Soós Tibor

Microsoft PowerShell 2.0

rendszergazdáknak – elmélet és gyakorlat

```
PS C:\> Get-Book | Where-Object {$_.Title -match "PowerShell 2"}
```

Microsoft Magyarország
2010

© 2010, Soós Tibor, a könyv egyes részei az 1. kiadásból Szerényi László munkái

Második, a PowerShell 2.0-ra átdolgozott kiadás.

Minden jog fenntartva!

A könyv vagy annak bármely része, valamint a benne szereplő példák a szerzővel kötött megállapodás nélkül nem használhatók fel üzleti és oktatási tevékenység során, különös tekintettel **tanfolyami felhasználásra**!

Tanfolyami célra a szerző két kötetre bontott, vízjel nélküli példányt biztosít, gyakorló feladatokkal és megoldásokkal együtt.

A szerző a könyv írása során törekedett arra, hogy a leírt tartalom a lehető legpontosabb és naprakész legyen. Ennek ellenére előfordulhatnak hibák, vagy bizonyos információk elavulttá válhattak.

A könyvben leírt programkódokat mindenki saját felelősségére alkalmazhatja. Javasoljuk, hogy ezeket ne éles környezetben próbálják ki. A felhasználásából fakadó esetleges károkért sem a szerzők, sem a kiadó nem vonható felelősségre.

Az oldalakon előforduló márka- valamint kereskedelmi védjegyek bejegyzőjük tulajdonában állnak.

Zsófinak, aki elviselte, hogy sok-sok éjszakán át dolgoztam ezen a könyvön,

és

Édesapámnak, aki megismertetett a számítógépek világával.

Tartalomjegyzék

1. Elmélet 9

1.1 Előszó a 2. kiadáshoz 9

1.2 Kezdetek 11

1.2.1 A PowerShell telepítése 11
1.2.2 Indítsuk el a PowerShellt! 11
1.2.3 Hello World! 15
1.2.4 DOS parancsok végrehajtása 16
1.2.5 Gyorsbillentyűk, beállítások 16
1.2.6 A TAB-billentyű 19
1.2.7 Promptok, beviteli sor 20
1.2.8 Parancstörténet 21
1.2.9 A PowerShell, mint számológép 23
1.2.10 A konzol törlése 23
1.2.11 Kis-nagybetű 23
1.2.12 A grafikus PowerShell felület – Integrated Scripting Environment 24

1.3 Alapfogalmak 27

1.3.1 Architektúra 27
1.3.2 Command és cmdlet 29
1.3.3 Segítség! (Get-Help) 30
1.3.4 A grafikus help 32
1.3.5 Cmdletek paraméterezése 33
1.3.6 Ki-mit-tud (Get-Member) 35
1.3.7 Alias, becenév, álnév 37
1.3.8 PSDrive, PSProvider 39
1.3.8.1 Meghajtók létrehozása és törlése (New-PSDrive, Remove-PSDrive) 41
1.3.9 Változók, konstansok 43
1.3.10 Idézőjelezés, escape használat 47
1.3.11 Sortörés, többsoros kifejezések 49
1.3.12 Kifejezés- és parancsfeldolgozás 50
1.3.13 Utasítások lezárása 52
1.3.14 Csővezeték, futószalag (Pipeline) 52
1.3.15 Kimenet (Output) 55
1.3.16 Jóváhagyás kezelése, óvatos végrehajtás 57
1.3.17 Egyszerű formázás 59
1.3.18 HTML output 64

1.4 Típusok 66

1.4.1 Típusok, típuskezelés 66
1.4.2 Számtípusok 69
1.4.3 Tömbök 69
1.4.3.1 Egyszerű tömbök 70
1.4.3.2 Többdimenziós tömbök 78
1.4.3.3 Típusos tömbök 79
1.4.3.4 Generic adattípus – paraméterezhető típusos tömb 79
1.4.4 Szótárak (hashtáblák) és szótártömbök 80
1.4.4.1 A szótár általános adattípusa 86
1.4.5 Dátumok ([datetime], Get-Date, Set-Date) 86
1.4.6 Időtartam számítás (New-TimeSpan) 89
1.4.7 Automatikus típuskonverzió 90
1.4.8 Típuskonverzió 91
1.4.9 .NET típusok, statikus tagok 93

A könyv tanfolyami felhasználása nem engedélyezett!

1.4.10 A .NET osztályok felderítése 95
1.4.11 PowerShell objektumok vizsgálata (Format-Custom) 98
1.4.12 Objektumok testre szabása, kiegészítése 99
1.4.13 Osztályok (típusok) testre szabása 103
1.4.14 PSBase, PSAdapted, PSExtended, PSObject nézetek 105
1.4.15 Új típusok létrehozása (Add-Type) 110
1.4.16 Formázás testre szabása (Export-FormatData, Get-FormatData, Update-FormatData) 114
1.4.17 Objektumok mentése, visszatöltése 118

1.5 Operátorok 120
1.5.1 Aritmetikai operátorok 120
1.5.1.1 Összeadás 120
1.5.1.2 Többszörözés 122
1.5.1.3 Osztás, maradékos osztás 123
1.5.2 Értékadás 124
1.5.3 Összehasonlító operátorok 126
1.5.4 Tartalmaz (-contains, -notcontains, -ccontains, -cnotcontains) 128
1.5.5 Dzsóker-minták (-like) 128
1.5.6 Regex (-match, -replace) 130
1.5.6.1 Van-e benne vagy nincs? 131
1.5.6.2 További karakterosztályok 133
1.5.6.3 Van benne, de mi? 135
1.5.6.4 A mohó regex 135
1.5.6.5 Escape a Regex-ben 136
1.5.6.6 Tudjuk, hogy mi, de hol van? 137
1.5.6.7 Tekintsünk előre és hátra a mintában 140
1.5.6.8 A mintám visszaköszön 142
1.5.6.9 Változatok a keresésre 143
1.5.6.10 Tudjuk, hogy mi, de hányszor? 145
1.5.6.11 Regexre cmdlettel (Select-String) 148
1.5.6.12 Csere 149
1.5.7 Logikai és bitszintű operátorok 150
1.5.8 Típusvizsgálat, típuskonverzió (-is, -isnot, -as) 152
1.5.9 Egytagú operátorok (+, -, ++, --, [típus]) 153
1.5.10 Csoportosító operátorok 154
1.5.10.1 Gömbölyű zárójel: () 154
1.5.10.2 Dolláros gömbölyű zárójel: $() 155
1.5.10.3 Kukacos gömbölyű zárójel: @() 156
1.5.10.4 Kapcsos zárójel: {} (bajusz) 156
1.5.10.5 Dolláros kapcsos zárójel: ${} 157
1.5.10.6 Szögletes zárójel: [] 158
1.5.11 Tömbelem-operátor: „,” 158
1.5.12 Tartomány-operátor: „..” 158
1.5.13 Tulajdonság, metódus és statikus metódus operátora: „.”, „::” 159
1.5.14 Végrehajtás 160
1.5.15 Formázó operátor 161
1.5.16 Átirányítás: „>”, „>>” 163
1.5.17 Az összefűzés és szétdarabolás operátora (-join, -split, -csplit) 164

1.6 Vezérlő utasítások 167
1.6.1 IF/ELSEIF/ELSE 167
1.6.2 WHILE, DO-WHILE 167
1.6.3 FOR 168
1.6.4 FOREACH 168
1.6.4.1 $foreach változó 169
1.6.5 ForEach-Object cmdlet 170
1.6.6 Where-Object cmdlet 172
1.6.7 Címkék, törés (Break), folytatás (Continue) 172

A könyv tanfolyami felhasználása nem engedélyezett!

1.6.8 SWITCH....174
1.6.8.1 –wildcard....176
1.6.8.2 –regex....176
1.6.8.3 A $switch változó....177

1.7 Függvények....178
1.7.1 Az első függvényem....178
1.7.2 Paraméterek....178
1.7.2.1 Paraméterinicializálás....179
1.7.2.2 Típusos paraméterek....180
1.7.2.3 Hibajelzés....181
1.7.2.4 Változó számú paraméter....182
1.7.2.5 Hivatkozás paraméterekre....182
1.7.2.6 Paraméterátadás változó szétpasszírozásával (splat operátor)....183
1.7.2.7 Kapcsoló paraméter ([switch])....184
1.7.2.8 Paraméter-definíció a függvénytörzsben (param)....185
1.7.2.9 Paraméterek, változók ellenőrzése (validálás)....186
1.7.3 Függvény a parancsfeldolgozás előnyeinek kihasználására....188
1.7.4 Változók láthatósága (scope)....189
1.7.4.1 Privát, privát, privát....192
1.7.5 Függvények láthatósága, „dotsourcing”....193
1.7.6 Referenciális hivatkozás paraméterekre ([ref])....195
1.7.7 Kilépés a függvényből (return)....196
1.7.8 Pipe kezelése, filter....197
1.7.9 Szkriptblokkok....200
1.7.9.1 Anonim függvények....200
1.7.10 Függvények törlése, módosítása....201
1.7.11 Gyári függvények....204

1.8 Szkriptek....206
1.8.1 Szkriptek engedélyezése és indítása....206
1.8.2 Változók kiszippantása a szkriptekből (dot sourcing)....210
1.8.3 Paraméterek átvétele és a szkript által visszaadott érték....211
1.8.4 Szkriptek írása a gyakorlatban....213
1.8.4.1 PowerGUI Script Editor....213
1.8.4.2 Megjegyzések, kommentezés (#, <# #>)....214
1.8.5 Adatbekérés (Read-Host)....215
1.8.6 Szkriptek digitális aláírása....216
1.8.7 Végrehajtási preferencia....221

1.9 Fontosabb cmdletek....223
1.9.1 Véletlen szám generálás és annál sokkal több (Get-Random)....223
1.9.2 Csővezeték feldolgozása (Foreach-Object) – újra....224
1.9.3 A csővezeték elágaztatása (Tee-Object)....226
1.9.4 Csoportosítás (Group-Object)....226
1.9.5 Objektumok átalakítása (Select-Object)....229
1.9.6 Rendezés (Sort-Object)....234
1.9.7 Még egyszer formázás (Format-Table, Format-Wide)....235
1.9.8 Kimenet megjelenítése grafikus rácsban (Out-GridView)....237
1.9.9 Gyűjtemények összehasonlítása (Compare-Object)....239
1.9.10 Különböző objektumok (Get-Unique)....241
1.9.11 Számlálás (Measure-Object)....242
1.9.12 Nyomtatás (Out-Printer)....243
1.9.13 Kiírás fájlba (Out-File, Export-)....243
1.9.14 Egyéni objektumok létrehozása CSV adatokból (ConvertFrom-CSV)....247
1.9.15 Táblázatok kezelése (Import-Csv, ConvertFrom-Csv, ConvertTo-Csv)....248
1.9.16 Átalakítás szöveggé (Out-String)....249
1.9.17 Lista-tulajdonságok módosítása (Update-List)....250

A könyv tanfolyami felhasználása nem engedélyezett!

1.9.18 Kimenet törlése (Out-Null) ... 252

1.10 Futtatás háttérben ... 253

1.11 Távoli futtatás ... 258

1.12 Összefoglaló: PowerShell programozási stílus ... 263

2. Gyakorlat ... 265

2.1 PowerShell környezet ... 265

2.1.1 Szkriptkönyvtárak, futtatási információk ($myinvocation) ... 265
2.1.1.1 A $MyInvocation felhasználása parancssor-elemzésre ... 269
2.1.2 Automatikus változók ... 270
2.1.2.1 Paraméterezés vizsgálata ($PSBoundParameters) ... 272
2.1.2.2 Preferencia-változók ... 274
2.1.2.3 Lépjünk kapcsolatba a konzolablakkal ($host) ... 275
2.1.3 Környezeti változók (env:) ... 277
2.1.4 Prompt beállítása ... 279
2.1.5 Snapin-ek ... 279
2.1.6 Konzolfájl ... 282
2.1.7 Modulok ... 282
2.1.7.1 Modulok importálása és eltávolítása ... 283
2.1.7.2 Szkriptmodulok ... 286
2.1.7.3 Moduljegyzék készítése (Module Manifest) ... 288
2.1.7.4 Jegyzékmodul (Manifest Module) ... 291
2.1.7.5 Bináris és dinamikus modulok ... 292
2.1.8 Profilok ... 293
2.1.9 Örökítsük meg munkánkat (start-transcript) ... 293
2.1.10 Stopperoljuk a futási időt és várakozzunk (measure-command, start-sleep, get-history) ... 294
2.1.11 Előrehaladás jelzése (write-progress) ... 296
2.1.12 Levélküldés ... 297

2.2 Segédprogramok ... 298

2.2.1 PowerGUI, PowerGUI Script Editor ... 298
2.2.2 RegexBuddy ... 302
2.2.3 Reflector ... 303
2.2.4 PSPlus ... 304

2.3 Hibakezelés ... 306

2.3.1 Megszakító és nem megszakító hibák ... 306
2.3.2 Hibajelzés kiolvasása ($error) ... 309
2.3.3 Hibakezelés globális paraméterei ... 311
2.3.4 Hibakezelés saját függvényeinkben (trap, try, catch, finally) ... 312
2.3.4.1 Többszintű csapda ... 317
2.3.4.2 Dobom és elkapom ... 319
2.3.4.3 Try, Catch, Finally ... 321
2.3.5 Nem megszakító hibák kezelése függvényeinkben ... 324
2.3.6 Hibakeresés ... 325
2.3.6.1 Státusjelzés (write-verbose, write-debug) ... 326
2.3.6.2 Lépésenkénti végrehajtás és szigorú változókezelés (set-psdebug) ... 327
2.3.6.3 Ássunk még mélyebbre (Trace-Command) ... 330
2.3.6.4 Megszakítási pontok kezelése a konzolon ... 333
2.3.6.5 Megszakítási pontok kezelése a grafikus szerkesztőben ... 338
2.3.7 A PowerShell eseménynaplója ... 339

2.4 Fejlett függvények – script cmdletek ... 341

2.4.1 Az első fejlett függvényem ... 341
2.4.2 Paraméterek ellenőrzése ... 345
2.4.3 Csőelemek kezelése ... 347

A könyv tanfolyami felhasználása nem engedélyezett!

