogy egy szolgáltatás elérhetőségét vizsgáljuk. A függvény szintén a számítógép és a szolgáltatás nevét várja. Visszatérési értékként igazat ad, ha a megadott szolgáltatás fut a kiválasztott számítógépen.

```
function ServiceRunning(sMachine, sService:
  PChar): Boolean;
begin
```

A függvény működése igen egyszerű, hiszen csak meg kell hívnunk a ServiceGetStatus függvényünket a kapott paraméterekkel és ellenőriznünk kell, hogy a visszaadott eredmény egyezike a SERVICE_RUNNING értékkel. Ha ez teljesül, akkor a vizsgált szolgáltatás fut.

```
  Result := SERVICE_RUNNING =
    ServiceGetStatus(sMachine, sService);
end;
```

Button1 OnClick függvény

A Button1-re történő kattintáskor ellenőrizzük, hogy a helyi gépen fut-e az Microsoft SQL szerver szolgáltatás vagy sem.

```
procedure TForm1.Button1Click(
  Sender: TObject);
begin
  if ServiceRunning(nil, 'MSSQLSERVER')
then
    ShowMessage('Az SQL szerver fut')
  else
    ShowMessage('Az SQL szerver nem fut')
end;
```

Katus Gábor

Kapcsolódó webhelyek

Service Functions ↳ 015

Átkapcsolás programok között programból

Cikk sorszáma	Szoftverfeltételek
102	➢ Delphi
Forráskód	\Source\Programs\

Programból kérdezzük le, hogy milyen más alkalmazások futnak jelenleg a számítógépen és megvalósítjuk azt is, hogy ezek között átkapcsolhasson programunk, mintha csak az Alt - Tab billentyű kombinációt nyomtuk volna le.

A feladat megvalósításának első lépése az éppen futó programok ablakainak lekérdezése, melyhez az EnumWindows függvényt hívjuk segítségül. E függvény deklarációja:

```
BOOL EnumWindows(
    WNDENUMPROC lpEnumFunc,
    LPARAM lParam
);
```

Paraméterek

- lpEnumFunc

Első paraméterként egy callback függvényt kell megadni. Az EnumWindows függvény annyiszor hívja meg az itt megadott függvényünket, ahány futó programot talál. A függvényben további adatokat olvashatunk ki az adott programokról.

- lParam

A callback függvény megkapja paraméterként az lParam-ban megadott értéket is. Ezt felhasználva tetszőleges adatot adhatunk át a callback függvényünknek szükség esetén.

Visszatérési érték

Ha a függvény futása sikeres, akkor a visszatérési értéke igaz lesz.

A példaprogram megvalósítása

Form OnCreate esemény

Amikor az alkalmazásunk indul, akkor a Form OnCreate eseményénél töröljük a ListBox1 tartalmát, ahová a futó alkalmazásokat gyűjtjük ki, majd az EnumWindows függvény hívásával feltöltjük. A függvénynek az EnumWindowsProc függvényünk címét adjuk át. Ezt hívja majd annyiszor, ahány futó program van. A találatok között csak azok a programok szerepelnek majd, melyek rendelkeznek a felhasználó számára is elérhető ablakkal.

```
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  Listbox1.Clear;
  EnumWindows(@EnumWindowsProc, 1);
end;
```

EnumWindowsProc függvény

Ha az EnumWindowsProc meghívásra kerül, akkor a HWND típusú paramétere az aktuálisan megtalált futó program fő ablakának a leírója lesz.

```
function EnumWindowsProc(Wnd: HWND; lParam:
  lParam): BOOL; stdcall;
var
  buffer: array[0..200] of Char;
begin
```

Mielőtt a megtalált ablakot az eredménylistához adnánk, végzünk némi szűrést: nem adjuk az eredményhez azokat az ablakokat, melyeknek láthatósága le van tiltva. Kimarad az az ablak is, mely minimalizálva van, vagy Tool Window típusú.

```
  if (IsWindowVisible(Wnd) or IsIconic(
    wnd)) and
    ((GetWindowLong(Wnd, GWL_HWNDPARENT) =
      0) or
    (GetWindowLong(Wnd, GWL_HWNDPARENT) =
      GetDesktopWindow)) and
    (GetWindowLong(Wnd, GWL_EXSTYLE) and
      WS_EX_TOOLWINDOW = 0) then
  begin
```

Ha az adott ablak a szűrésen túl jut, akkor a GetWindowText függvénnyel lekérdezzük az ablak fejlécének szövegét, és ezt rendeljük hozzá az eredmény listához abban az esetben.

```
    GetWindowText(Wnd, buffer, 256);
    if buffer <> 'GDI+ Window' then
      Form1.Listbox1.Items.Append(buffer);
  end;
end;
```

Timer1 OnTimer esemény

A programok közötti átkacsolást a Timer1 komponensre bízzuk. Amikor ennek OnTimer eseménye bekövetkezik, akkor hívjuk meg a Step eljárást, mely a következő alkalmazásra lép.

```
procedure TForm1.Timer1Timer(
  Sender: TObject);
begin
    Step;
end;
```

Step eljárás

Az aktuális programot egy Index nevű globális változó tartja nyilván. Ennek segítségével lépkedünk végig a ListBox elemein.

```
procedure TFom1.Step;
var
  hWnd: DWORD;
begin
  if Index < ListBox1.Items.Count - 1 then
    Inc(Index)
  else
    Index := 0;
```

A FindWindow függvény segítségével név alapján keressük elő az adott ablakot. A nevet a ListBox-ból vesszük.

```
  hWnd := FindWindow(nil, PChar(
    Listbox1.Items[Index]));
```

Ha a FindWindows sikeresen lefut, akkor gondoskodunk arról, hogy a soron következő ablak legyen az aktív.

```
  if hWnd <> 0 then
  begin
    ShowWindow(hwnd, SW_MAXIMIZE);
    SetActiveWindow(hwnd);
    SetForegroundWindow(hWnd);
    windows.SetFocus(hWnd);
  end;
end;
```

Károlyi Péter

Kapcsolódó webhelyek

EnumWindows ↳014

Hálózati információk gyűjtése

Cikk sorszáma	Szoftverfeltételek
103	➢ Delphi
Forráskód	\Source\Net\

E példaprogram segítségével a helyi hálózat egy adott gépéről gyűjtünk programból információkat. Többek között meghatározzuk az adott számítógépen lévő operációs rendszer típusát, lekérdezzük a megosztott mappákat, felderítjük a hálózaton lévő számítógépeket és SQL szerver kiszolgálókat.

A program használata

A példaprogram futás közben

A mellékelt példaprogram használata egyszerű: az Edit1-be írja be az elérni kívánt számítógép nevét, majd kattintson a Button1-re. Az adatok a ListBox-ban jelenek meg.

Elméleti áttekintés

A program megvalósításhoz szükségünk lesz a NetShareEnum, NetShareGetInfo, NetApiBufferFree, NetServerGetInfo és NetServerEnum függvények ismeretére.

NetShareEnum függvény

A NetShareEnum függvénnyel a hálózati megosztásokat kérdezhetjük le programból.

A függvény deklarációja az alábbi:

```
NET_API_STATUS NetShareEnum(
  LPWSTR servername,
  DWORD level,
  LPBYTE* bufptr,
  DWORD prefmaxlen,
  LPDWORD entriesread,
  LPDWORD totalentries,
  LPDWORD resume_handle
);
```

Paraméterek

- servername

A lekérdezendő számítógép DNS, vagy NetBIOS neve. Ha e paraméter NULL, akkor a helyi számítógép hálózati megosztásai lesznek lekérdezve.

Windows NT esetén a számítógép nevet kötelezően \\ karakterekkel kell kezdeni.

- level

A lekérdezendő információ adatstruktúrájának kódja. Értéke az alábbiak egyike lehet:

0 - A függvény a hálózati megosztások neveivel tér vissza.

A bufptr paraméter SHARE_INFO_0 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

1 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_1 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén a pbBuffer paraméter SHARE_INFO_1 elemeket tartalmazó tömbre kell mutatnia.

2 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó jogokat, aktuális kapcsolatok számát is.

A bufptr paraméter SHARE_INFO_2 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

50 - Windows Me/98/95 esetén visszaadja a hálózati erőforrásokat név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_50 struktúrát kell, hogy tartalmazzon.

502 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó jogokat, aktuális kapcsolatok számát és egyéb ezekhez kapcsolódó információkat is.

A bufptr paraméter SHARE_INFO_502 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

- bufptr

A lekérdezés eredményét e paraméterben kapjuk vissza. A level paraméter alapján meghatározott struktúrát kell itt megadnunk.

- prefmaxlen

A bufferben rendelkezésre álló hely bájtban. Az ajánlott méretet a MAX_PREFERRED_LENGTH konstans adja.

- entriesread

A függvény hívásakor aktuálisan visszaadott hálózati megosztás információk darabszáma.

- totalentries

Az összes hálózati megosztás információk darabszáma.

- resume_handle

Ha az első kereséssel nem sikerült minden adatot lekérdezni, akkor folytathatjuk a keresést egy következő függvény hívással. Ekkor első híváskor nullát, második híváskor már egyet írjunk a resume_handle paraméterbe.

Visszatérési érték

Ha a függvény sikeresen végrehajtja a lekérdezést, akkor NERR_Success értéket ad vissza, különben a hibakódot.

NetShareGetInfo függvény

A NetShareGetInfo függvény részletes információkkal szolgál a megosztott hálózati erőforrásokról.

A függvény deklarációja az alábbi:

```
NET_API_STATUS NetShareGetInfo(
  LPWSTR servername,
  LPWSTR netname,
  DWORD level,
  LPBYTE* bufptr
);
```

Paraméterek

- servername

A lekérdezendő számítógép DNS, vagy NetBIOS neve. Ha e paraméter NULL, akkor a helyi számítógép hálózati megosztásai lesznek lekérdezve.

Windows NT esetén a számítógép nevet kötelezően \\ karakterekkel kell kezdeni.

- netname

A hálózati erőforrás neve, melyről információt kérünk.

- level

A lekérdezendő információ adatstruktúrájának kódja. Értéke az alábbiak egyike lehet:

0 - A függvény a hálózati megosztások neveivel tér vissza.

A bufptr paraméter SHARE_INFO_0 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

1 - A függvény a hálózati megosztások neveivel, egyéb tulajdonságaival tér vissza.

A bufptr paraméter SHARE_INFO_1 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

1 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_1 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén a pbBuffer paraméter SHARE_INFO_1 elemeket tartalmazó tömbre kell mutatnia.

2 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó jogokat, aktuális kapcsolatok számát is.

A bufptr paraméter SHARE_INFO_2 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

50 - Windows Me/98/95 esetén visszaadja a hálózati erőforrásokat név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_50 struktúrát kell, hogy tartalmazzon.

501 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_501 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

502 - A függvény a megosztott hálózati erőforrásokat adja vissza név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó jogokat, aktuális kapcsolatok számát és egyéb ezekhez kapcsolódó információkat is.

A bufptr paraméter SHARE_INFO_502 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

50 - Windows Me/98/95 esetén visszaadja a hálózati erőforrásokat név és típus szerint. Ezen kívül megkapjuk az azokhoz tartozó megjegyzéseket is.

A bufptr paraméter SHARE_INFO_1005 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén ez az érték nem támogatott.

- bufptr

A lekérdezés eredményét e paraméterben kapjuk vissza. A level paraméter alapján meghatározott struktúrát kell itt megadnunk.

Visszatérési érték

Ha a függvény sikeresen végrehajtja a lekérdezést, akkor NERR_Success értéket ad vissza, különben a hibakódot.

NetApiBufferFree függvény

A NetApiBufferFree függvény használatával felszabadíthatjuk a NetApiBufferAllocate függvény által lefoglalt memóriaterületet.

A függvény deklarációja az alábbi:

```
NET_API_STATUS NetApiBufferFree(
  LPVOID Buffer
);
```

Paraméterek

- Buffer

A felszabadítandó memória terület buffere.

Visszatérési érték

Ha a függvény sikeresen végrehajtja a műveletet, akkor NERR_Success értéket ad vissza, különben a hibakódot.

NetServerGetInfo függvény

Egy hálózati gépről szerezhetünk információkat a NetServerGetInfo függvény segítségével.

A függvény deklarációja az alábbi:

```
NET_API_STATUS NetServerGetInfo(
  LPWSTR servername,
  DWORD level,
  LPBYTE* bufptr
);
```

Paraméterek

- servername

A lekérdezendő számítógép DNS, vagy NetBIOS neve. Ha e paraméter NULL, akkor a helyi számítógép hálózati megosztásai lesznek lekérdezve.

Windows NT esetén a számítógép nevet kötelezően \\ karakterekkel kell kezdeni.

- level

A lekérdezendő információ adatstruktúrájának kódja. Értéke az alábbiak egyike lehet:

100 - szerver név és platform információk lekérdezése.

A bufptr paraméter SERVER_INFO_100 struktúrát kell, hogy tartalmazzon.

101 - szerver név és típus információk lekérdezése.

A bufptr paraméter SERVER_INFO_101 struktúrát kell, hogy tartalmazzon.

102 - szerver név és típus információk és egyéb tulajdonságok lekérdezése.

A bufptr paraméter SERVER_INFO_102 struktúrát kell, hogy tartalmazzon.

1 - szerver név és típus információk lekérdezése.

A bufptr paraméter SERVER_INFO_1 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén is használható.

50 - szerver név és típus információk és egyéb tulajdonságok lekérdezése.

A bufptr paraméter SERVER_INFO_50 struktúrát kell, hogy tartalmazzon.

Windows Me/98/95 esetén is használható.

- bufptr

A lekérdezés eredményét e paraméterben kapjuk vissza. A level paraméter alapján meghatározott struktúrát kell itt megadnunk.

Visszatérési érték

Ha a függvény sikeresen végrehajtja a lekérdezést, akkor NERR_Success értéket ad vissza, különben az alábbi hibakód egyikét kapjuk:

ERROR_ACCESS_DENIED - az adott felhasználó nem férhet hozzá a kért információhoz.

ERROR_INVALID_LEVEL - a level paraméter nem elfogadható értéket tartalmaz.

ERROR_INVALID_PARAMETER - a megadott paraméterek nem elfogadható értékeket tartalmaznak.

ERROR_NOT_ENOUGH_MEMORY - nincs elég memória

NetServerEnum függvény

Számítógépek, szerverek listázására különféle szűrő feltételek alapján a NetServerEnum függvény hívásával végezhetők.

A függvény deklarációja az alábbi:

```
NET_API_STATUS NetServerEnum(
  LPCWSTR servername,
  DWORD level,
  LPBYTE* bufptr,
  DWORD prefmaxlen,
  LPDWORD entriesread,
  LPDWORD totalentries,
  DWORD servertype,
  LPCWSTR domain,
  LPDWORD resume_handle
);
```

Paraméterek

- servername

Fentartott. Érteke mindig NULL.

- level

A lekérdezendő információ adatstruktúrájának kódja. Értéke az alábbiak egyike lehet:

100 - számítógép nevek és platform információk.

A bufptr paraméter SERVER_INFO_100 struktúrát kell, hogy tartalmazzon.

101 - számítógép nevek és típus információk lekérdezése.

A bufptr paraméter SERVER_INFO_101 struktúrát kell, hogy tartalmazzon.

- bufptr

A lekérdezés eredményét e paraméterben kapjuk vissza. A level paraméter alapján meghatározott struktúrát kell itt megadnunk.

- prefmaxlen

A bufferben rendelkezésre álló hely bájtban. Az ajánlott méretet a MAX_PREFERRED_LENGTH konstans adja.

- entriesread

A függvény hívásakor aktuálisan visszaadott hálózati megosztás információk darabszáma.

- totalentries

Az összes hálózati megosztás információk darabszáma.

- servertype

Listázandó számítógépek szűrése. Az alábbi értékek kombinálhatók egymással:

SV_TYPE_WORKSTATION - minden munkaállomás

SV_TYPE_SERVER - minden számítógép, melyen szerver szolgáltatás fut

SV_TYPE_SQLSERVER - minden számítógép, melyen Microsoft SQL Server fut

SV_TYPE_DOMAIN_CTRL - elsődleges tartományvezérlők

SV_TYPE_DOMAIN_BAKCTRL - Backup tartományvezérlők

SV_TYPE_TIME_SOURCE - Timesource szolgáltatást futtató szerverek

SV_TYPE_AFP - Apple File Protocol szerverek

SV_TYPE_NOVELL - Novell szerverek

SV_TYPE_DOMAIN_MEMBER - LAN Manager 2.x domain tagok

SV_TYPE_PRINTQ_SERVER - megosztott nyomtatót kínáló számítógépek

SV_TYPE_DIALIN_SERVER - betárcsázható szerverek

SV_TYPE_XENIX_SERVER - Xenix szerverek

SV_TYPE_SERVER_MFPN - Microsoft File and Print for NetWare számítógépek

SV_TYPE_NT - Windows NT munkaállomások és szerverek

SV_TYPE_WFW - Windows for Workgroups-ot futtató szerverek

SV_TYPE_SERVER_NT - nem tartományvezérlő szerverek

SV_TYPE_POTENTIAL_BROWSER - böngészhető szerverek

SV_TYPE_DOMAIN_ENUM - Primary domain számítógépek

SV_TYPE_WINDOWS - Windows 95 vagy későbbi Windows verziót futtató számítógépek

SV_TYPE_ALL - minden szerver

SV_TYPE_TERMINALSERVER - Terminal Server számítógépek

- domain

NetBIOS formában megadott domain név. Ha NULL, akkor az elsődleges domain lesz alapul véve.