2.4.4 Függvényeink óvatos végrehajtása (-WhatIf) 349
2.4.5 Meglevő cmdletek kiegészítése, átalakítása 352
2.4.6 Dinamikus paraméterek 356
2.4.7 Súgó készítése 361
2.4.8 Szkriptek nemzetköziesítése 365

2.5 Fájlkezelés 370
2.5.1 Fájl és könyvtár létrehozása (new-item), ellenőrzése (test-path) 370
2.5.2 További játékok az elérési utakkal 371
2.5.3 Rejtett fájlok 373
2.5.4 Szövegfájlok feldolgozása (Get-Content, Select-String) 373
2.5.5 Sortörés kezelése szövegfájlokban 377
2.5.6 Fájl hozzáférési listája (get-acl, set-acl) 378
2.5.6.1 Fájlok tulajdonosai 380
2.5.6.2 Öröklődés ellenőrzése, beállítása 381
2.5.7 Ideiglenes fájlok létrehozása 384
2.5.8 XML fájlok kezelése 385
2.5.9 Megosztások és webmappák elérése 386

2.6 Az Eseménynapló feldolgozása 388
2.6.1 Hagyományos eseménynaplók kezelése (Get-EventLog) 388
2.6.2 Az új alkalmazás- és szolgáltatásnaplók kezelése (Get-WinEvent) 392
2.6.3 Távoli gépek eseménynaplóinak megtekintése 395
2.6.4 Eseménynaplóval kapcsolatos egyéb műveletek 396

2.7 Registry kezelése 399
2.7.1 Olvasás a registryből 399
2.7.2 Registry elemek létrehozása, módosítása 401
2.7.3 Registry elemek hozzáférési listájának kiolvasása 403

2.8 Tranzakciókezelés 404

2.9 Számítógépek és a hálózati kapcsolatok cmdletei 408

2.10 Helyreállítási pontok kezelése 411

2.11 WMI, processzek, rendszerszolgáltatások 413
2.11.1 WMI objektumok elérése PowerShell-ből 413
2.11.2 Folyamatok és rendszerszolgáltatások 419
2.11.2.1 Szolgáltatások Startup tulajdonsága 422
2.11.3 WMI objektumok metódusainak meghívása (Invoke-WMIMethod) 423
2.11.4 WMI objektumok tulajdonságainak módosítása, új objektumok létrehozása és eltávolítása (Set-WMIInstance, Remove-WMIObject) 428
2.11.5 Fontosabb WMI osztályok 430

2.12 Teljesítmény-monitorozás (Get-Counter, Export-, Import-Counter) 433

2.13 Az ActiveDirectory modul 438
2.13.1 Szűrés AD objektumokra 442
2.13.2 Keresés egyéb paraméterei 447
2.13.3 Objektumok létrehozása 448
2.13.4 Objektumok módosítása 449
2.13.5 Az AD meghajtó 453
2.13.6 Információk az AD erdőről, tartományról, tartományvezérlőkről 459
2.13.7 Egyéb műveletek AD objektumokkal 461
2.13.8 A Management Gateway 462

2.14 Távoli futtatási környezet testre szabása 463

2.15 .NET Framework hasznos osztályai 472
2.15.1 Környezet ([environment]) 472

A könyv tanfolyami felhasználása nem engedélyezett!

2.15.2 Konzol ([console]) 474
2.15.3 Böngészés 477
2.15.4 Felhasználói információk 478
2.15.5 DNS adatok lekérdezése 478

2.16 SQL adatelérés 480
2.16.1 Microsoft Access 480
2.16.2 Windows Search 485

2.17 COM objektumok kezelése 487
2.17.1 A Windows shell kezelése 487
2.17.2 WScript osztály használata 488
2.17.3 Alkalmazások kezelése 490
2.17.3.1 Internet Explorer 490
2.17.3.2 Microsoft Word 492
2.17.3.3 Microsoft Excel 493
2.17.4 Windows Update 495

2.18 Eseményvezérelt futtatás 497
2.18.1 EngineEvent kezelése 497
2.18.2 ObjectEvent kezelése 500
2.18.3 WMIEvent kezelése 504

3. Függelékek 510

3.1 OOP alapok 511
3.1.1 Osztály (típus) 511
3.1.2 Példány (objektum) 511
3.1.3 Példányosítás 511
3.1.4 Metódusok és változók 512
3.1.5 Példányváltozó, példánymetódus 512
3.1.6 Statikus változó, statikus metódus 512
3.1.7 Változók elrejtése 512
3.1.8 Overloaded metódusok 513
3.1.9 Öröklődés 513

3.2 Mi is az a .NET keretrendszer 514
3.2.1 Futtatókörnyezet (Common Language Runtime, CLR) 514
3.2.2 Class Library (osztálykönyvtár) 514
3.2.3 Programnyelvek 515
3.2.4 Programfordítás 515
3.2.5 Programfuttatás 515
3.2.6 Assemblyk (kódkészletek) 516
3.2.7 Érték- és referenciatípusok 516

3.3 A WMI áttekintése 517
3.3.1 A WMI felépítése 517
3.3.2 A WMI objektummodellje 519
3.3.3 Sémák 520
3.3.4 Névterek 520
3.3.5 A legfontosabb providerek 521

3.4 COM-objektumok 523
3.4.1 Típuskönyvtárak 523

3.5 Active Directory kezelése PowerShell 1.0-val 524
3.5.1 Mi is az ADSI? 524
3.5.2 ADSI providerek 524
3.5.3 Az ADSI gyorsítótár 525
3.5.4 Active Directory információk lekérdezése 525

A könyv tanfolyami felhasználása nem engedélyezett!

3.5.5 Csatlakozás az Active Directory-hoz 526
3.5.6 AD objektumok létrehozása 528
3.5.7 AD objektumok tulajdonságainak kiolvasása, módosítása 528
3.5.7.1 Munka többértékű (multivalued) attribútumokkal 533
3.5.7.2 Speciális tulajdonságok kezelése 535
3.5.8 Jelszó megváltoztatása 535
3.5.9 Csoportok kezelése 536
3.5.10 Keresés az AD-ben 536
3.5.10.1 Keresés idő típusú adatokra 539
3.5.10.2 Keresés bitekre 541
3.5.11 Objektumok törlése 542
3.5.12 AD objektumok hozzáférési listájának kezelése 542
3.5.13 Összetett feladat ADSI műveletekkel 544

3.6 Hasznos linkek 547
3.6.1 PowerShell 1.0 linkek 547
3.6.2 PowerShell 2.0 linkek 549

4. Tárgymutató 555

A könyv tanfolyami felhasználása nem engedélyezett!

1. Elmélet

1.1 Előszó a 2. kiadáshoz

A Windows PowerShellnek immár a 2.0 verziója érhető el. Sokan a Microsoft szoftvereit nem használják, amíg azok csak az 1.0-ás verziónál tartanak, megvárják valamelyik magasabb verziószámú változat megjelenését. Ez most a PowerShell esetében elkövetkezett, így már senki sem bújhat ki a továbbiakban a használata alól! ☺

Azok számára, akik még nem találkoztak vele, tisztázzuk, hogy mi is a PowerShell? A PowerShell a Microsoft automatizációs platformja, mellyel olyan feladatokat oldhatunk meg, amelyek ismétlődőek, vagy nagyszámú elemre kell egyszerre alkalmazni, vagy amelyek megoldására megfelelő egyéb eszköz nem áll rendelkezésre.

A PowerShellel leginkább karakteres parancsablakban találkozhat(t)unk, de ez már egyre inkább a múlté, hiszen a 2.0-ás verzióban már kapunk egy grafikus szkriptszerkesztő-futtató környezetet is, másrészt egyéb gyártók is kínálnak már régebb óta ilyen kiegészítéseket, így ha valaki nem szereti a karakteres képernyőt, azok számára is van megoldás. De mindenképpen fontos szerintem a PowerShell megismerését azzal kezdeni, hogy ezt a karakteres-parancssoros környezetet derítsük fel, hiszen a filozófiájának nagyon sok vonatkozása ezen keresztül érthető meg.

A PowerShellt programozási környezetnek is felfoghatjuk, meg programozási nyelvnek is, hiszen minden eszköz rendelkezésünkre áll ahhoz, hogy akár komplett programokat is készítsünk. Ezek a programok interpretáltak, azaz futásidőben értelmezettek, így olyan felhasználási területeken használhatjuk, ahol nem a sebesség a fő szempont.

A PowerShellt én, mint oktató egy nagyon jó pedagógiai eszköznek is tekintem, hiszen segítségével nagyon gyorsan közel lehet vinni bárkihez, - aki egy picit is konyít és érdeklődik a számítógépekhez - a programozást, az objektum-orientáltságot, a .NET keretrendszert és a Windows belsejét. Ezért nagyon örülnék és szívesen segítséget is nyújtanék ahhoz, hogy a PowerShell bekerülhessen a közoktatásba.

Ha egy kicsit szabadabb, szubjektívebb megfogalmazást is megengednek, akkor hadd mondjam azt, hogy szerintem:

> A Windows PowerShell egy univerzális segédprogram-építő platform, amely segít kitörni a grafikus eszközök korlátai mögül.

Persze a PowerShell nagyon nagy tudású eszköz! Ahhoz, hogy ezt hatékonyan tudjuk használni, természetesen meg kell tanulni. Első lépés az elmélet megértése, megtanulása. Ebben a könyvben a PowerShell elméletének majdnem minden olyan porcikáját megpróbálom bemutatni egyszerű példákon keresztül, amelyre egy rendszergazdának, rendszerüzemeltetőnek szüksége lehet. A következő lépés a gyakorlás! Gyakorlás nélkül a PowerShellt nem lehet megtanulni! Én próbáltam, de nem sikerült! Amikor úgy éreztem néhány könyv elolvasása után, hogy értem, tudom, és neki kellett fogni egy konkrét feladat megoldásához, akkor döbbentem rá, hogy nem is tudok semmit a PowerShellből. Azaz pusztán a könyv

A könyv tanfolyami felhasználása nem engedélyezett!

olvasgatása, megtanulása ehhez nem elég. Ez a könyv nem gyakorló füzet, nem is akar az lenni. Gyakorlásra több lehetőséget tudok javasolni: az egyik eljönni hozzám tanfolyamra ☺, ahol sok önálló feladatot adok a tanulóknak, amelyek megoldását közösen megbeszéljük. A másik, hogy tervezek kialakítani egy „PowerShell felhasználók klubja” közösséget az interneten, ahol problémákat lehet felvetni, amelyekhez én, vagy más PowerShell felhasználók megpróbálnak megoldást adni. Ilyen megoldások megalkotásával, vagy ilyen megoldásokat elemzésével, továbbfejlesztésével lehet igazán gyakorolni, tanulni.

Engedjenek meg még pár szót a PowerShell különböző verzióiról. Az első és második verzió között az alapokat tekintve nagyon kevés a különbség. Talán ahhoz lehetne hasonlítani ezt, mint egy autó alapkivitele és extrákkal teletűzdelt változata közti különbség. Azaz amit megszokhattunk az alapkivitelben (forma, lóerő, főbb kezelőszervek) az mind ugyanott van, ugyanolyan a gazdagabb felszereltségűben is, de sok új kényelmi funkcióval találkozunk: automatikus, kétzónás légkondicionáló, négy kerék meghajtás, tolatóradar. Akik persze nem ismerték az első verziót, az „alapkivitelt”, azok számára persze ezek a különbségek nem érzékelhetők, ők már a „tutiba” születtek bele. Ezzel csak azt akarom szemléltetni, hogy nagyon jó alapok lettek lerakva a PowerShell esetében, így az új verziók megjelenésével nem kell újra megtanulni mindent, az előző verzióban megtanult dolgok szinte 100%-ban hasznosulnak a következőben is.

És egy figyelmeztetés! A PowerShell megtanulása nem könnyű feladat. Nekem kb. egy évbe tellett, bízom benne, hogy ennek a könyvnek a birtokában a Tisztelt Olvasónak jóval kevesebb időbe telik majd, de azért ne gondolják, hogy egy hét elég erre! De ne csüggedjenek, vegyék a fáradtságot visszalapozni a könyvben, újra és újra kipróbálni, az itteni példákat és azokat alakítgatni, fejlesztgetni! Megéri, mert ha tényleg készség szinten elsajátítják a „powershellezés” képességét, akkor rengeteg sikerélményhez lehet jutni egy-egy eleinte bonyolultnak tűnő feladat tömör, elegáns megoldása során.