- resume_handle

Fenntartott. Értéke mindig 0.

Visszatérési érték

Ha a függvény sikeresen végrehajtja a lekérdezést, akkor NERR_Success értéket ad vissza, különben a hibakódot.

Példaprogram elkészítése

A mellékelt példaprogram elkészítésének első fázisa a netapi32.dll-ben található függvények deklarációja, melyekre programunk épít.

```
  function NetShareEnum(servername:
PWideChar;
    level: DWORD; var buf: Pointer;
prefmaxlen:
    DWORD; var entriesread: DWORD; var
    totalentries: DWORD; var resume_handle:
    DWORD): NET_API_STATUS; stdcall;
external
    'netapi32.dll';
  function NetShareGetInfo(servername:
    PWideChar; netname: PWideChar; level:
    DWORD; var buf: Pointer):
NET API STATUS;
    stdcall; external 'netapi32.dll';
  function NetApiBufferFree(P: Pointer):
    NET_API_STATUS; stdcall;external
    'netapi32.dll';
 function NetServerGetInfo(Server:
   PWideChar; Level: DWord; var
   BufPtr:Pointer): LongInt; stdcall;
   external 'netapi32.dll';
function NetServerEnum(const ServerName:
  PWideString; level: DWORD; var Buffer:
  pointer; PrefMaxLen: DWORD; var
  EntriesRead: DWORD; var TotalEntries:
  DWORD;ServerType: DWORD; const Domain:
  PWideChar; var ResumeHandle: DWORD):
DWORD;
  stdcall; external 'netapi32.dll';
```

Button1 OnClick esemény

A program futtatását követően a Button1-re történő kattintáskor gyűjtjük össze és jelenítjük meg a szükséges információkat. Előtte az Edit1-ben kell megadni annak a számítógépnek a nevét, melyről az információkat kérjük.

```
procedure TForm1.Button1Click(
  Sender: TObject);
var r : string;
begin
```

Elsőként az operációs rendszerről szerzünk információt a megadott számítógépen. E feladatot a GetOSVersion függvényünk végzi.

```
  r := GetOSVersion(Edit1.Text);
  if r <> '' then begin
    ListBox1.Items.add('Operációs rendszer:
      ' + r);
  end else begin
    ListBox1.Items.add('A(z) ' + edit1.Text
      + ' nevû számítógép nem elérhetõ!');
  end;
```

A hálózati megosztásokról a Share eljárás gyűjt információt.

```
  ListBox1.Items.add('_____________');
  ListBox1.Items.add(' - Megosztások:');
  Share;
```

Következőként kilistázzuk azokat a számítógépeket, melyeken van SQL szerver:

```
  ListBox1.Items.add('_____________');
  ListBox1.Items.add(' - SQL szerverek:');
  GetServerNames(SV_TYPE_SQLSERVER);
```

Végezetül az összes hálózatban elérhető számítógép nevét írjuk a ListBox-ba.

```
  ListBox1.Items.add('_____________');
  ListBox1.Items.add(' - Hálózati gépek:');
  GetServerNames(SV_TYPE_WORKSTATION);
end;
```

GetOSVersion függvény

A GetOSVersion függvény a paraméterként megadott számítógépen futó operációs rendszer nevét adja vissza sztringként.

```
function GetOSVersion(name: String):String;
var
  ptr : Pointer;
  info : PSERVER_INFO_101;
  server : WideString;
begin
  server := name;
```

A feladat elvégzéséhez a NetServerGetInfo függvényt használjuk, a 101-es szintű adatstruktúrával.

```
  if NetServerGetInfo(PWideChar(Server),
    101, ptr) <> 0 then begin
    result := '';
  end else begin
```

Ha sikeres a lekérdezés, akkor kiolvassuk az sv101_version_major adatmezőből a fő verziószámot és ez alapján meghatározzuk azt a szöveges információt, melyet visszatérési értékként adunk.

```
    if (ptr <> nil) then begin
      info := PSERVER_INFO_101(ptr);
      if info^.sv101_version_major <= 4
then
        Result := 'Windows NT'
      else begin
        case info^.sv101_version_minor of
          0: result := 'Windows 2000';
          1: result := 'Windows XP';
        end;
      end;
    end else
      result := 'Windows 9x';
  end;
end;
```

Share eljárás

A Share eljárás a hálózati megosztások felderítését végzi.

```
procedure TForm1.Share();
var
  server: PWideChar;
  Buffer, share: PShare_Info_0;
  Buf1: PShare_Info_1;
  result: DWORD;
  entries: DWORD;
  totals: DWORD;
  r: DWORD;
  i: Integer;
  s: string;
begin
  server := PWideChar(WideString(
    edit1.Text));
  entries := 0;
  totals := 0;
  r := 0;
```

E feladathoz a NetShareEnum függvényt használjuk.

```
  result := NetShareEnum(Server, 0,
Pointer(
    Buffer), DWORD(-1), entries, totals,
r);
```

Sikeres hívás esetén végigmegyünk egy for ciklussal a kapott elemek tömbjén.

```
  if (result = NERR_Success) then begin
    share := Buffer;
    for i := 1 to entries do begin
      s:=string(share.shi0_netname);
      try
```

Minden hálózati megosztásról további információt kérünk le a NetShareGetInfo függvényt felhasználva.

```
        result := NetShareGetInfo(server,
          share.shi0_netname, 1,
          Pointer(Buf1));
        if (result = NERR_Success)
          then begin
          s:=s+' - ' + string(
            Buf1.shi1_remark);
        end;
      except
      end;
```

Az összegyűjtött adatokat megjelenítjük a ListBox-ban.

```
      listBox1.Items.Add(s);
      Inc(share);
    end;
```

A művelet végén a NetApiBufferFree függvény hívásával felszabadítjuk a lefoglalt memóriaterületet.

```
    NetApiBufferFree(Buffer);
  end;
end;
```

GetServerNames eljárás

Ebben az eljárásban a hálózati számítógépek neveit kérdezzük le. A lekérdezett listát szűrhetjük a paraméterben megadott feltételek szerint.

```
procedure TForm1.GetServerNames(t
  : integer);
var
  Buffer: pointer;
  entries, i, res, r, totals:DWORD;
  si: PServerInfo101;
begin
  r := 0;
```

A NetServerEnum függvénnyel végeztetjük el a lekérdezést.

```
  res := NetServerEnum(nil, 101, Buffer,
    MAX_PREFERRED_LENGTH, entries, totals,
t,
    nil, r);
```

Sikertelen hívás esetén befejezzük az eljárás futását.

```
  if (res <> NERR_SUCCESS) then exit;
  try
```

Amennyiben a lista nem üres, egy for ciklussal végigmegyünk a kapott tömbön.

```
    si := Buffer;
    if (entries <> 0) then begin
      for i := 1 to entries do begin
        listBox1.Items.Add('Kiszolgáló
neve:
          ' + si^.name);
        Inc(si);
      end;
    end else
      ShowMessage('Nincs a keresett
        feltételeknek megfelelõ
rendszer!');
  finally
```

A művelet végén a NetApiBufferFree függvény hívásával felszabadítjuk a lefoglalt memóriaterületet.

```
    NetApiBufferFree(Buffer);
  end;
end;
```

Garami Fülöp
Medve László

Kapcsolódó webhelyek

Network Management Functions ↳ 025

Felhasználók ki, be és átjelentkezéseinek figyelése

Cikk sorszáma	Szoftverfeltételek
104	➢ Delphi ➢ Windows XP
Forráskód	\Source\Session\

Windows XP-től kezdve lehetőségünk nyílik arra is, hogy programból figyeljük a felhasználók bejelentkezéseit, illetve a felhasználó váltással járó átjelentkezéseket.

Példaprogram használata

Példaprogram futás közben

Indítsuk el a programot, majd válasszuk a Start menüből a Kijelentkezés - Felhasználó váltása lehetőséget és jelentkezzünk be egy másik felhasználó nevében. Ezt követően jelentkezzünk be ismét az előző felhasználóval. A programunk ablakában látható lesz az átjelentkezések nyomonkövetése.

WTSRegisterSessionNotification függvény

A feladat megvalósításához a WTSRegisterSessionNotification függvényt kell alkalmaznunk. Ennek segítségével WM_WTSSESSION_CHANGE üzenetek formájában értesítést kap programunk.

A függvény deklarációja:

```
BOOL WTSRegisterSessionNotification(
  HWND hWnd,
  DWORD dwFlags
);
```

Paraméterek

- hWnd

Annak az ablaknak a leírója, amelynek az üzenetet kell majd küldeni.