Továbbra is szívesen fogadom a könyvvel kapcsolatos visszajelzéseket! Akár valami elgépelést, vagy nehezen érthető példát találnak, vagy javasolnának valamit, ami még illene egy ilyen jellegű könyvbe, kérem, írják meg! Szívesen elmegyek előadást, tanfolyamot tartani. A tanfolyam elvégzésével még inkább lerövidül az az idő, ami alatt valaki eljut arra a szintre, hogy gyakorlati feladatokat is könnyedén megoldhasson. Szívesen segédkezek akár konzultációs formában konkrét feladatok megoldásában is, ha hívnak.

Jó tanulást és sok sikerélményt!

Budapest, 2010. március 7.

Soós Tibor
soost@IQJB.hu
PowerShell MVP, MCT, MCITP…

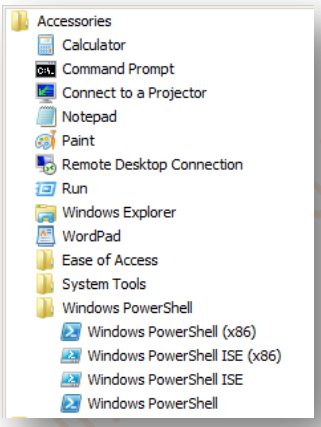
A könyv tanfolyami felhasználása nem engedélyezett!

1.2 Kezdetek

1.2.1 A PowerShell telepítése

A PowerShell 2.0 a Windows 7 és a Windows Server 2008 R2 része, de a Windows Management Framework[1] részeként telepíthető Windows XP SP3, Windows Server 2003 SP2 és ezeknél újabb operációs rendszerre. A telepítés előfeltétele a .NET Framework 2.0 megléte a gépen. Mind a PowerShell, mind a .NET Framework letölthető a Microsoft weboldaláról.

A PowerShellt a Start menü *All Programs*/Accessories/*Windows PowerShell* csoportjában található *Windows PowerShell* parancsikon segítségével, vagy közvetlenül a *powershell.exe* meghívásával indíthatjuk. A 64 bites Windows-okon két PowerShell ikonunk is lesz, egy 64 bites és egy 32 bites parancssori környezet indítására:



1. ábra PowerShell programcsoport ikonjai (64 bites)

Találhatunk itt még egy (esetleg kettő) PowerShell ISE parancsikont is, erről majd kicsit később.

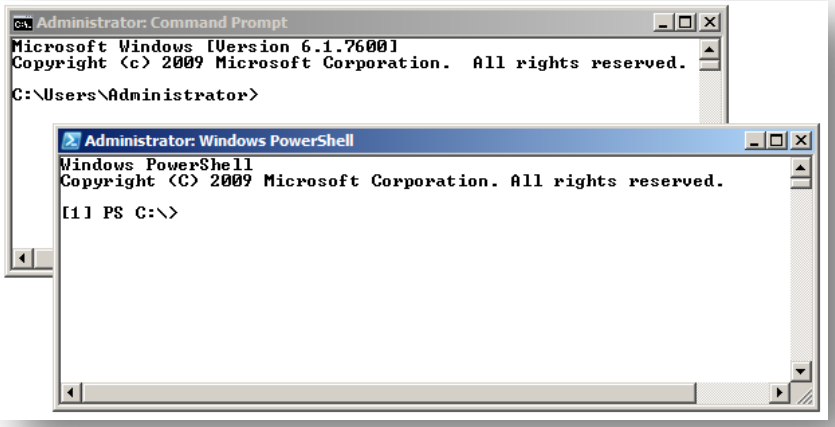
1.2.2 Indítsuk el a PowerShellt!

A Windows PowerShell indítása után a régi, megszokott Parancssorhoz (cmd.exe) hasonló felület jelenik meg. Amint az alábbi ábrán is látható, kísérteties a hasonlóság, nem lesz itt baj, gondosan tartsuk magunktól távol a dokumentációt és próbálkozzunk bátran.

[1] Letölthető a http://support.microsoft.com/default.aspx/kb/968929 helyről.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



2. ábra, A cmd.exe és a PowerShell

Megjegyzés

A könyvben szereplő képernyőképek és kimásolt szkriptek többsége Windows Server 2008 R2 verzión születtek. Az esetleges nyomtatás festéktakarékossága okán az alaphelyzet szerinti sötétkék háttérkép előtti fehér betűket kicseréltem fehér háttérkép előtti fekete betűkre. Hogy könnyebben tudjak a példaszkriptek soraira hivatkozni az alaphelyzet szerinti

```
PS C:\Users\Administrator>
```

promptot lecseréltem

```
[sorszám] PS C:\
```

promptra. Így a könyvben szereplő képernyőképek és példaszkriptek formailag nem pont ugyanazt adják, mint ha Ön kipróbálja ezeket a saját gépén.

Kezdetnek próbáljuk ki tehát a megszokott parancsokat, vajon mit szól hozzájuk az új shell.

```
[1] PS C:\> dir


    Directory: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----        2009.07.14.      5:20            PerfLogs
d-r--        2009.11.09.     21:23            Program Files
d-r--        2009.11.11.     22:22            Program Files (x86)
d-r--        2009.11.09.     21:15            Users
d----        2009.11.15.     12:35            Windows
-a---        2009.11.11.      0:32        458 users.txt
```

Amint látható, nem történt semmi meglepő, kaptunk egy mappa- és fájllistát, nagyjából a megszokott módon. Eddig rendben, a nehezén túl is vagyunk, ez is csak egy parancssor. Próbálkozzunk valami nehezebbel, írjuk be mondjuk a következőt:

```
[2] PS C:\> dir /s
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Get-ChildItem : Cannot find path 'C:\s' because it does not exist.
At line:1 char:4
+ dir <<<<  /s
    + CategoryInfo          : ObjectNotFound: (C:\s:String) [Get-ChildItem
   ], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.G
   etChildItemCommand
```

Hát ez már nem jött össze: szép piros hibaüzenet tájékoztat, hogy a `C:\s` mappa nem létezik. Ez így teljesen igaz is, de ki akart oda menni? (Egyébként teljesen korrekt a hibaüzenet, mivel ha mégis van ilyen mappa, akkor azt gond nélkül kilistázza, vagyis a / karakter a gyökérmappát jelenti.) Mi történik itt? Úgy látszik, mégis segítséget kell kérnünk, de vajon hogyan? Mondjuk, legyen a következő:

```
[3] PS C:\> help dir

NAME
    Get-ChildItem

SYNOPSIS
    Gets the items and child items in one or more specified locations.


SYNTAX
    Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Exclude <stri
    ng[]>] [-Force] [-Include <string[]>] [-Name] [-Recurse] [-UseTransacti
    on] [<CommonParameters>]

    Get-ChildItem [-LiteralPath] <string[]> [[-Filter] <string>] [-Exclude
    <string[]>] [-Force] [-Include <string[]>] [-Name] [-Recurse] [-UseTran
    saction] [<CommonParameters>]


DESCRIPTION
    The Get-ChildItem cmdlet gets the items in one or more specified locati
    ons. If the item is a container, it gets the items inside the container
-- More  --
```

Talán meglepő, de a `help` parancs működik. Talán még meglepőbb, a `man` parancs is működik! A válaszból viszont az derül ki, hogy a `dir` egyáltalán nem is létezik, a mappalistát valójában a `Get-ChildItem` nevű parancs produkálta az előbb. Kiderül az is, hogy hogyan kell őt az alkönyvtárakba leküldeni: `-recurse`, vagy egyszerűen: `-r` kapcsolóval:

```
[4] PS C:\> dir -r


    Directory: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----        2009.07.14.      5:20            PerfLogs
d-r--        2009.11.09.     21:23            Program Files
d-r--        2009.11.11.     22:22            Program Files (x86)
d-r--        2009.11.09.     21:15            Users
d----        2009.11.15.     12:35            Windows
-a---        2009.11.11.      0:32        458 users.txt
```

```
    Directory: C:\PerfLogs


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       2009.07.14.      4:35            Admin
…
```

A `Get-ChildItem` leírását elolvasva joggal kérdezhetjük, hogy miért változott meg az égvilágon minden, ha egyszer ugyanazt a feladatot hajtja végre a `Get-ChildItem`, mint a jó öreg `dir` parancs. A válasz pedig egyszerűen az, hogy nem ugyanazt végzi el, nem ugyanúgy, és nem ugyanazzal az eredménnyel! A `Get-ChildItem` például nemcsak mappákból, hanem számos más adatforrásból is képes listát produkálni, így a regisztrációs adatbázisból, a tanúsítványtárból, stb. is, kimenete pedig korántsem az a lista, amit a képernyőn láttunk.

Ha kiadjuk az alábbi parancsot, szomorúan láthatjuk, hogy `help` parancs sem létezik, az ő igazi neve `Get-Help`.

```
[5] PS C:\> help help

NAME
    Get-Help

SYNOPSIS
    Displays information about Windows PowerShell commands and concepts.


SYNTAX
    Get-Help [-Full] [[-Name] <string>] [-Category <string[]>] [-Component
    <string[]>] [-Functionality <string[]>] [-Online] [-Path <string>] [-Ro
    le <string[]>] [<CommonParameters>]
…
```

Hogy tovább szaporítsuk a nem létező, de ennek ellenére jól működő parancsok számát, próbáljuk ki a `cd` parancsot, majd kérjük el az ő súgólapját is. Láthatjuk, hogy az igazi neve `Set-Location`, és természetesen nemcsak a fájlrendszerben, hanem a regisztrációs adatbázisban, tanúsítványtárban, stb. is remekül működik. Valójában a `cd`**,** és a többi nem létező, de működő parancs csupán egy alias[2], egy becenév, álnév a megfelelő PowerShell parancsra (lásd: `Get-Alias`), amelyek hivatalos neve cmdlet (ejtsd: kommandlet, magyarul parancsocska).

A következőkben felsorolunk a fenti parancsokkal kapcsolatos néhány egészen egyszerű példát, amelyek között részben a „régi” cmd parancsok megfelelőit, részben az új képességek demonstrációját találhatjuk.

- dir *.txt /s

```
PS C:\> Get-ChildItem -Include *.txt -Recurse
```

- A HKLM/Software ág kilistázása a registryből.

[2] A `help` és a `man` nem alias, hanem függvény, de ennek a mi szempontunkból most nincs jelentősége.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> Get-ChildItem registry::HKLM/Software
```

- dir *.exe

```
PS C:\> Get-ChildItem * -i *.exe
```

- cd q:

```
PS C:\> Set-Location q:
```

- Jelenlegi pozíció elmentése[3].

```
PS C:\> Push-Location
```

- Átállás a HKEY_CURRENT_USER ágra a registryben.

```
PS C:\> Set-Location HKCU:
```

- Visszaugrás egy korábbi elmentett pozícióra.

```
PS C:\> Pop-Location
```

- Átállás a tanúsítványtár „meghajtóra”.

```
PS C:\> Set-Location Cert:
```

Azért volt néhány elég furcsa külsejű parancs, igaz? Tanúsítványtár meghajtó!? Mi lesz még itt?

Kísérletezzünk tovább! Ha már ilyen szépen tudjuk használni a nem létező parancsokat, próbáljunk ki valami olyat, ami tényleg biztosan nem létezhet:

```
PS C:\> 1
1
```

Semmi hibaüzenet, a PowerShell egyszerűen visszaírta a számot a konzolra.

A PowerShell ablakot bezárni az `Exit` paranccsal lehet, de ez nem PowerShell parancs, nem is alias, a help sem ad semmi támpontot.

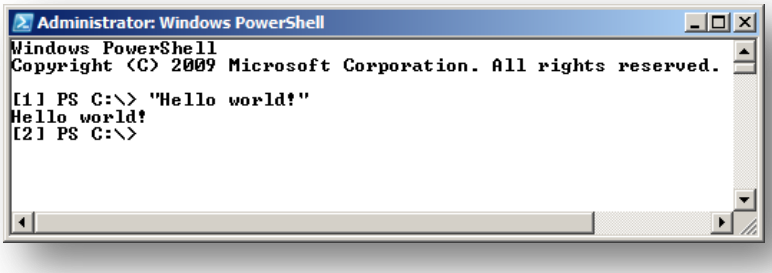
1.2.3 Hello World!

Az előzőekben a PowerShell saját parancsaival és DOS-os álnevekkel kísérleteztünk, most kezdjünk el alkotni, saját programot írni! Bjarne Stroustrup óta, azt hiszem, minden programozással foglalkozó könyv azzal kezdődik, hogy olyan programot írunk, amelyik kiírja a képernyőre, hogy „Hello world!”. Nézzük meg, hogy ez hogyan megy PowerShellben:

[3] Ahogyan a név sugallja, több pozíció is elmenthető egymás „tetejére”, vagyis egy verembe. A Pop-Location mindig a legfelül lévőre tér vissza, majd eldobja azt.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

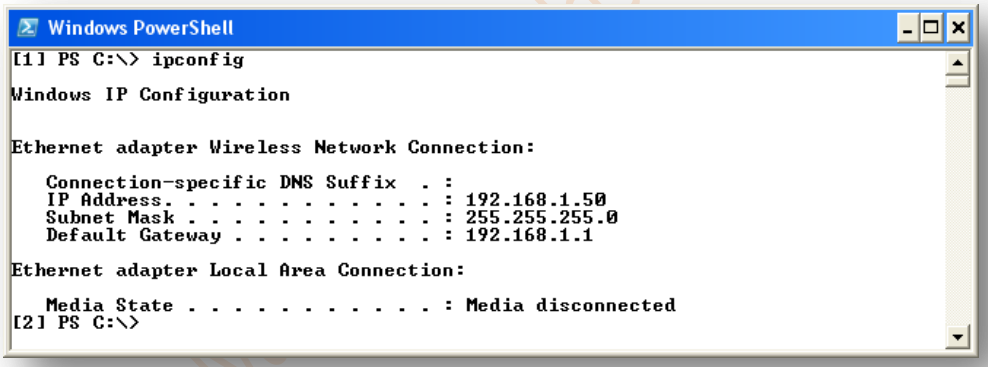


3. ábra Első PowerShell „programom”

Nagyon egyszerű, idézőjelek közt beírjuk a szöveget, Enter, és már ki is írta. Ennél egyszerűbben nem is lehetne ezt!

1.2.4 DOS parancsok végrehajtása

Láttuk, hogy a `Dir` parancs lefutott, de könnyű neki, hiszen a `Dir` egy PowerShell parancs álnéven. De vajon a többi DOS paranccsal mi történik? Azokkal is nyugodtan próbálkozzunk! Például próbáljuk ki az `IPConfig` parancsot:



4. ábra IPConfig a PowerShell ablakban

Ez is lefutott, az `ipconfig` ugyanúgy működik, mint a DOS ablakban. Ehhez hasonlóan a többi DOS parancsot is nyugodtan kipróbálhatjuk, azaz akár teljes egészében áttérhetünk a PowerShell környezet használatára, akár a nem PowerShell parancsok futtatásával is. (Persze óvatosan, nem biztos hogy minden korábbi DOS-os batch fájlunk le fog futni.)

1.2.5 Gyorsbillentyűk, beállítások

Nagyban meggyorsítja munkánkat, ha ismerjük azokat a gyorsbillentyűket, amelyek elérhetőek a PowerShell ablakban:

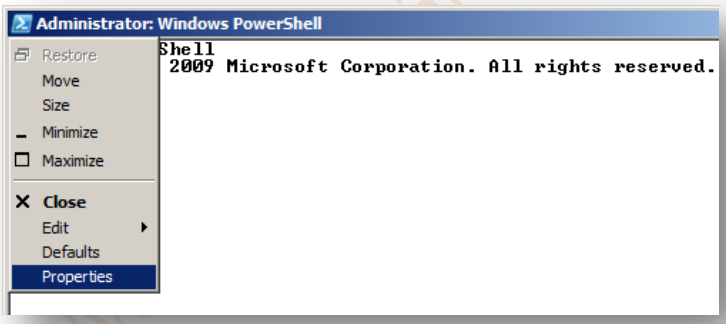
TAB	Automatikus kiegészítés, további TAB további találatokat ad
Shift + TAB	Visszalép az előző találatra

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

F1	1 karakter a történet-tárból
F2	Beírt karakterig másol
F3 vagy ↑	Előző parancs a történet-tárból
F4	Kurzortól az adott karakterig töröl
F5 vagy ↓	Lépked vissza a történet-tárból
F6	hatástalan
F7	Történettárat megjeleníti
F8	Parancstörténet, de csak a már begépelt szövegre illeszkedőeket
F9	Bekéri a történettár adott elemének számát és futtatja
Jobbkattintás	Beillesztés
Egérrel kijelöl + Enter	Másolás
→	Karaktert lépked, üres sorban a parancstörténet-tárból hív elő karaktereket
←	Karaktert lépked
ESC	Törli az aktuális parancssort

Tovább gyorsítja a munkánkat, ha a PowerShell ablak néhány beállítására is odafigyelünk. Ha jobb egérgombbal kattintunk az ablak bal felső sarkában levő PowerShell ikonra ( ), akkor a következő menü jelenik meg:

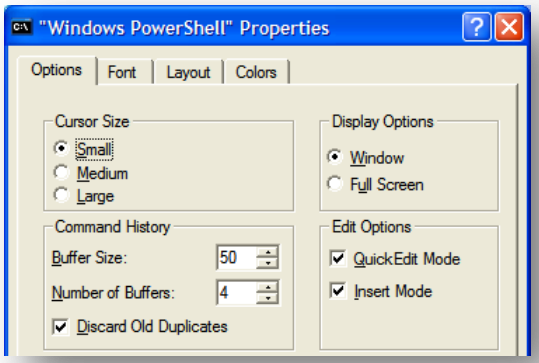


5. ábra A PowerShell ablak menüje

Itt minket a „*Properties*” menüpont érdekel, kattintsunk rá. A megjelenő párbeszédpanelben van néhány fontos, munkánkat segítő beállítási lehetőség:

Az „*Options*” fülön a „*QuickEdit Mode*” fontos, hogy be legyen kapcsolva. Ennek birtokában tudjuk az egérrel kijelölni a konzolon megjelenő szövegeket és Enter leütésével a vágólapra helyezni. Szintén ez kell ahhoz, hogy a vágólapra valahonnan kimásolt szöveget a jobb egérgombbal beilleszthessük a kurzor pozíciójától kezdődően.

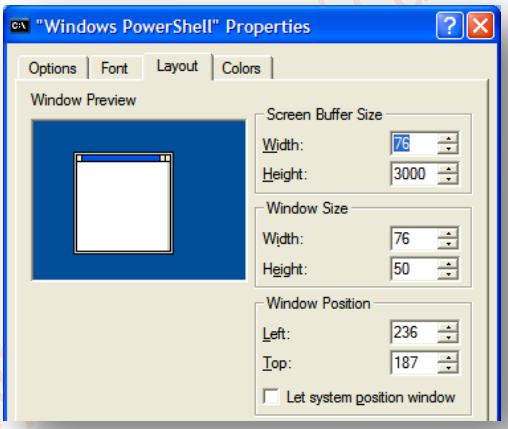
A könyv tanfolyami felhasználása nem engedélyezett!



6. ábra Options fül

A „*Font*” fülön állíthatunk be az igényeinknek és szemünknek esetleg jobban megfelelő betűtípust és méretet.

A „*Layout*” fülön lehet optimálisabb ablakméretet beállítani. Ez főleg az ablak szélessége szempontjából fontos, mert majd látni fogjuk, hogy a PowerShell parancsok kimenete nagyon gyakran táblázat jellegű, és annak jobb, ha minél szélesebb ablakban tudjuk megjeleníteni.



7. ábra Az ablak méretének beállítása

Jelen könyvbe másolt konzoltartalmak miatt én az ablakméretemet levettem 79 karakterre, mert ez pont még megfelelő fontmérettel beilleszthető a könyv lapjaira, de igazi munkában, amikor jó nagy felbontású monitort használunk, akkor ezt érdemes akár 150 karakterre megnövelni.

Szintén praktikus, ha a látható ablak szélessége és a mögötte levő „*Screen Buffer*” szélessége egyforma, mert akkor nem lesz vízszintes gördítő sávunk, ami jó, ha nincs. Ezzel szemben a függőleges puffer-méret lehet jó nagy, hogy hosszabb listák is gond nélkül teljes terjedelmükkel kiférjenek, és függőleges gördítéssel megtekinthetőek legyenek.

Az utolsó fül a színek beállítására való. Alaphelyzetben mélykék alapon fehér betűkkel nyílik meg a PowerShell ablak. Ha valaki könyvet ír, és a képernyőképek nyomtatásra kerülnek, akkor érdemes festékkímélés céljából inkább fehér alapon fekete betűket használni, ezt tettem én is.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ha testre szabtuk az ablakot, akkor a párbeszédpanel bezáráskor rákérdez a felület, hogy ezeket a beállítások csak az adott ablakra vonatkozzanak, vagy legközelebb is viszont akarjuk látni a változtatásainkat, ha újra megnyitjuk a Start menüből a PowerShell ablakot. Érdemes ez utóbbi opcióval elmenteni változtatásainkat kipróbálás után.

1.2.6 A TAB-billentyű

Az egyik legtöbbet használt gyorsbillentyű a TAB-billentyű, amit érdemes kicsit részletesebben is tárgyalni, mivel ez rengeteg fölösleges gépeléstől mentheti meg a lustább (ez határozottan pozitív tulajdonság) szkriptelőket. A PowerShell TAB-billentyűje annyira sokrétűen használható, hogy akár olyan információforrásként is tekinthetünk rá, ami gyorsan, könnyen elérhető és mindig kéznél van.

A TAB használata nem igazi újdonság, nyilván sokan használták a *cmd.exe*-be épített változatát, de a PowerShellben számos új képességgel is találkozhatunk. Mi a teendő például, ha át akarunk váltani a *C:\Users* mappára? Természetesen begépelhetjük a teljes parancsot:

```
PS C:\> Set-Location "c:\Users"
```

Nem is olyan sok ez. Akinek mégis, az csak ennyit írjon:

```
PS C:\> Set-Location c:\U<tab>
```

Ha a c: meghajtó gyökerében nincs más 'u' betűvel kezdődő mappa, akkor készen is vagyunk, a shell kiegészítette a parancsot, még az idézőjeleket is begépelte helyettünk. Ha több mappa is kezdődik 'u'-val akkor sincs nagy baj, a TAB-billentyű szorgalmas nyomkodásával előbb-utóbb eljutunk a megfelelő eredményig. Még ez is túl sok? Próbálkozzunk tovább! A PowerShell nemcsak a mappa és fájlneveket, hanem a cmdletek nevét is kitalálja helyettünk:

```
PS C:\> Set-L<tab> c:\u<tab>
```

Mi történik vajon, ha csak ezt gépeljük be:

```
PS C:\> Set-<tab>
```

Bizony, a TAB-billentyű nyomkodásával végiglépkedünk az összes `Set-` parancson, így akkor is könnyen kiválaszthatjuk a megfelelőt, ha a pontos név éppen nem jut eszünkbe. A Shift+TAB-bal vissza tudunk lépni az előző találatra, ha túl gyorsan nyomkodva esetleg tovább léptünk, mint kellett volna.

Mi a helyzet a cmdletek paramétereivel? Gépeljük be a következőt:

```
PS C:\> Get-ChildItem -<tab>
```

Talán már nem is meglepő, hogy a lehetséges paraméterek listáján is végiglépkedhetünk, hogy a megfelelőt könnyebben kiválaszthassuk közülük.

A paraméterek megadásával kapcsolatban itt említünk meg egy másik lehetőséget is (bár nem kell hozzá TAB). Egyszerűen annyi a teendőnk, hogy egyáltalán nem adunk meg paramétert, mivel ebben az esetben a parancs indítása előtt a PowerShell szép sorban bekéri a kötelező paramétereket:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> New-Alias

Supply values for the following parameters:
Name: ga
Value: Get-Alias
```

Hamarosan részletesen megismerkedünk a PowerShell változók rejtelmeivel és az objektumok használatával, most csak egy kis előzetes következik a TAB-billentyű kapcsán. Ha a PowerShell munkamenetben saját változókat definiálunk, a TAB ezek nevének kiegészítésére is képes, amint az alábbi példán látható:

```
PS C:\> $ezegynagyonhosszunevuvaltozo = 1
PS C:\> $ez<tab>
```

Ügyes! És mi a helyzet a .NET vagy COM-objektumok tulajdonságaival és metódusaival? A TAB-billentyű ebben az esetben is készségesen segít:

```
PS C:\> $s = "helló"
PS C:\> $s.<tab>
PS C:\> $excel = new-object -comobject excel.application
PS C:\> $excel.<tab>
```

Aki még arra is kíváncsi, hogy ezt a sok okosságot mi végzi el helyettünk, adja ki a következő parancsot:

```
PS C:\> $function:tabexpansion
```

A képernyőn a PowerShell beépített `TabExpansion` nevű függvényének kódja jelenik meg; ez fut le a TAB-billentyű minden egyes leütésekor. A kód most még talán egy kicsit ijesztőnek tűnhet, de hamarosan mindenki folyékonyan fog olvasni PowerShellül. A kód azonban nemcsak olvasgatásra jó, meg is változtathatjuk azt, sőt akár egy teljesen új, saját függvényt is írhatunk a „gyári” darab helyett (a függvényekről később természetesen még lesz szó).