- dwFlags

A dwFlags paraméterben rendelkezhetünk arról, hogy mikor kérünk értesítést:

- ha csak az aktuális munkaszakaszban van változás: NOTIFY_FOR_THIS_SESSION

- ha bármelyik munkaszakaszban van változás: NOTIFY_FOR_ALL_SESSIONS

Visszatérési érték

A függvény igaz értékkel tér vissza, ha a futása sikeresen zárul.

WTSUnRegisterSessionNotification függvény

Ha már nincs többé szükségünk a WM_WTSSESSION_CHANGE üzenetekre, ak-

kor a WTSUnRegisterSessionNotification függvényt kell meghívnunk.

A függvény deklarációja:

```
BOOL WTSUnRegisterSessionNotification(
  HWND hWnd
);
```

Paraméter:

- hWnd

Annak az ablaknak a leírója, melyet a WTSRegisterSessionNotification függvénynél regisztráltunk.

Visszatérési érték

A függvény visszatérési értéke igaz, ha a futása sikeresen zárul.

WM_WTSSESSION_CHANGE üzenet

Amikor bármilyen változás bekövetkezik, akkor kerül elküldésre a WM_WTSSESSION_CHANGE üzenet.

Az üzenetet az ablak kezelő függvénnyel dolgozhatjuk fel.

```
LRESULT CALLBACK WindowProc(
  HWND hWnd,
  UINT Msg,
  WPARAM wParam,
  LPARAM lParam
);
```

Paraméter

- hWnd

Az üzenetet fogadó ablak leírója.

- Msg

WM_WTSSESSION_CHANGE üzenet.

- wParam

A bekövetkezett esemény:

- WTS_CONSOLE_CONNECT

0x1 - munkafolyamat kapcsolódás

WTS_CONSOLE_DISCONNECT

0x2 - munkafolyamat kapcsolat vége

WTS_REMOTE_CONNECT

0x3 - távoli munkafolyamat kapcsolódás

WTS_REMOTE_DISCONNECT

0x4 - távoli munkafolyamat kapcsolat vége

WTS_SESSION_LOGON

0x5 - a felhasználó bejelentkezett

WTS_SESSION_LOGOFF

0x6 - a felhasználó kijelentkezett

WTS_SESSION_LOCK

0x7 - a felhasználó zárolta a munkafolyamatot

WTS_SESSION_UNLOCK

0x8 - a felhasználó feloldotta a zárolt munkafolyamatot

WTS_SESSION_REMOTE_CONTROL

0x9 - távoli vezérlés státusza változott

- lParam

A munkafolyamat egyedi azonosítója.

Visszatérési érték

Az üzenet visszatérési értéke nem kerül feldolgozásra.

A példaprogram megvalósítása

A TForm1-ben deklarálunk egy DoMessage eljárást. Ez kezeli majd az ablakunkhoz beérkező üzeneteket.

```
type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
  { Private declarations }
    FReg: boolean;
    procedure DoMessage(var Msg: TMSG; var
      Handled: boolean);
  end;
```

A feladat megvalósításához deklarálnunk kell külsőként a WTSRegisterSessionNotification és WTSUnRegisterSessionNotification függvényeket, melyek a Wtsapi32.dll-ben találhatók, valamint ezek működéséhez szükséges konstansokat.

```
const
  WTS_CONSOLE_CONNECT = 1;
  WTS_CONSOLE_DISCONNECT = 2;
  WTS_REMOTE_CONNECT = 3;
  WTS_REMOTE_DISCONNECT = 4;
  WTS_SESSION_LOGON = 5;
  WTS_SESSION_LOGOFF = 6;
  WTS_SESSION_LOCK = 7;
  WTS_SESSION_UNLOCK = 8;
  WTS_SESSION_REMOTE_CONTROL = 9;
  WM_WTSSESSION_CHANGE = $02B1;
  NOTIFY_FOR_THIS_SESSION = 0;
  NOTIFY_FOR_ALL_SESSIONS = 1;
  function WTSRegisterSessionNotification(
    Wnd: HWND; dwFlags: DWORD): Boolean;
    stdcall; external 'Wtsapi32.dll';
  function
WTSUnRegisterSessionNotification(
    Wnd: HWND): Boolean; stdcall; external
    'Wtsapi32.dll';
```

Form OnCreate esemény

A program indulásakor, a Form OnCreate eseményénél hívjuk meg a WTSRegisterSessionNotification függvényt, kérve ablakunk részére a WM_WTSSESSION_CHANGE üzenetek küldését, bármely munkafolyamat változás esetén.

```
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  FReg:=WTSRegisterSessionNotification(
    Handle, NOTIFY_FOR_ALL_SESSIONS);
```

A beérkező üzenetek kezelésére a DoMessage eljárásunkat adjuk meg.

```
  Application.OnMessage:=DoMessage;
end;
```

Form OnDestroy esemény

A programunk futásának végén, a Form OnDestroy eseményénél hívjuk a WTSUnRegisterSessionNotification függvényt, annak érdekében, hogy a rendszer ne küldjön több üzenetet a programunk számára.

```
procedure TForm1.FormDestroy(
  Sender: TObject);
begin
  if FReg then begin
    WTSUnRegisterSessionNotification(
      Handle);
  end;
end;
```

DoMessage eljárás

Amikor a DoMessage eljárás meghívásra kerül, akkor elsőként azt kell ellenőriznünk, hogy a beérkező üzenet a WM_WTSSESSION_CHANGE.

```
procedure TForm1.DoMessage(var Msg: TMSG;
  var Handled: Boolean);
var
  s: string;
begin
  Handled:=false;
  if Msg.Message = WM_WTSSESSION_CHANGE
    then begin
    s:=IntToStr(Msg.lParam);
```

Ennél az üzenetnél naplózás céljából egy ListBox-ba gyűjtjük össze a bekövetkezett eseményeket.

```
    case Msg.wParam of
      WTS_CONSOLE_CONNECT: ListBox1.Items.
        Add('CONSOLE_CONNECT: '+s);
      WTS_CONSOLE_DISCONNECT: ListBox1.
        Items.Add('CONSOLE_DISCONNECT:
'+s);
      WTS_REMOTE_CONNECT: ListBox1.Items.
        Add('REMOTE_CONNECT: '+s);
      WTS_REMOTE_DISCONNECT:
ListBox1.Items.
        Add('REMOTE_DISCONNECT: '+s);
      WTS_SESSION_LOGON:
ListBox1.Items.Add(
        'SESSION_LOGON: '+s);
      WTS_SESSION_LOGOFF: ListBox1.Items.
        Add('SESSION_LOGOFF: '+s);
      WTS_SESSION_LOCK: ListBox1.Items.Add(
        'SESSION_LOCK: '+s);
      WTS_SESSION_UNLOCK: ListBox1.Items.
        Add('SESSION_UNLOCK: '+s);
      WTS_SESSION_REMOTE_CONTROL: ListBox1.
        Items.Add(
        'SESSION_REMOTE_CONTROL: '+s);
    end;
  end;
end;
```

Ambrás Péter

Kapcsolódó webhelyek

WTSRegisterSessionNotification ↳ 024

Egy számítógép felhasználói

Cikk sorszáma	Szoftverfeltételek
105	➢ Delphi
Forráskód	\Source\Users\

Programból lekérdezhetjük egy számítógép felhasználóinak adatait nem csak a helyi gépen, hanem a hálózat bármely gépén. Ehhez persze a megfelelő jogosultságokra is szüksége lesz programunknak.

NetUserEnum függvény

A feladat megoldásának kulcsa a NetUserEnum függvény lesz. Ennek segítségével szerezhetünk információkat egy számítógép felhasználóiról.

A függvény deklarációja:

```
NET_API_STATUS NetUserEnum(
  LPCWSTR servername,
  DWORD level,
  DWORD filter,
  LPBYTE* bufptr,
  DWORD prefmaxlen,
  LPDWORD entriesread,
  LPDWORD totalentries,
  LPDWORD resume_handle
);
```

Paraméterek

- servername

A lekérdezendő számítógép neve DNS, vagy NetBIOS formában. Ha ez a paraméter NULL, akkor a lekérdezés a helyi számítógépre vonatkozik.

Windows NT esetén e paraméternél a megadott sztringet mindig \\ karakterekkel kell kezdenünk.

- level

Az adatok lekérdezéséhez többféle struktúrát használhatunk. Hogy aktuálisan melyiket vesszük igénybe, azt a level paraméterben megadott számmal jelezhetjük.

0 - csak a felhasználói neveket kérdezzük le.

A bufptr paraméter USER_INFO_0 struktúrákat tartalmazó tömbre kell, hogy mutasson.

1 - részletesebb információ a felhasználói adatokról.

A bufptr paraméter USER_INFO_1 struktúrákat tartalmazó tömbre kell, hogy mutasson.

2 - részletesebb információ a felhasználói adatokról, kiegészítve felhasználói felhatalmazás és bejelentkezési információkkal.

A bufptr paraméter USER_INFO_2 struktúrákat tartalmazó tömbre kell, hogy mutasson.