1.2.7 Promptok, beviteli sor

Látható, hogy a PowerShell is a parancssorok elején, amikor tőlünk vár valamilyen adatbevitelt, akkor egy kis jelző karaktersorozatot, un. promptot ír ki. Ez alaphelyzetben így néz ki:

```
PS alaphelyzet szerinti könytár elérési útja>
```

Az „alaphelyzet szerinti könyvtár” nagy valószínűséggel a `Users\felhasználónév` könyvtárra mutat. Ez a prompt testre szabható. Itt ebben a könyvben sok helyen ilyen módosított prompt látható a példáimnál:

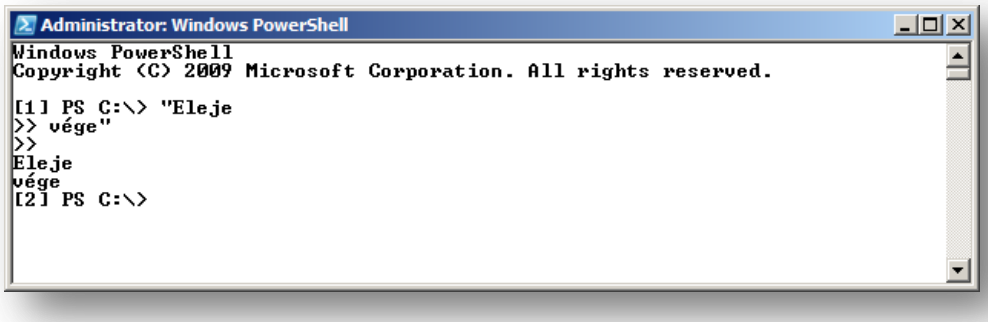
```
[1] PS C:\>
```

Ahogy korábban említettem, a PS elé biggyesztettem egy növekvő számot szögletes zárójelek között, hogy jobban tudjak hivatkozni az egyes sorokra a magyarázó szövegben.

A prompt többszintű, viszont csak 1 szint mélységig jelzi a >> jellel az alapszinttől eltérő mélységet, ennél mélyebb szintekre nincs külön speciális prompt:

A könyv tanfolyami felhasználása nem engedélyezett!

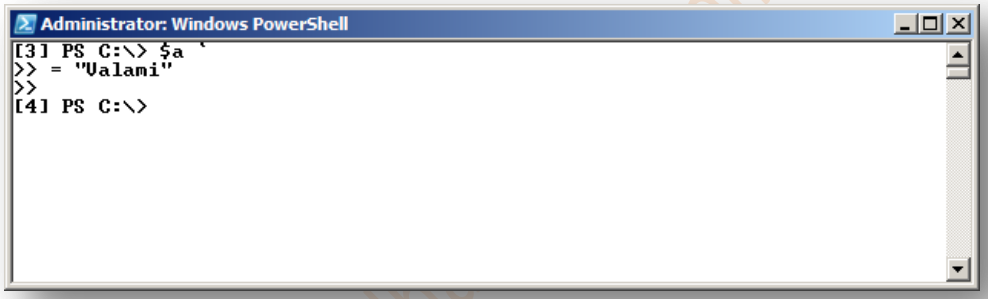
© Soós Tibor – soost@ IQJB.hu



8. ábra Többszintű prompt

Ha „érzi”, hogy kell még folytatódnia a sornak, mert idézőjel szerepel a sor elején, vagy valamilyen nyitó zárójel, akkor >> lesz a prompt. Ha már ki akarunk szállni a mélyebb szintű promptból, akkor azt egy üres sorban leütött Enterrel tehetjük meg.

Ha egy hosszabb sort inkább külön sorba szeretnénk tenni, a „visszafeleaposztróf” (` - AltGr 7) használatos:



9. ábra Sortörés Escape (`) karakter után

Ez az un. *Escape* karakter, ami hatástalanítja a következő speciális karakter (itt pl. az Enter) hatását. Amúgy nyugodtan folytatódhat a parancssorunk a következő sorban, ha túl hosszú.

Ha több utasítást szeretnénk egy sorba beírni, akkor a „;” karakterrel tudjuk azokat elválasztani.

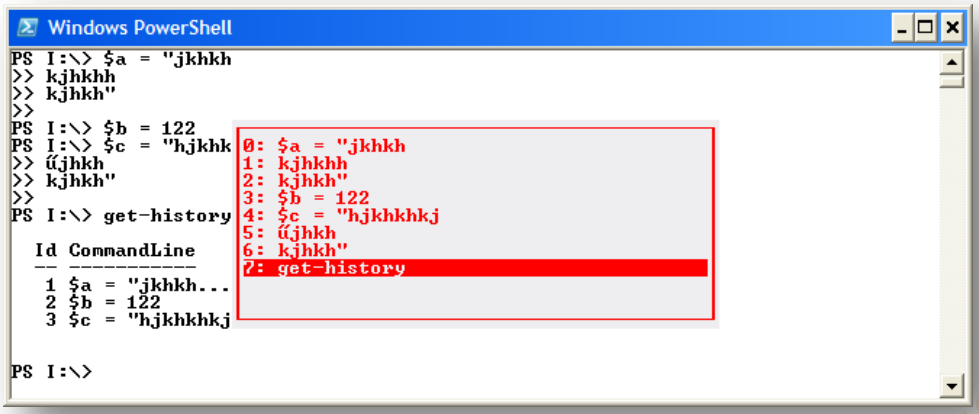
```
[1] PS C:\> "Első utasítás"; 1+3; "Vége"
Első utasítás
4
Vége
```

1.2.8 Parancstörténet

A PowerShell ablak megjegyzi a korábban begépelt parancssorainkat. Ezt a parancstörténet-tárat az F7 billentyűvel tudjuk megjeleníteni. Vigyázat! Kétfajta történet-sorszámozás van:

A könyv tanfolyami felhasználása nem engedélyezett!

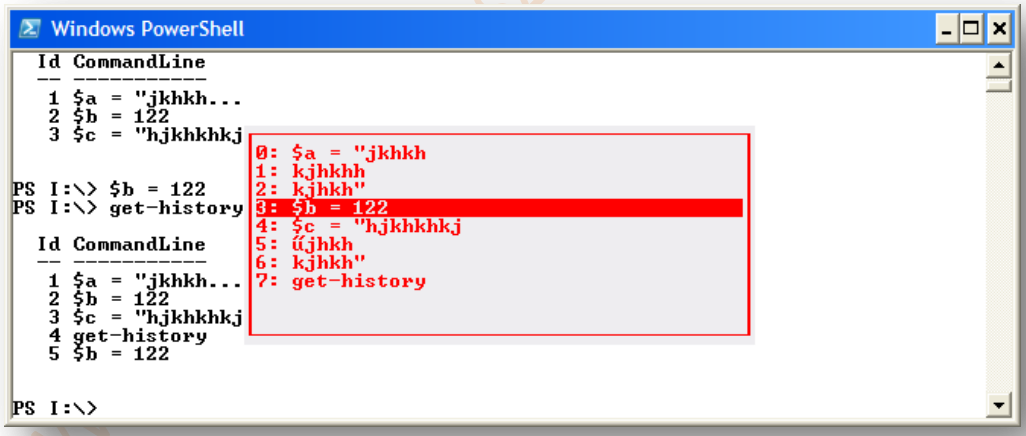
© Soós Tibor – soost@ IQJB.hu



10. ábra A PowerShell parancstörténet-tár

Az F7-re megjelenő történettárban minden tényleges sor (alpromptok is) külön vannak nyilvántartva. Van azonban egy PowerShell cmdletünk is, a `get-history`, amiben főpromptonként kapunk új sorszámot. Ráadásul a történettárból előhívott parancsok esetében a sorszámok másként viselkednek a kétfajta előhívás során.

Az alábbi ábra azt demonstrálja, hogy előhívtam a 3. parancsot a történettárból. Az F7-es történettár ezek után ezt a parancsot nem jeleníti meg újra, azt mutatja, hogy ezt a 3. parancsot egyszer hajtottam végre. Ezzel szemben a `get-history` kiírta, mint 5. végrehajtást:



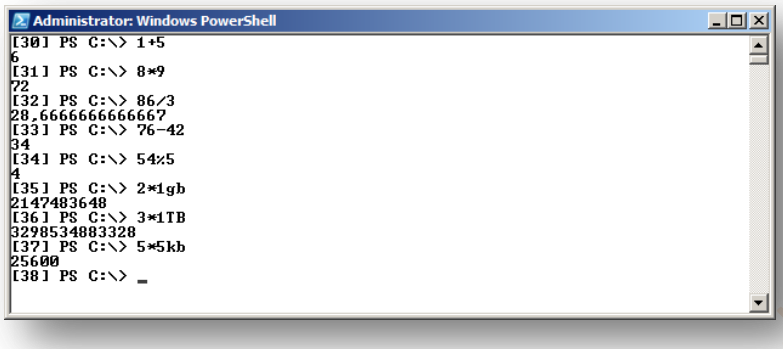
11. ábra A megismételt parancsok nem jelennek meg a történettárban

A felugró parancstörténetből Enterrel lehet végrehajtatni a kijelölt korábbi parancsot, de gyakran arra van szükségünk, hogy egy régebbi parancsot kicsit módosítva hajtsuk újra végre. Ekkor ne Entert, hanem a jobbnyíl billentyűt nyomjuk meg.

A könyv tanfolyami felhasználása nem engedélyezett!

1.2.9 A PowerShell, mint számológép

A PowerShell alaphelyzetben számológépként is működik:



12. ábra Számolási műveletek a PowerShell ablakban

Látszik a fenti ábrán, hogy támogatja az alapműveleteket és a leggyakoribb számítástechnikában előforduló mértékegységeket: `kb`, `mb`, `gb`, `tb`. Ezeket számmal egybeírva kell használni, különben hibajelzést kapunk. A % a maradékos osztást végzi.

1.2.10 A konzol törlése

Ha már jól teleírtuk a PowerShell ablakot, és szeretnénk, ha üres lenne újra, akkor futtassuk a `Clear-Host` cmdletet. Ennek van két gyakran használt álneve is: `cls`, `clear`.

1.2.11 Kis-nagybetű

A PowerShell a legtöbb esetben nem különbözteti meg a kis- és nagybetűket. A parancsokat, kifejezéseket akár kis, akár nagybetűkkel beírhatjuk. De még az általunk létrehozott változónevek tekintetében is figyelmen kívül hagyja ezt:

```
[2] PS I:\>$kis = 1
[3] PS I:\>$KIS
1
[4] PS I:\>get-command get-help

CommandType     Name                              Definition
-----------     ----                              ----------
Cmdlet          Get-Help                          Get-Help [[-Name] <String>...


[5] PS I:\>GET-COMMAND get-HELP

CommandType     Name                              Definition
-----------     ----                              ----------
Cmdlet          Get-Help                          Get-Help [[-Name] <String>...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Az összehasonlításoknál sincs alaphelyzetben szerepe a betű méretének, természetesen itt majd ezt az alapműködést felülbírálhatjuk, hogy a kis-nagybetűket tekintse különbözőnek:

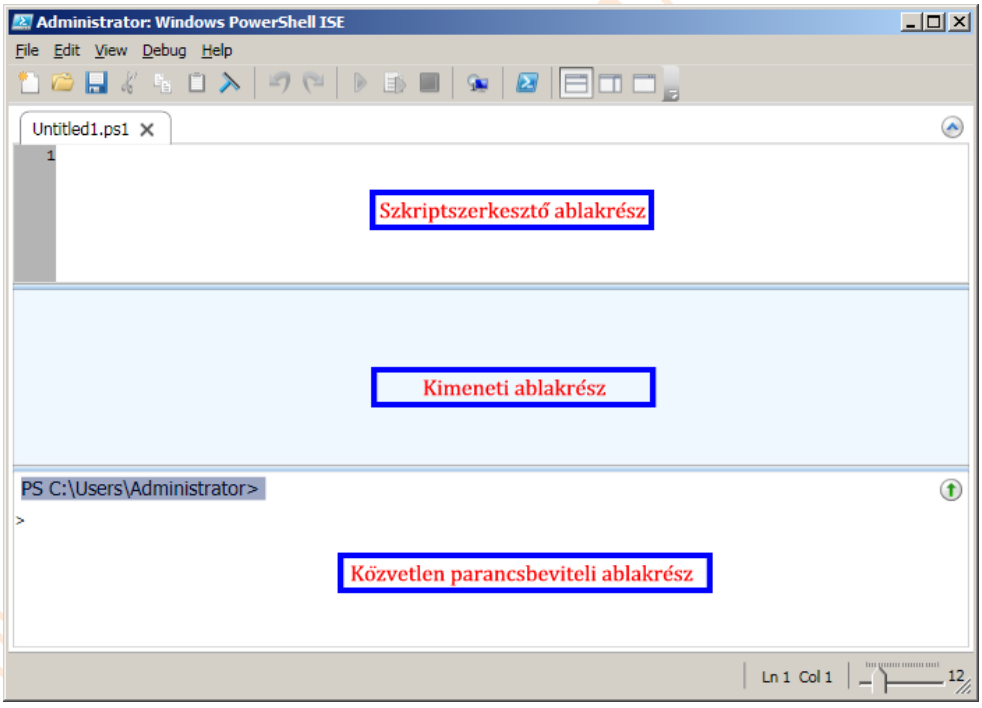
```
[6] PS I:\>"alaphelyzet" -eq "ALAPHELYZET"
True
[7] PS I:\>"kis-nagybetű érzékeny" -ceq "kis-NAGYbetű ÉrZéKeNy"
False
```

A fenti példában az alaphelyzet szerinti egyenlőséget vizsgáló operátor nem kis-nagybetű érzékeny, ha érzékeny változatot szeretnénk használni, akkor az a „c” előtaggal (azaz a „case-sensitive”) külön jelezni kell. Az összehasonlítás műveletéről az *1.5 Operátorok* fejezetben lehet bővebben olvasni.

1.2.12 A grafikus PowerShell felület – Integrated Scripting Environment

Nagyon sok esetben elég nekünk a karakter alapú konzolablak, de többsoros szkriptek írásához azért még mindig jobb valami grafikus szkriptszerkesztő. Eddig csak külső gyártóktól juthattunk ilyenhez, most a PowerShell 2.0-ás verziójában már kapunk egyet beépítetten is.

A Windows PowerShell ISE parancsikonnal indítható a grafikus szerkesztőfelület:

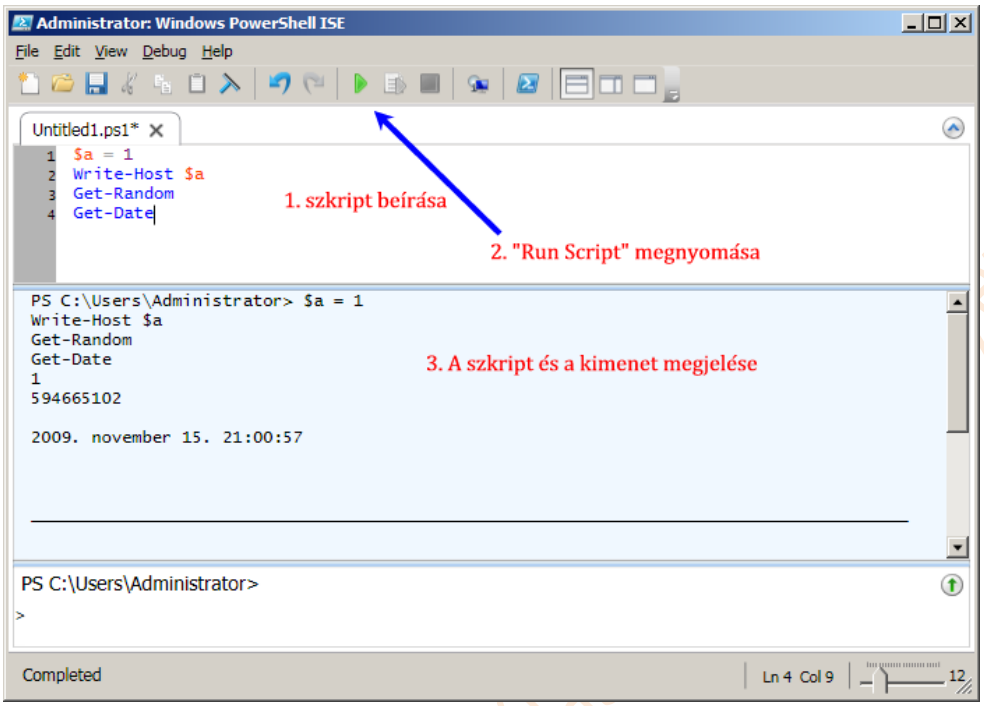


13. ábra A PowerShell ISE

Amint az ábrán is látható, három ablakrészből áll össze a felület: a legfölső fülekkel lapozható módon tartalmazza az általunk begépelt vagy fájlból betöltött szkriptjeinket. A középső részben kapjuk meg a szkriptjeink és parancsaink futtatásának kimenetét, a legalsó ablakrészbe közvetlenül gépelhetünk parancsokat és az Enter lenyomására ezek azonnal végrehajtódnak.

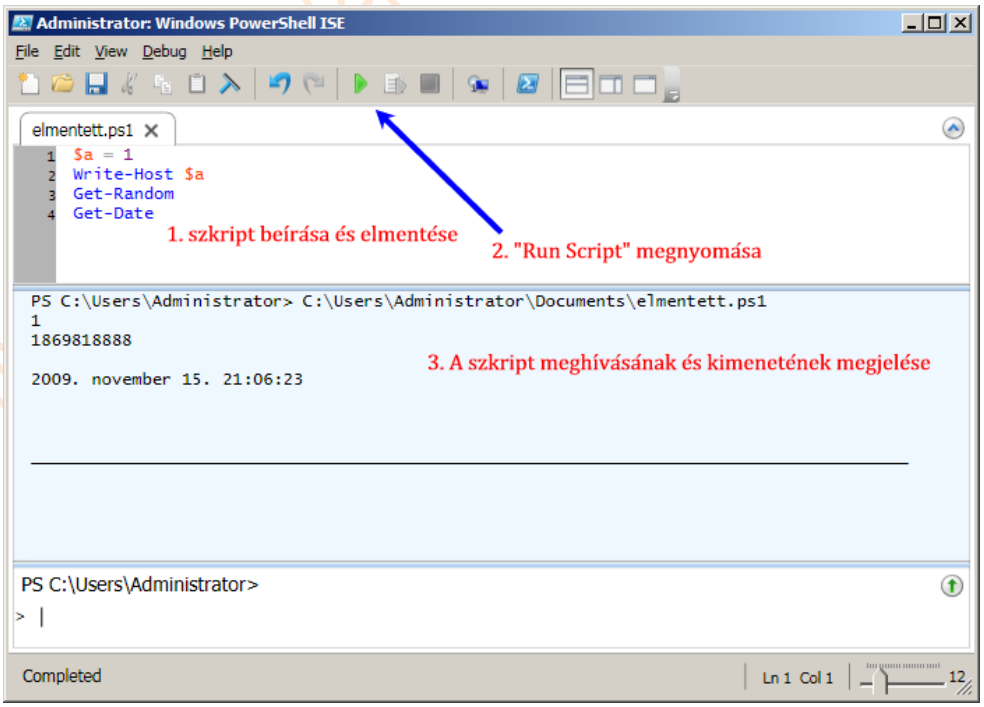
A könyv tanfolyami felhasználása nem engedélyezett!

Ez a program kicsit másképp működik akkor, ha amit a legfelső részbe írunk még nem mentettük el. Ilyenkor az eszközsori futtatás (Run Script) gomb megnyomására az egész általunk begépelt szkript úgy hajtódik végre, mintha alulra gépeltük volna be.



14. ábra Elmentetlen szkript futtatása

Ha elmentettük a szkriptet, akkor csak a szkript meghívása látható a kimeneti ablakban, az egyes sorokat nem kapjuk vissza.

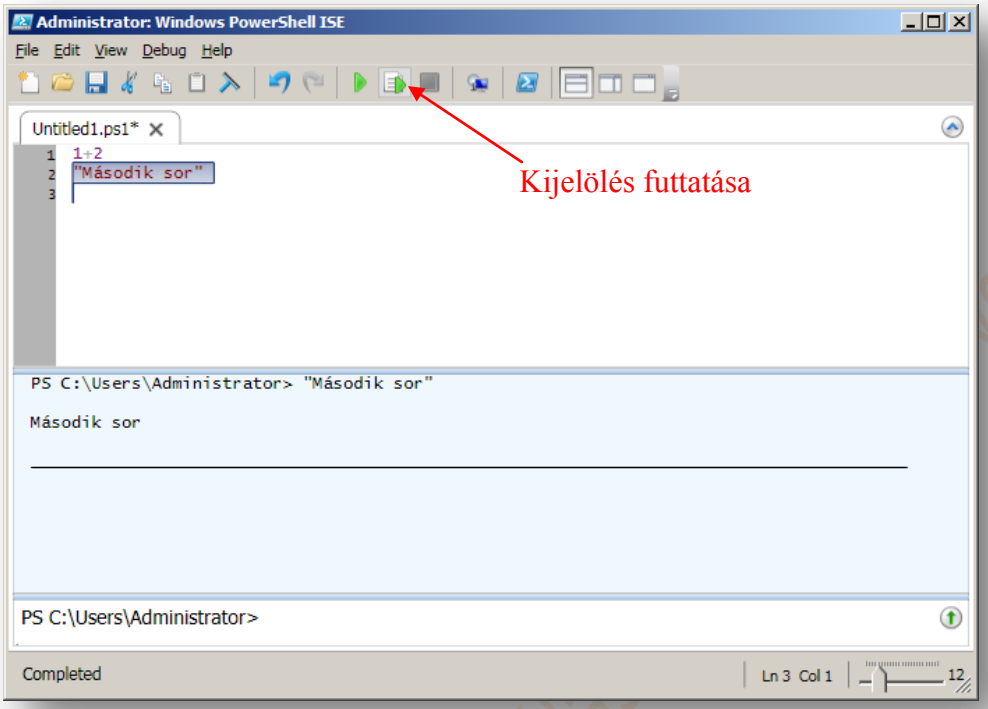


15. ábra Elmentett szkript futtatása

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nagyon praktikus, főleg a szkriptjeink tesztelési fázisában, hogy lehetőség van csak a kijelölt szkriptrészek futtatására is az eszközsor Run Selection gombjával.



16. ábra A kijelölt szkriptrész futtatása

A könyv tanfolyami felhasználása nem engedélyezett!

1.3 Alapfogalmak

Mi is a PowerShell? A Microsoft legújabb definíciója szerint a PowerShell a rendszergazda tevékenységek automatizációs platformja. Kettős szerepe van: egyrészt a PowerShell egy parancssori környezet, azaz egy olyan program, ahol a felhasználó kapcsolatba léphet a számítógépes rendszerrel (direkt nem korlátozva le az operációs rendszerre!). Ebből a szempontból sok olyan szolgáltatást kell nyújtania, ami a gyors, kényelmes adatbevitelt, azaz a gyors gépelést szolgálja. Másrészt a PowerShell egy programnyelv is, mint ilyennek rendelkeznie kell azokkal a programíráshoz szükséges elemekkel, mint például a ciklus, változók, függvények, adatszerkezetek.

A PowerShell mindkét szempontot próbálja a lehető legjobban kielégíteni, támaszkodik más, elsősorban a Unix és Linux különböző shelljeire és onnan átveszi a legjobb megoldásokat. De természetesen ez egy Windowsos shell, tehát nem próbál kompatibilis lenni a Unixos shellekkel, hanem inkább a Windows adottságaihoz és lehetőségeihez alkalmazkodik, ezen belül is a .NET Framework a legfontosabb alapja.

Ellentétben a CMD shellel, a PowerShell önmagában is nagyon sok parancsot tartalmaz. Például CMD shellben a fájlok másolását a copy.com végzi, maga a parancssori környezet nem tud fájlt másolni. Ezzel szemben a PowerShellnek van egy `copy-item` parancsa, un. cmdlet-je, amellyel lehet fájlt (de nem csak fájlt!) másolni.

1.3.1 Architektúra

A PowerShell tehát a .NET Frameworkre épül, a PowerShellhez a 2.0-ás .NET Framework szükséges, de a PowerShell ISE futtatásához szükséges a .NET 3.51 kiegészítő csomag is. Ez a ráépülés számos előnnyel jár, főleg ha azt is tudjuk, hogy a PowerShell objektumokkal dolgozik. Pl. a `dir` parancsot futtatva nem egyszerűen egy karaktersorozatot kapunk vissza válaszként, hanem a fájlok, alkönyvtárak objektumainak halmazát, un. collection-t, gyűjteményt.

Tehát két megjegyzendő fogalom: OBJEKTUM, GYŰJTEMÉNY!

Mi az, hogy objektum? Olyan képződmény, amely tulajdonságokkal (property) és meghívható metódusokkal (method) rendelkezik.

A hagyományos shellek (command.com, cmd.exe, bash, stb.) mindegyikét karakterlánc-orientáltnak nevezhetjük; a parancsok bemenete és kimenete (néhány speciális esetet nem számítva) karakterláncokból áll. Például a cmd.exe esetén a `dir` parancs kimenete valóban és pontosan az a fájl- és mappalistát leíró karaktersorozat, amely a képernyőn megjelenik, se több, se kevesebb. Ez bizonyos szempontból jó, hiszen így azt látjuk, amit kapunk. Másrészről viszont csak azt kapjuk, amit látunk, ez pedig sajnos nem túl sok. A PowerShell adatkezelése azonban alapjaiban különbözik ettől; az egyes parancsok bemenete és kimenete is .NET osztályokon alapuló objektumokból (objektumreferenciákból) áll, vagyis a kiírtakon kívül tartalmazza a .NET objektum számtalan egyéb tulajdonságát is. Amint láthattuk, a `dir` parancs kimenete itt is a képernyőre kiírt fájl és mappalistának tűnik, de ez csak egyszerű érzéki csalódás, a kimenet valójában egy `System.IO.DirectoryInfo` és `System.IO.FileInfo` osztályú objektumokból álló gyűjtemény. Mivel azonban az objektumok nem alkalmasak emberi fogyasztásra, képernyőn való megjelenítésük természetesen szöveges formában, legfontosabb (illetve kiválasztott) jellemzőik felsorolásával történik.

A fentiekből nyilvánvalóan következik az objektumorientált megközelítés két fontos előnye; egyrészt a parancsok kimenete rengeteg információt tartalmaz, amelyeket szükség esetén felhasználhatunk, másrészt nincs szükség szövegfeldolgozó parancsok és eszközök használatára, ha a kimenetként kapott adathalmazból

A könyv tanfolyami felhasználása nem engedélyezett!

A könyv tanfolyami felhasználása nem engedélyezett! © Soós Tibor – soost@ IQJB.hu

csak egy meghatározott információdarabra van szükségünk, mivel minden egyes adatmező egyszerűen a nevére való hivatkozással, önállóan is lekérdezhető. Képzeljük el például, hogy a `dir` parancs által produkált fenti listából szövegfeldolgozó parancsokkal kell kiválasztanunk azokat a fájlokat (mappákat nem), amelyek 2007. április 12-e után módosultak. Jaj. Persze megoldható a feladat, de helyesen fog lefutni az a szkript egy német nyelvű Windowson is? (És akkor például a kínai nyelvű rendszerekről még nem is beszéltünk.) Ha viszont a kimenet objektumokból áll, akkor természetesen egyáltalán nincs szükség a megjelenő szöveg feldolgozására, mert minden egyes objektum „tudja" magáról, hogy ő fájl, vagy mappa, és az operációs rendszer nyelvétől függetlenül, `DateTime` objektumként adja vissza az utolsó módosításának idejét.

Most egyelőre - különösebb magyarázat nélkül - nézzük, hogy egy fájl milyen objektumjellemzőkkel, tagjellemzőkkel (member) bír, a `get-member` cmdlet futtatásával:

```
[2] PS C:\> get-item szamok.txt | get-member


   TypeName: System.IO.FileInfo

Name                       MemberType     Definition
----                       ----------     ----------
AppendText                 Method         System.IO.StreamWriter AppendTe...
CopyTo                     Method         System.IO.FileInfo CopyTo(Strin...
Create                     Method         System.IO.FileStream Create()
...
Delete                     Method         System.Void Delete()
...
MoveTo                     Method         System.Void MoveTo(String destF...
...
Attributes                 Property       System.IO.FileAttributes Attrib...
CreationTime               Property       System.DateTime CreationTime {g...
CreationTimeUtc            Property       System.DateTime CreationTimeUtc...
Directory                  Property       System.IO.DirectoryInfo Directo...
DirectoryName              Property       System.String DirectoryName {get;}
Exists                     Property       System.Boolean Exists {get;}
Extension                  Property       System.String Extension {get;}
FullName                   Property       System.String FullName {get;}
IsReadOnly                 Property       System.Boolean IsReadOnly {get;...
LastAccessTime             Property       System.DateTime LastAccessTime ...
LastAccessTimeUtc          Property       System.DateTime LastAccessTimeU...
LastWriteTime              Property       System.DateTime LastWriteTime {...
LastWriteTimeUtc           Property       System.DateTime LastWriteTimeUt...
Length                     Property       System.Int64 Length {get;}
Name                       Property       System.String Name {get;}
BaseName                   ScriptProperty System.Object BaseName {get=[Sy...
Mode                       ScriptProperty System.Object Mode {get=$catr =...
ReparsePoint               ScriptProperty System.Object ReparsePoint {get...
```

Mint ahogy egy fájltól elvárhatjuk, másolhatjuk (`CopyTo`), törölhetjük (`Delete`) és egy csomó jellemzőjét kiolvashatjuk: utolsó hozzáférés idejét (`LastAccessTime`) és hosszát (`Length`). Azaz ezekhez a tevékenységekhez nem kell külső program, még csak külön PowerShell parancs sem, maga az objektum tudja ezeket.

Pl. a törlés:

```
[3] PS C:\> (get-item szamok.txt).Delete()
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Visszatérve pl. a `dir` parancs futtatására, általában egy könyvtárban nem egy fájl vagy alkönyvtár van, hanem több. Ebben az esetben a PowerShell a parancs végrehajtása során egy un. collection-t, azaz egy objektumhalmazt, objektumgyűjteményt vagy más kifejezéssel objektumtömböt kapunk vissza. Ennek a halmaznak ráadásul nem feltétlenül kell egyforma típusú elemekből állnia, mint ahogy egy könyvtárban is lehetnek fájlok is és alkönyvtárak is. A PowerShell az ilyen collection-ök, azaz gyűjtemények feldolgozására nagyon sok praktikus, nyelvi szintű támogatást ad, amit a későbbiekben látni fogunk.

1.3.2 Command és cmdlet

A PowerShellben a parancsokat cmdlet-nek (kommandlet) hívjuk. Hogy miért? Valószínű megkülönböztetésül a DOS-os parancsoktól, amelyek sok esetben valójában külső programok meghívását jelentik (pl.: netsh → netsh.exe, attrib → attrib.exe). Lényeg, hogy a cmdletek a PowerShell „sajátjai”, egységes szerkezetet alkotnak „ige-főnév” formában. Ezeket a `get-command` cmdlettel ki is listázhatjuk:

```
[1] PS C:\> Get-Command

CommandType     Name                                Definition
-----------     ----                                ----------
Alias           %                                   ForEach-Object
Alias           ?                                   Where-Object
Function        A:                                  Set-Location A:
Alias           ac                                  Add-Content
Cmdlet          Add-Computer                        Add-Computer [-DomainName] ...
Cmdlet          Add-Content                         Add-Content [-Path] <String...
Cmdlet          Add-History                         Add-History [[-InputObject]...
Cmdlet          Add-Member                          Add-Member [-MemberType] <P...
Cmdlet          Add-PSSnapin                        Add-PSSnapin [-Name] <Strin...
Cmdlet          Add-Type                            Add-Type [-TypeDefinition] ...
Alias           asnp                                Add-PSSnapIn
Function        B:                                  Set-Location B:
Function        C:                                  Set-Location C:
Alias           cat                                 Get-Content
Alias           cd                                  Set-Location
Function        cd..                                Set-Location ..
Function        cd\                                 Set-Location \
Alias           chdir                               Set-Location
Cmdlet          Checkpoint-Computer                 Checkpoint-Computer [-Descr...
Alias           clc                                 Clear-Content
…
```

A PowerShell 2.0-ban a `Get-Command` nem csak az „igazi” cmdleteket adja vissza, hanem az „álneveket”, függvényeket is, így ha csak a cmdletekre vagyunk kíváncsiak, akkor a következő formában kell írni a parancsunkat:

```
[2] PS C:\> Get-Command -CommandType cmdlet

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Add-Computer                        Add-Computer [-DomainName] ...
Cmdlet          Add-Content                         Add-Content [-Path] <String...
Cmdlet          Add-History                         Add-History [[-InputObject]...
Cmdlet          Add-Member                          Add-Member [-MemberType] <P...
Cmdlet          Add-PSSnapin                        Add-PSSnapin [-Name] <Strin...
Cmdlet          Add-Type                            Add-Type [-TypeDefinition] ...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Cmdlet          Checkpoint-Computer                Checkpoint-Computer [-Descr...
Cmdlet          Clear-Content                      Clear-Content [-Path] <Stri...
Cmdlet          Clear-EventLog                     Clear-EventLog [-LogName] <...
Cmdlet          Clear-History                      Clear-History [[-Id] <Int32...
Cmdlet          Clear-Item                         Clear-Item [-Path] <String[...
Cmdlet          Clear-ItemProperty                 Clear-ItemProperty [-Path] ...
Cmdlet          Clear-Variable                     Clear-Variable [-Name] <Str...
Cmdlet          Compare-Object                     Compare-Object [-ReferenceO...
…
```

Említettem, hogy a cmdletek a PowerShell sajátjai, tehát nem külső programok, de ez természetesen nem azt jelenti, hogy külső programokat nem lehet meghívni. És azt sem jelenti, hogy csak a PowerShell telepítésével létrejövő cmdleteket használhatjuk, a Windows különböző verzióival és egyéb Microsoft szoftverekkel más és más kiegészítő modulokat telepíthetünk, vagy különböző gyártók által (pl.: PowerShell Community Extensions: http://www.codeplex.com/PowerShellCX) készített cmdleteket is beépíthetünk a PowerShellbe, de innentől kezdve ezek a cmdletek is a PowerShell sajátjaivá válnak.

Természetesen arra is lehetőség van, hogy magunk hozzunk létre cmdleteket, de ennek csak azon vonatkozásaival fogunk foglalkozni, amit a PowerShell keretein belül meg tudunk oldani. A dll-ek készítése és Visual Studio segítségével létrehozandó bővítmények olyan mélyen bevinnének minket az igazi programozók veszélyekkel teli földjére, hogy ebben könyvben nem foglakozunk a témával. Akit mégis érdekel a dolog, és elegendő bátorságot is érez magában, az például elolvashatja a *Professional Windows PowerShell Programming - Snap-ins, Cmdlets, Hosts, and Providers* című könyvet, ami a Wiley Publishing, Inc. gondozásában jelent meg 2008-ban angol nyelven.

Ezen kívül a PowerShellben vannak kifejezések és egyéb nyelvi elemek, amelyekkel a következőkben foglalkozom.

1.3.3 Segítség! (Get-Help)

A PowerShell kiterjedt belső súgórendszerrel rendelkezik, a súgótémák teljes listáját a `get-help` paranccsal kérhetjük le (itt most helytakarékosságból csak az 'a' betűvel kezdődő témákat):

```
[5] PS C:\> get-help a*

Name                                Category  Synopsis
----                                --------  --------
ac                                  Alias     Add-Content
asnp                                Alias     Add-PSSnapin
Add-History                         Cmdlet    Appends entries to the session ...
Add-PSSnapin                        Cmdlet    Adds one or more Windows PowerS...
Add-Member                          Cmdlet    Adds a user-defined custom memb...
Add-Type                            Cmdlet    Adds a Microsoft .NET Framework...
Add-Content                         Cmdlet    Adds content to the specified i...
Add-Computer                        Cmdlet    Add the local computer to a dom...
Alias                               Provider  Provides access to the Windows ...
…
```

A megjelenő listában szerepel valamennyi alias, cmdlet, provider és számos olyan súgótéma, amely nem konkrét utasításhoz, hanem valamely PowerShellbeli fogalomhoz köthető. A lista egyes elemeihez tartozó súgótéma a következő paranccsal kérhető le:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
get-help <alias, cmdlet, provider, helpfile>
```

Például:

```
[7] PS C:\> get-help Add-Content

NAME
    Add-Content

SYNOPSIS
    Adds content to the specified items, such as adding words to a file.


SYNTAX
    Add-Content [-LiteralPath] <string[]> [-Value] <Object[]> [-Credential <PS
    Credential>] [-Exclude <string[]>] [-Filter <string>] [-Force] [-Include <
    string[]>] [-PassThru] [-Confirm] [-WhatIf] [-UseTransaction] [<CommonPara
    meters>]

    Add-Content [-Path] <string[]> [-Value] <Object[]> [-Credential <PSCredent
…
```

Ha a `Get-Help` helyett a `help` függvényt használjuk, akkor a szöveget oldalakra tördelve láthatjuk.

Ha nem tudjuk a keresett funkcióhoz tartozó cmdlet teljes nevét, megtippelhetjük a kéttagú kifejezés egyik felét, vagy bármilyen töredékét, és ez alapján is kérhetünk segítséget. Az alábbi utasítás például kilistázza valamennyi Service végződésű parancsot:

```
[8] PS C:\> get-help *-service

Name                              Category  Synopsis
----                              --------  --------
Get-Service                       Cmdlet    Gets the services on a local or...
Stop-Service                      Cmdlet    Stops one or more running servi...
Start-Service                     Cmdlet    Starts one or more stopped serv...
Suspend-Service                   Cmdlet    Suspends (pauses) one or more r...
Resume-Service                    Cmdlet    Resumes one or more suspended (...
Restart-Service                   Cmdlet    Stops and then starts one or mo...
Set-Service                       Cmdlet    Starts, stops, and suspends a s...
New-Service                       Cmdlet    Creates a new Windows service.
```

A megjelenő listából a pontos név és a rövid leírás alapján már könnyen kiválaszthatjuk a megfelelőt. Az egyes cmdletek súgóoldalai négy különböző nézetben is megjeleníthetők. A nézetet a `Get-Help` paramétereivel választhatjuk ki. Alapértelmezés szerint egy rövidített, gyors áttekintésre alkalmas változat jelenik meg. A `-detailed` paraméter használatával egy bővebb változat, az `-example` paraméter hatására csak a mintapéldák (ezek nagyon hasznosak lehetnek), a `-full` paraméter segítségével pedig teljes súgótéma jeleníthető meg.

A PowerShellben nagyon sok fogalom is van, szerencsére ezekre is van súgó-téma. Ezeket `about_` előtaggal látták el:

```
[9] PS C:\> get-help about_*

Name                              Category  Synopsis
----                              --------  --------
about_aliases                     HelpFile  Describes how to use alternate ...
```

A könyv tanfolyami felhasználása nem engedélyezett!

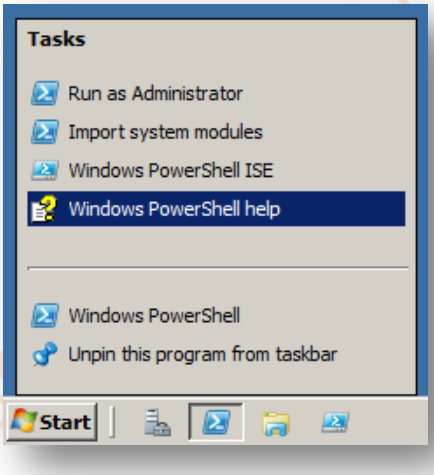
© Soós Tibor – soost@ IQJB.hu