3 - részletesebb információ a felhasználói adatokról, kiegészítve felhasználói felhatalmazás és bejelentkezési információkkal, valamint RIDs és profil adatokkal.

A bufptr paraméter USER_INFO_3 struktúrákat tartalmazó tömbre kell, hogy mutasson.

10 - felhasználói név és megjegyzés.

A bufptr paraméter USER_INFO_10 struktúrákat tartalmazó tömbre kell, hogy mutasson.

11 - részletesebb információ a felhasználói adatokról.

A bufptr paraméter USER_INFO_11 struktúrákat tartalmazó tömbre kell, hogy mutasson.

20 - felhasználói azonosító és számos attribútum.

A bufptr paraméter USER_INFO_20 struktúrákat tartalmazó tömbre kell, hogy mutasson.

Windows XP-től kezdve használható.

23 - felhasználói azonosító és számos attribútum.

A bufptr paraméter USER_INFO_23 struktúrákat tartalmazó tömbre kell, hogy mutasson.

Windows 2000/NT esetén nem használható.

- filter

A felhasználók listája szűrhető a filter paraméter segítségével. Ekkor az eredményhalmazban csak azokat a felhasználói adatokat kapjuk vissza, melyek megfelelnek a megadott feltételnek.

A szűrést az alábbi konstansok egyikének megadásával írhatjuk elő:

FILTER_TEMP_DUPLICATE_ACCOUNT - helyi felhasználók listája egy tartomány kiszolgálón.

FILTER_NORMAL_ACCOUNT - globálisan elérhető felhasználók listája.

FILTER_INTERDOMAIN_TRUST_ACCOUNT - tartomány kiszolgálón elérhető felhasználók.

FILTER_WORKSTATION_TRUST_ACCOUNT - munkaállomáson elérhető felhasználók.

FILTER_SERVER_TRUST_ACCOUNT - tartomány kiszolgálón elérhető felhasználók.

- bufptr

A felhasználói adatok listáját kapjuk vissza a bufptr paraméterben. A rendszer által lefoglalt memóriát a NetApiBufferFree függvény hívásával kell majd felszabadítanunk, miután a lekérdezett adatokra már nincs szükségünk.

- prefmaxlen

Az itt megadott darabszámú felhasználói adat kerülhet az eredmény listába. Célszerű használnunk a MAX_PREFERRED_LENGTH konstans értékét.

- totalentries

E paraméterbe kapjuk eredményül, hogy hány felhasználói adat került a bufptr tömbbe.

- resume_handle

Ha az első kereséssel nem sikerült minden adatot lekérdezni, akkor folytathatjuk a keresést egy következő függvény hívással. Ekkor első híváskor nullát, második híváskor már egyet írjunk a resume_handle paraméterbe.

Visszatérési érték

Ha a függvény futása sikeres volt, akkor a NERR_Success értékkel tér vissza, különben a hibakóddal.

USER_INFO_1 struktúra

A mellékelt példaprogramban a USER_INFO_1 struktúrát használjuk majd adatlekérdezéshez, így ismerkedjünk meg ezzel is kicsit közelebbről.

A struktúra deklarációja:

```
typedef struct _USER_INFO_1 {
  LPWSTR usri1_name;
  LPWSTR usri1_password;
  DWORD usri1_password_age;
  DWORD usri1_priv;
  LPWSTR usri1_home_dir;
  LPWSTR usri1_comment;
  DWORD usri1_flags;
  LPWSTR usri1_script_path;
}
 USER_INFO_1,
 *PUSER_INFO_1,
 *LPUSER_INFO_1;
```

A struktúra elemei

- usri1_name

A felhasználó bejelentkező nevét kapjuk meg Unicode sztringben.

- usri1_password

A felhasználó jelszava adható meg. A NetUserEnum és NetUserGetInfo függvények esetén ez a mező NULL.

- usri1_password_age

DWORD típusban kapjuk meg a jelszó életkorát másodpercben mérve. NetUserAdd és NetUserSetInfo függvények esetén e mező nem használt.

- usri1_priv

Felhasználó privilégiuma:

USER_PRIV_GUEST - vendég

USER_PRIV_USER - felhasználó

USER_PRIV_ADMIN - rendszergazda

- usri1_home_dir

Felhasználó kezdőkönyvtára Unicode sztringként.

- usri1_comment

Felhasználóhoz rendelt megjegyzés Unicode sztringként.

- usri1_flags

Felhasználói tulajdonságok, melyek az alábbi konstansok kombinációjából tevődik össze:

UF_SCRIPT

UF_ACCOUNTDISABLE

UF_HOMEDIR_REQUIRED

UF_PASSWD_NOTREQD

UF_PASSWD_CANT_CHANGE

UF_LOCKOUT

UF_DONT_EXPIRE_PASSWD

UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED

UF_NOT_DELEGATED

UF_SMARTCARD_REQUIRED

UF_USE_DES_KEY_ONLY

UF_DONT_REQUIRE_PREAUTH

UF_TRUSTED_FOR_DELEGATION

UF_PASSWORD_EXPIRED

UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION

UF_NORMAL_ACCOUNT

UF_TEMP_DUPLICATE_ACCOUNT

UF_WORKSTATION_TRUST_ACCOUNT

UF_SERVER_TRUST_ACCOUNT

UF_INTERDOMAIN_TRUST_ACCOUNT

- usri1_script_path

Bejelentkezési script elérési útja Unicode sztringként. A script a felhasználó minden bejelentkezésekor lefut.

NetApiBufferFree

A NetUserEnum függvény a futása során a felhasználó adatok tárolásához memóriaterületet foglal a NetApiBufferAllocate függvény segítségével. Azt követően, hogy már nincs szükségünk a lekérdezett adatokra, meg kell hívnunk a NetApiBufferFree függvényt, hogy ezt a lefoglalt memóriaterületet felszabadítsuk.

A függvény deklarációja:

```
NET_API_STATUS NetApiBufferFree(
  LPVOID Buffer
);
```

Paraméterek

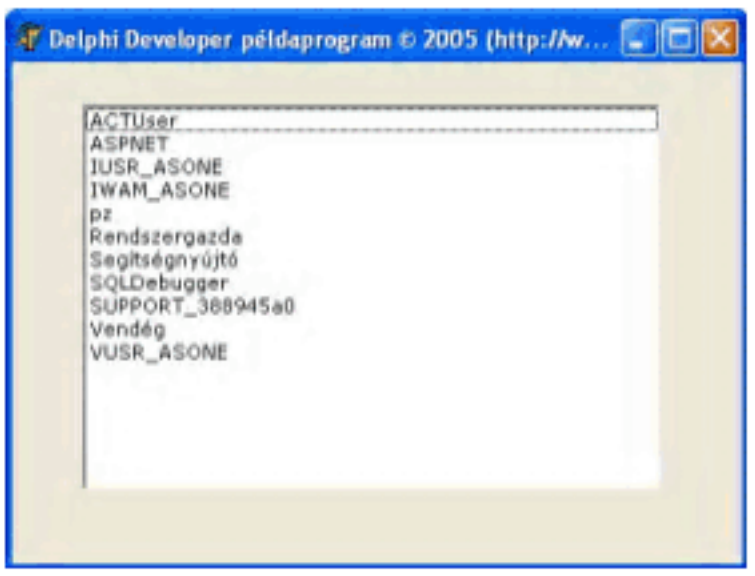
- Buffer

A felszabadítandó memória területre mutató pointer.

Visszatérési érték

Ha a memóriaterület felszabadítása rendben lezajlott, akkor NERR_Success értéket kapunk vissza, ellenkező esetben a hibakódot.

Példaprogram elkészítése



Lekérdezett felhasználói nevek

Fenti ismereteket felhasználva készítünk most egy egyszerű kis példa alkalmazást, melyben egy ListBox-ba megjelenítjük a helyi gépen található felhasználói neveket.

Elsőként deklaráljuk Delphi-ben is a USER_INFO_1 struktúrát:

```
type
  USER_INFO_1 = record
    usri1_name: LPWSTR;
    usri1_password: LPWSTR;
    usri1_password_age: DWORD;
    usri1_priv: DWORD;
    usri1_home_dir: LPWSTR;
    usri1_comment: LPWSTR;
    usri1_flags: DWORD;
    usri1_script_path: LPWSTR;
  end;
  lpUSER_INFO_1 = ^USER_INFO_1;
```

Ezek után deklaráljuk a szükséges függvényeket is, melyeket a netapi32.dll rejt magában.

```
function NetUserEnum(ServerName: PWideChar;
  Level,
  Filter: DWORD;
  var Buffer: Pointer;
  PrefMaxLen: DWORD;
  var EntriesRead,
  TotalEntries,
  ResumeHandle: DWORD): Longword; stdcall;
    external 'netapi32.dll';
function NetApiBufferFree(pBuffer: PByte):
  Longint; stdcall; external
 'netapi32.dll';
```

Form OnCreate esemény

Az adatok lekérdezését a Form OnCreate eseményénél végezzük el.

```
procedure TForm1.FormCreate(
  Sender: TObject);
var
  EntiesRead: DWORD;
  TotalEntries: DWORD;
  UserInfo: lpUSER_INFO_1;
  lpBuffer: Pointer;
  ResumeHandle: DWORD;
  Counter: Integer;
  NetApiStatus: LongWord;
begin
  ResumeHandle := 0;
```

A lekérdezést több részletben végezzük egy repeat ciklus segítségével.

```
  repeat
    NetApiStatus := NetUserEnum(nil, 1, 0,
      lpBuffer, 0, EntiesRead,
      TotalEntries, ResumeHandle);
    UserInfo     := lpBuffer;
```

A kapott felhasználói adatokat egy for ciklussal dolgozzuk fel. Az egyszerűség kedvéért most csupán a felhasználói neveket írjuk ki egy ListBox-ba.

```
    for Counter := 0 to EntiesRead - 1 do
    begin
      listbox1.Items.Add(WideCharToString(
        UserInfo^.usri1_name));
      Inc(UserInfo);
    end;
```

Felszabadítjuk a lefoglalt erőforrásokat.

```
    NetApiBufferFree(lpBuffer);
```

Amennyiben még nem kérdeztük le minden felhasználó adatát, akkor folytatjuk tovább a repeat ciklust, mindaddig, amíg az összes felhasználó neve nem kerül a ListBox-ba.

```
  until (NetApiStatus <> ERROR_MORE_DATA);
end;
```

Másik számítógép felhasználói

Abban az esetben, ha nem a saját gépen szeretnénk lekérdezni a felhasználókat, hanem egy másikon, akkor módosítjuk úgy a NetUserEnum függvény hívását, hogy első paraméterként megadjuk az elérni kívánt számítógép nevét.

```
NetApiStatus :=
  NetUserEnum(PChar('\\NT-Domain'),
  1, 0, lpBuffer,
  0,EntiesRead,
  TotalEntries,
  ResumeHandle);
```


Kapcsolódó webhelyek

NetUserEnum	↳018
NetApiBufferFree	↳019
USER_INFO_1	↳020

Delphi-ben programozható hardverek

Viszonteladók és rendszerfejlesztők

2005. május közepétől indul a viszonteladói és a rendszerfejlesztői státusz megszerzésének lehetősége.

Viszonteladók nagykereskedelmi áron juthatnak bármely Hardware Online termékhez és azt tovább értékesíthetik.

A rendszerfejlesztők lehetnek olyan szoftverfejlesztéssel foglalkozó cégek, személyek, akik a Hardware Online termékekhez saját programot készítenek és azzal együtt értékesítik azokat. A rendszerfejlesztők is nagykereskedelmi áron juthatnak a termékekhez.

Bővítse Ön is cége, vállalkozása termékkínálatát, szerezzen újabb ügyfélkört! Mi adjuk a hardvert, Ön adja a szoftvert!

További részletek a Hardware Online webhelyén jelennek meg előre láthatóan május közepén.

Hardware Online

A Hardware Online termékeivel Ön kibővítheti számítógépeinek képességeit! Például számítógépétől akár több száz méterre elhelyezett szenzor segítségével mérheti az adott hely hőmérsékletét, páratartalmát és akár a légnyomást is. Egy másik eszköz segítségével számítógépe érzékelheti, hogy egy ajtó, vagy ablak nyitva van-e a házában vagy sem. Számítógépével nem csak különféle adatokat gyűjthet, hanem vezérléseket is megvalósíthat, például egy kattintással kinyithatja a bejárati ajtót, ha csengettek, vagy zárhatja a garázs kaput, bekapcsolhat bármilyen 220 volttal működő eszközt. Bármilyen elektronikusan mérhető adatot, vagy elektronikusan vezérelhető eszközt képes lesz számítógépén keresztül használni.

Termékek

A Hardware Online termékeit Ön saját igényei szerint építheti össze. A számítógépéhez tetszőleges számú eszközt csatlakoztathat, melyek akár több száz méterre is lehetnek egymástól, illetve számítógépétől. Minden egyes eszköz igen egyszerűen és rugalmasan kezelhető, a kiépíthető rendszer teljes mértékben csak az Ön igényeitől függ. A rendszer létrehozása szaktudást nem igényel, az eszközöket csupán csatlakoztatni kell egymáshoz, mintha csak építőkockákkal játszana. E "kockákból" pedig teljesen egyedi rendszereket építhet össze, sőt a későbbiek folyamán bármikor bővítheti újabb eszközökkel is a már meglévő rendszerét.

Központi egység

A központi egység köti össze a számítógépet a Hardware Online további eszközeiből felépített hálózattal.

A központi egység USB porton keresztül csatlakozik a számítógéphez. Egy számítógéphez egy központ csatlakoztatható. Minden központ 32 db Hardware Online eszközt képes kezelni, azonban a hálózatba illeszthetünk egy alközponti egységet (jelenleg fejlesztés alatt áll). Ennek segítségével újabb 32 db egység csatlakoztatható a rendszerhez. Az alközpontok száma a hálózatban tetszőleges lehet, így az összeépíthető rendszernek elméletileg nincs felső határa.

A központtól kiépített hálózat kiterjedése elérheti a több száz métert is.

A központi egység üzemeltetéséhez szükség van külső áramforrásra. Ehhez egy olyan külső tápegység szükséges, mely 10-15 V közötti egyen-, vagy váltóáramot biztosít.

A rendszer üzemeltetését a Hardware Online Manager nevű alkalmazással végezheti. Ez a szoftver biztosítja a felépített hálózat működtetését, az adatok gyűjtését, tárolását, sőt akár az internetre történő publikálását is.

Szenzor

A szenzor nevű egység segítségével mérhetünk hőmérsékletet, páratartalmat, légnyomást.

Az eszköz többféle változatban készül. Van olyan változat, mely közvetlenül az USB portra csatlakoztatható, így a szenzoron kívül nincs másra szükség, hogy az használható legyen. Van olyan változat is, mely a központi egységhez csatlakozik és így több száz méterre is helyezhető a számítógéptől, sok más eszközzel együtt.

A szenzor egység öt változatra bontható, mely az alábbi képességekkel rendelkezik:

- precíziós hőmérő + páratartalom mérő + légnyomásmérő
- precíziós hőmérő + páratartalom mérő
- hőmérő + légnyomásmérő
- légnyomásmérő
- hőmérő

Mind az öt fenti változat kapható USB csatolóval, illetve a központi egységhez csatlakoztatható felülettel, így a szenzor nevű egység összesen tíz különböző változatban áll rendelkezésre.

Precíziós hőmérő paraméterei

- Mérési tartomány: -40 °C-tól +80 °C-ig
- Felbontás: 0.01 °C
- Pontosság: ± 0.1 °C

Hőmérő paraméterei

- Mérési tartomány: -40 °C-tól +80 °C-ig
- Felbontás: 0.06 °C
- Pontosság: ± 1.5 °C

Páratartalom mérő paraméterei

- Mérési tartomány: 0-100 %
- Felbontás: 0.03 %
- Pontosság: ± 2%

Légnyomásmérő paraméterei

- Mérési tartomány: 150 hPa - 1150 hPa
- Felbontás: 0.1 hPa
- Pontosság: ± 0.5 hPa

A rendszer üzemeltetését a Hardware Online Manager nevű alkalmazással végezheti. Ez a szoftver biztosítja a felépített hálózat működtetését, az adatok gyűjtését, tárolását, sőt akár az internetre történő publikálását is.

Szoftverfejlesztőknek

Szoftverfejlesztők számára rendkívüli lehetőséget biztosítunk: a Hardware Online minden egyes eszközéhez egyedi szoftvert készíthetnek akár saját használatra, akár eladásra. Így akár még pénzt is kereshet: mi adjuk a hardvert viszonteladói áron, Ön készít hozzá tetszőleges funkciójú szoftvert és egyben értékesíti a kész rendszerét a végfelhasználói számára. Szoftverfejlesztők számára biztosítjuk az eszközök közvetlen, saját programból történő elérését. A Hardware Online SDK teljes mértékben támogatja az új szoftverek készítését az eszközökhöz. A szoftvert készítheti a Delphi-vel, a Visual Studio.NET-tel, vagy bármilyen egyéb szoftverfejlesztői eszközzel.

Hardware Online SDK

A Hardware Online webhelyen bemutatott termékekhez egyedi szoftvereket készíthet. Ebben nyújt segítséget a Hardware Online Software Development Kit (SDK), melyben teljes körű programozói dokumentációt talál, példa forráskódokkal együtt.

A termékek programozása elsősorban Visual Studio.NET-tel, illetve Delphi-vel javasolt. Az SDK-ban e két programozói környezethez talál példaprogramokat. Ez persze nem jelenti azt, hogy más környezetben ne lenne lehetősége új programokat készíteni az eszközökhöz. A felhasználás egyetlen alapfeltétele, hogy az Ön szoftverfejlesztői környezete lehetővé tegye más DLL-ekben lévő funkciók elérhetőségét.

A Hardware Online SDK hamarosan elérhető és ingyenesen letölthető lesz.

Viszonteladói rendszer

Amennyiben Ön, mint szoftverfejlesztő úgy dönt, hogy készítene új, egyedi szoftvereket a Hardware Online oldalon található termékekhez és ezeket értékesítené végfelhasználói, megrendelői számára, akkor lehetőséget biztosítunk, hogy viszonteladói áron juthasson hozzá a termékekhez.

A termékek mellé természetesen továbbra is jár cégünk által készített felhasználói szoftver, melyet Ön szintén továbbadhat a saját szoftverével együtt.

Egyedi szoftverek készítéséhez a Hardware Online SDK nyújt teljes körű segítséget a programozók számára. Ebben teljes körű leírást és forráskód példákat is talál, melyek segítenek abban, hogy saját szoftvereket készítsen az eszközökhöz.

Viszonteladói minősítés megszerzéséhez kérjük vegye fel a kapcsolatot ügyfélszolgálatunkkal e-mail-ben.

Fejlesztés alatt álló termékek

Az alábbiakban röviden összefoglaljuk, hogy milyen eszközök várhatók még a Hardware

Online termékei között a közeljövőben. E termékek egy része már elkészült, de még tesztelés, utolsó finomhangolás alatt állnak, másik része még csak a tervezés fázisában van.

Minden újonnan megjelenő eszközről fontos tudni, hogy azok a már felépített Hardware Online hálózatához csatlakoztathatóak lesznek. Alközpont

Az alközpont nevű egység feladata, hogy a rendszerbe kötve újabb 32 db eszköz csatlakoztatható legyen a hálózathoz. Ez az eszköz szükséges olyan esetben, ha a központi egységhez 32 db-nál több eszközt kívánunk csatlakoztatni. Az alközpontok száma nem korlátozott, így a felépíthető hálózatban szereplő eszközök számának elméletileg nincs felső határa.

Intelligens központ

Az intelligens központi egység képes a számítógép kikapcsolt állapotában is a hálózat vezérlésére, adatok gyűjtésére. A számítógépre csak akkor van szükség az intelligens központ használatakor, amikor a rendszert elindítjuk. Ezt követően csak az adatok továbbításakor, az elemzések megtekintéséhez szükséges a számítógép.

Bemeneti egység

A bemeneti egység segítségével érzékelhetjük, ha egy kapcsoló állapota megváltozik. Ezt felhasználva készíthetünk olyan megoldásokat, hogy a kapucsengőt egy bemeneti egységre kötve számítógépünk érzékelheti ha csengetnek, és ennek megfelelően cselekedhet. A bemeneti egységre mágnes kapcsolót, mikrokapcsolót kötve érzékelheti rendszerünk az ajtó, ablak nyitásokat is. A konkrét megvalósítások lehetősége szinte korlátlan.

Kapcsoló egység

A kapcsoló egységet felhasználva a számítógépünkről kiadott utasítással kapcsolhatunk be különféle eszközöket. Lehetőségünk van kisfeszültségű, vagy akár 220 V-os berendezések bekapcsolására is a számítógépünkön keresztül. Konkrét alkalmazási példa lehet erre a következő: felszerelünk egy mágnes zárt az ajtóra és azt távvezérléssel tudjuk nyitni, zárni a számítógépünk segítségével. Bármilyen elektromos készüléket, berendezést vezérelhetünk ily módon, legyen az egy lámpa, egy vízszivattyú, egy motor vagy akár egy elektromos garázsajtó.

Vezérlő egység

Mivel a Hardware Online hálózata akár több száz méter kiterjedésű is lehet, így nem biztos, hogy minden esetben a számítógépünk mellett tartózkodunk, amikor vezérelni kívánjuk az egyes eszközöket. Ebben segít bennünket a vezérlő egység, melyet a hálózat bármely pontjára ileszthetünk. Az ezen található billentyűk segítségével egyszerű utasításokat adhatunk a rendszer számára, melyek alapján programozható események történnek. Például beütjük a 124-es kódszámot a billentyűzeten, melynek hatására a számítógép felkapcsolja a kültéri világítást az udvarban.

Fény és hangjelző egység

A Hardware Online hálózatában számos adat, információ keletkezik. Ennek elsődleges megtekintési lehetősége nyílván a számítógép monitora lesz. Azonban szükségessé válhat, hogy bizonyos értékek, adatok a hálózat más pontján is elérhetőek, láthatóak legyenek. A fény és hangjelző egység segítségével különféle módon adhat tájékoztatást a rendszer. Például a 150 méterre lévő szenzor által beküldött hőmérséklet értékét egy számítógéptől távol eső szobában kijelezhetjük egy fény és hangjelző egység beépítésével. Hogy miként építjük össze a rendszert és hogy melyik egység, hogyan működjön az csak rajtunk múlik, így végtelenül sokféle, minden igényt kielégítő hálózatot hozhatunk létre.

Szélirány és sebesség egyésg

A szélirány és sebesség egységet felhasználva mérhetjük, hogy merről fúj a szél és azt aktuálisan milyen sebességgel teszi.

Csapadék mennyiség egység

Lehetőségünk van a csapadék mennyiségének mérésére.

Folyadékszint és folyadékáramlás egység

Egy tartályban lévő folyadék mennyiségét mérhetjük a folyadékszint egységgel, míg egy csövön áthaladó folyadék mennyiségét a folyadékáramlás egységgel mérhetjük le.

Mozgásérzékelő, fénysorompó egység

A mozgásérzékelő, fénysorompó egység segítségével rendszerünk érzékelheti a mozgást. Ezt az információt felhasználva az alkalmazásunk tetszőleges műveleteket hajthat végre.

Kamera rendszer

A hálózatunk számos pontján helyezhetünk el miniatűr kamerákat, melyeknek képét számítógépünk monitorán megtekinthetjük, sőt felvételt is készíthetünk a számítógép merevlemezére. Így a felvett anyagot később bármikor visszanézhetjük. A kamera egységet összeköthetjük a mozgásérzékelő egységgel, vagy a bemeneti egységgel és így a felvételt csak akkor kell elindítanunk, amikor a rendszer mozgást, vagy ajtó nyitást érzékel.

GPS vevő

A GPS modul felhasználásával számítógépünk megtudhatja pontos földrajzi helyzetét. E modul

alkalmazása elsősorban mobil gépek esetén hasznos, hiszen ekkor mindig tudhatjuk pontos helyzetünket, bármerre is járnánk.

Digitális iránytű

A digitális iránytű segítségével érzékelhetjük a föld mágneses terét és egy valós iránytűvé változtathatjuk számítógépünket.

Rádiós, GSM és infra adatátvitel

A Hardware Online hálózata vezetékes adatátvitelen alapul. Bizonyos esetekben gondot okozhat a vezetékes rendszer kiépítése. Ekkor jönnek segítségül az alternatív adatátviteli módokat biztosító egységek. Ekkor több vezetékes hálózatot összekapcsolhatunk rádiós adatátvitellel, vagy GSM alapú mobiltelefonos adatátvitellel, de akár lehetőségünk lesz infra adatátvitel megvalósítására is.

Távirányító a számítógéphez

TV, Videó, DVD lejátszó berendezések esetén megszokott a távirányító használata. Most egy olyan eszközt hozunk létre, mellyel a számítógépét vezérelheti. Hogy a távvezérlő egyes gombjai mire képesek, azt Ön programból szabályozhatja. Így a távvezérlővel lehetősége van a Hardware Online hálózatát vezérelni, de akár más programok működését is befolyásolhatja. A távvezérlő vevő berendezését a Hardware Online hálózatra kötve Ön a számítógépétől, akár több száz méterre lévő helyen is használhatja az eszközt vezeték nélküli távvezérléshez.

Analóg-digitális átalakító

Az analóg-digitális átalakítóval lehetősége van 0-10 V közötti feszültség szint digitális jellé történő alakítására. Így analóg adatforrásait is integrálhatja a Hardware Online hálózatába.

Motor vezérlés

Motor vezérlő egységünk segítségével tetszőleges motorok működését befolyásolhatja. Így készíthet akár olyan megoldást is, mely a számítógépről vezérelve felhúzza, leereszti az ablakon lévő redőnyöket, mozgat bármit, amihez egy motor működése szükséges.

Beléptető rendszer

Vezeték nélküli, kártyás beléptető rendszereket integrálhat a Hardware Online hálózatába.

Gáz érzékelő

Földgáz, szénmonoxid és sok más gáznemű anyag érzékelésére és mennyiségének mérésére alkalmas modulokat is csatlakoztathat a Hardware Online hálózatához.

Törésérzékelő, behatolás érzékelő egység

A törésérzékelő, behatolás érzékelő egységet felhasználva a rendszer értesülhet, ha betörik egy ablak és ennek megfelelően tetszőleges műveleteket végezhet.

Egyedi igények

A Hardware Online a termékek széles skáláját biztosítja az Ön igényeinek kielégítésére. Azonban biztos Önben is felmerül majd az eszközök használata közben, hogy mire lenne még szüksége, esetleg egy meglévő eszközben milyen új funkciót látna még szívesen. Új igények nem csak az eszközökkel kapcsolatban hanem a Hardware Online Manager nevű szoftvernél is felmerülhetnek Önben. Ezekkel az igényekkel, egyedi fejlesztésekkel ne habozzon megkeresni bennünket! Szívesen állunk rendelkezésére bármilyen ötlete, igénye van, melyre nem talál megoldást a Hardware Online jelenlegi eszközeivel, szoftvereivel.

Villamosmérnök csapatunk biztosíték az új eszközök gyors, precíz tervezéséhez és kivitelezéséhez, míg szoftverfejlesztői gárdánk elkészíti az új hardver eszközök működtetéséhez szükséges programokat is az Ön számára.

Ötleteit, igényeit kérjük jelezze e-mail-ben ügyfélszolgálatunk felé.

Szoftverfigyelő

↳ - a parancsikon sorszáma (**\Shortcut\Software Lookout**)
web - a szoftver webhelye
trial - a szoftver letölthető próbaverziójának webhelye

SPx - szerviz csomag rendelkezésre áll
UPDx - update pack rendelkezésre áll
BETAx - a szoftver még béta verzió
RCx - a szoftver még RC vezió

Visual Studio.NET

Visual Studio.NET 2002 -
↳ **web: 001**

Visual Studio.NET 2003 -
↳ **web: 002** ↳ **trial: 003**

Visual Studio.NET 2005 **BETA1**
↳ **web: 004**

Delphi

Delphi 6 **UPD2**
↳ **web: 100**

Delphi 7 **UPD1**
↳ **web: 102**

Delphi 8 **UPD2**
↳ **web: 104** ↳ **trial: 107**

Delphi 2005 **UPD1**
↳ **web: 108** ↳ **trial: 109**

.NET Framework

.NET Framework 1.0 **SP3**
↳ **web: 200**

.NET Framework 1.1 **SP1**
↳ **web: 201**

.NET Framework 2.0 **BETA2**
↳ **web: 202**

Windows Family

Windows 2000 **SP4**
↳ **web: 300**

Windows XP **SP2**
↳ **web: 301**

Windows 2003 **SP1**
↳ **web: 303** ↳ **trial: 304**

Virtual PC 2004 **SP1**
↳ **web: 306** ↳ **trial: 307**

Office

Office XP **SP3**
↳ **web: 400**

Office 2003 **SP1**
↳ **web: 401** ↳ **trial: 402**

DirectX

DirectX 9.0c -
↳ **web: 500**

Server

MS SQL Server 2000 **SP3a**
↳ **web: 600** ↳ **trial: 601**

MS SQL Server 2005 **BETA2**
↳ **web: 602**

MS Exchange Server 2000 **SP3**
↳ **web: 603**

MS Exchange Server 2003 **SP1**
↳ **web: 604** ↳ **trial: 605**

Sharepoint Portal Server 2003 **SP1**
↳ **web: 606** ↳ **trial: 607**

Application Center 2000 **SP2**
↳ **web: 608** ↳ **trial: 609**

Commerce Server 2002 **SP2**
↳ **web: 612** ↳ **trial: 613**

Host Integration Server 2000 **SP1**
↳ **web: 616** ↳ **trial: 617**

Identity Integration Server 2003 -
↳ **web: 618** ↳ **trial: 619**

ISA Server 2004 -
↳ **web: 620** ↳ **trial: 621**

Systems Management Server 2003 -
↳ **web: 627** ↳ **trial: 628**

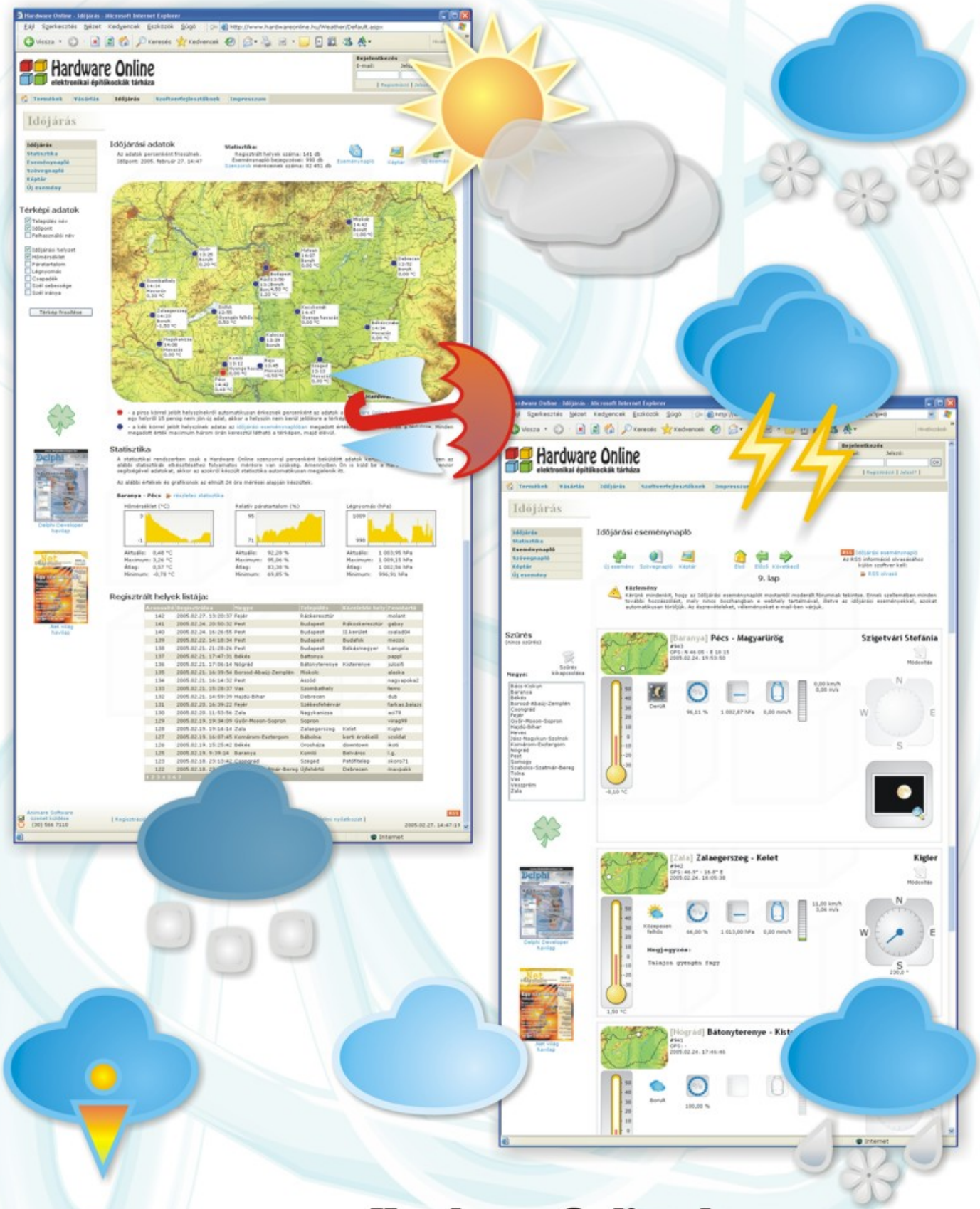
Small Business Server 2003 -
↳ **web: 629** ↳ **trial: 630**

Virtual Server 2005 -
↳ **web: 633** ↳ **trial: 634**

Egyéb letöltés

Microsoft termékek javítócsomagjainak letöltése
↳ **web: 700**

Magyarországi időjárás percről-percre



www.HardwareOnline.hu

Hőmérséklet mérése számítógéppel!

Számítógépétől akár több száz méterre
is helyezhet hőmérőt, páratartalom mérőt,
légnyomásmérőt.

Egy számítógéphez több szenzort
is kapcsolhat és a képernyőn
figyelheti a mért
értékeket, statisztikákat.

Szoftverfejlesztők akár saját, egyedi programokat is készíthetnek
a Hardware Online eszközeihez.

Várjuk viszonteladók jelentkezéseit is, akik nagykereskedelmi áron
juthatnak a Hardware Online termékekhez.

Hardware Online

elektronikai építőkockák tárháza

www.hardwareonline.hu
iroda@animare.hu
(30) 566 7110