```
about_Arithmetic_Operators        HelpFile  Describes the operators that pe...
about_arrays                      HelpFile  Describes a compact data struct...
about_Assignment_Operators        HelpFile  Describes how to use operators ...
about_Automatic_Variables         HelpFile  Describes variables that store ...
about_Break                       HelpFile  Describes a statement you can u...
about_command_precedence          HelpFile  Describes how Windows PowerShel...
about_Command_Syntax              HelpFile  Describes the notation used for...
about_Comment_Based_Help          HelpFile  Describes how to write comment-...
…
```

Érdemes ezeket is elolvasni, mert gyakran olyan referenciainformációk is fellelhetők bennük, amelyek ilyen „tanító” jellegű könyvekben nem szerepelnek.

1.3.4 A grafikus help

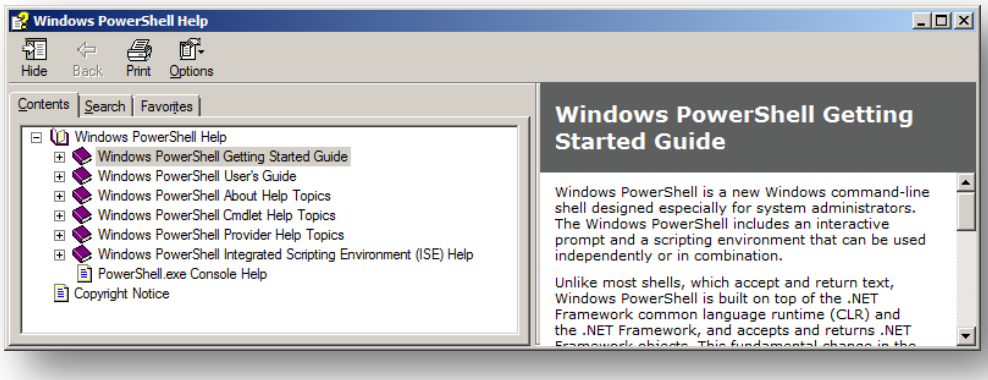
Ha valaki nem szeret karakteres konzolon olvasgatni, akkor elkészítették ezen súgótémák grafikus változatát is, amit a PowerShell ISE környezetből egyszerűen elérhetünk a Help menüből, a `C:\Windows\Help\mui\0409\WindowsPowerShellHelp.chm` helyről, vagy a tálcára kitűzött PowerShell ikonra jobb egérgombbal kattintva megjelenő menüből is indítható:



17. ábra A grafikus súgó ikonja a tálca PowerShell ikonjánál

Ezzel nem csak könnyebben olvasható szöveget kapunk, hanem a teljes szöveges keresés és a témák közti navigáció is sokkal egyszerűbbé válik.

A könyv tanfolyami felhasználása nem engedélyezett!



18. ábra A grafikus súgó

1.3.5 Cmdletek paraméterezése

A PowerShell cmdletjeinek átadhatunk paramétereket, mint ahogy azt többször láthattuk a megelőző példákban. Nézzünk például egy cmdlet paraméterezését:

```
[17] PS C:\> (get-help write-host).parameters

    -BackgroundColor <ConsoleColor>
        Specifies the background color. There is no default.

        Required?                    false
        Position?                    named
        Default value                None
        Accept pipeline input?       false
        Accept wildcard characters?  false

    -ForegroundColor <ConsoleColor>
        Specifies the text color. There is no default.

        Required?                    false
        Position?                    named
        Default value                None
        Accept pipeline input?       false
        Accept wildcard characters?  false

    -NoNewline [<SwitchParameter>]
        Specifies that the content displayed in the console does not end
        with a newline character.

        Required?                    false
        Position?                    named
        Default value                None
        Accept pipeline input?       false
        Accept wildcard characters?  false

    -Object <Object>
        Objects to display in the console.

        Required?                    false
        Position?                    1
        Default value                None
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
        Accept pipeline input?       true (ByValue)
        Accept wildcard characters?  false

    -Separator <Object>
        String to the output between objects displayed on the console.

        Required?                    false
        Position?                    named
        Default value                None
        Accept pipeline input?       false
        Accept wildcard characters?  false

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer and OutVariable. For more information, type,
        "get-help about_commonparameters".
```

A fenti példában a `Write-Host` cmdlet súgójának csak a paraméterekkel kapcsolatos részét írattam ki. A paraméterek nevét kiírhatjuk, ha egyértelművé akarjuk tenni, hogy az adott értéket melyik paraméternek szánjuk. A paraméterek nevét egy '-' (kötőjellel) kell bevezetni. Vannak olyan paraméterek, amelyek „pozícionális" paraméterek, azaz nem muszáj kiírni a nevüket, elég rögtön a paraméter értékét beírni:

```
[18] PS C:\> Write-Host "Ezt akarom kiírni" -ForegroundColor yellow -Backg
roundColor red
Ezt akarom kiírni
```

19. ábra Cmdlet paraméterezése

A fenti példában az „Ezt akarom kiírni" szöveg a súgó alapján az `Object` paraméter lesz, az egyes színek átadását kiírt paraméterekkel adtam meg, mert azok nem pozícionálisak.

Nézzük milyen változatai vannak a paraméterezésnek:

```
[21] PS C:\> Write-Host -obj "Ezt akarom kiírni" -f yellow -b red
Ezt akarom kiírni
```

20. ábra Másik paraméter-megadási lehetőség

Látható, hogy akár ki is írhatjuk a pozícionális paraméterek nevét, ráadásul elég csak annyi karaktert megadni, ami már egyértelműen azonosítja a paramétert. Az `Object` paraméterhez nem elég egy darab 'o' betűt megadni, mert a „`CommonParameters`" között van `OutBuffer` és `OutVariable` is. Az 'Ob' sem elég, mert – bár nem mutatja a help – az `OutBuffer` paraméternek az `Ob` egy álneve, szinonimája.

Ezzel szemben a színeknél az 'f' és 'b' egyértelműen azonosítja, hogy melyik paraméterre gondolunk.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.3.6 Ki-mit-tud (Get-Member)

Korábban már megállapítottuk, hogy a PowerShellben minden objektum, és mint ilyen, lekérdezhető tulajdonságok és meghívható metódusok jól meghatározott készletével rendelkezik. Az egyes objektumok képességeinek felderítésére a `Get-Member` cmdlet szolgál. Korábban, a nem létező parancsok kipróbálásakor már találkozhattunk az alábbi példával:

```
PS C:\> 1
1
```

Mi is történt itt? A beírt szám ebben az esetben nyilvánvalóan nem parancs, hanem mi is lehet? Természetesen egy integer típusú konstans. Ha ezt a konstanst odaadjuk a `Get-Member` cmdletnek, az megmondja, hogy az integer (ami egy érték-típus) milyen tulajdonságokkal és képességekkel rendelkezik (bármilyen szám jó☺) :

```
PS C:\> 1 | Get-Member


   TypeName: System.Int32

Name         MemberType Definition
----         ---------- ----------
CompareTo    Method     System.Int32 CompareTo(Int32 value), System.Int3...
Equals       Method     System.Boolean Equals(Object obj), System.Boolea...
GetHashCode Method     System.Int32 GetHashCode()
GetType      Method     System.Type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode()
ToString     Method     System.String ToString(), System.String ToString...
```

A megjelenő listából látható, hogy a típust tartalmazó struktúra neve `System.Int32`, és ugyan nem rendelkezik tulajdonságokkal (egyetlen tulajdonsága az értéke), de van néhány meghívható metódusa: vissza tudja adni a típusát, át tudja alakítani magát karakterlánccá, összehasonlítható egy másik számmal, stb. A listában szerepelnek az adott metódus különféle paraméterezéssel meghívható (overloaded) változatai, és az egyes változatok által visszaadott érték típusa is.

Próbáljuk ki, mit tud egy karakterlánc!

```
[24] PS C:\> "Helló" | Get-Member


   TypeName: System.String

Name             MemberType           Definition
----             ----------           ----------
Clone            Method               System.Object Clone()
CompareTo        Method               int CompareTo(System.Object val...
Contains         Method               bool Contains(string value)
…
PadLeft          Method               string PadLeft(int totalWidth),...
PadRight         Method               string PadRight(int totalWidth)...
Remove           Method               string Remove(int startIndex, i...
Replace          Method               string Replace(char oldChar, ch...
Split            Method               string[] Split(Params char[] se...
…
ToUpper          Method               string ToUpper(), string ToUppe...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
…
Chars             ParameterizedProperty char Chars(int index) {get;}
Length            Property              System.Int32 Length {get;}
```

Ez a lista már jóval bővebb, csak győzzünk válogatni. Akinek esetleg feltűnt korábban, hogy a PowerShellben egyetlen karakterlánc-feldolgozásra képes utasítás sincsen, most megkaphatja a magyarázatot: nincs szükség ilyesmire, mert a .NET Framework `String` osztálya mindent tud, ami ebben a témában felmerülhet.

A következőkben kipróbálunk néhány egyszerű karakterlánc-műveletet.

- Bővítsük a beírt karakterláncot (vagyis a „Helló” karakterláncra hívjuk meg a `String` osztály `Insert` metódusát, első paraméter a pozíció, második a beszúrandó karakterlánc):

```
PS C:\> "Helló".Insert(5, " világ!")
Helló világ!
```

- Cseréljünk ki benne egy betűt egy másikra (a karaktereket idézőjelek vagy aposztrófok között adhatjuk meg):

```
PS C:\> "Helló".Replace('e', 'a')
Halló
```

- Alakítsuk át a karakterláncot csupa nagybetűssé:

```
PS C:\> "Helló".ToUpper()
HELLÓ
```

- Számoljuk meg a karakterlánc betűit (nem metódus, hanem tulajdonság, így nem kell zárójel):

```
PS C:\> "Helló".Length
5
```

A `Get-Member` lehet a segítségünkre akkor is, ha egy adott cmdlet kimenetét szeretnénk közelebbről megvizsgálni, megmondja milyen típusú objektumokból áll a kimenet, és azoknak milyen tulajdonságai, metódusai vannak. Ha a kimenet nem egyetlen objektum, hanem gyűjtemény (legtöbbször ez a helyzet), akkor abban akár több különböző típusú objektum is előfordulhat, de a `Get-Member` ebben az esetben sem jön zavarba; a gyűjteményben lévő valamennyi típust kilistázza. A mappákra kiadott `Get-ChildItem` például szokásosan `FileInfo` és `DirectoryInfo` objektumokat is tartalmaz, ekkor a `Get-Member` kimenetében mindkét típus megtalálható.

```
PS C:\> Get-ChildItem | Get-Member


   TypeName: System.IO.DirectoryInfo

Name                      MemberType     Definition
----                      ----------     ----------
Create                    Method         System.Void Create(), System.V...
CreateObjRef              Method         System.Runtime.Remoting.ObjRef...
...
   TypeName: System.IO.FileInfo
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name                           MemberType     Definition
----                           ----------     ----------
AppendText                     Method         System.IO.StreamWriter AppendT...
CopyTo                         Method         System.IO.FileInfo CopyTo(Stri...
...
```

A .NET objektumokkal, metódusokkal, tulajdonságokkal és adattípusokkal kapcsolatos információt a .NET Framework SDK dokumentációjából kaphatunk, itt igen részletes leírást találhatunk valamennyi osztály valamennyi elemével kapcsolatban. Egyszerűbb esetben azonban nem érdemes a rettenetes mennyiségű információ közötti keresgéléssel sokkolni magunkat, némi segítség közvetlenül a PowerShellből is hozzáférhető az alábbi módszer használatával (egy karakterlánc objektumra meghívható `Replace()` metódus definícióját kérjük le, a kimenetben látható a visszaadott érték, és a lehetséges paraméterlisták):

```
[28] PS C:\> ("Helló" | Get-Member replace).definition
string Replace(char oldChar, char newChar), string Replace(string oldValue, st
ring newValue)
```

Ugyanez (legalábbis az eleje) persze megjelenik egy „közönséges” `Get-Member` hívás kimenetében is („`Definition`” oszlop), de ott a szűkös hely miatt a hosszabb szövegek vége általában már nem látható.

1.3.7 Alias, becenév, álnév

Ahogy a fejezet bevezetőjében említettem, a PowerShell kettős célt szolgál: jó shell és jó programnyelv próbál lenni. A jó shellre az jellemző, hogy kényelmes, könnyen kezelhető, keveset kell gépelni, „fél szavakból” is megérti a felhasználót. Ennek eszköze a néhány betűből álló alias (becenevek, álnevek, szinonimák) használatának lehetősége. Álnevet kapcsolhatunk (akár többet is) különböző PowerShell elemekhez. Az alias bármilyen parancsban, kifejezésben teljes értékű helyettesítője gazdájának, így nincs szükség az eredeti név begépelésére. Az álnevek természetesen szkriptekben is korlátozás nélkül használhatók, de alkalmazásuk jelentősen ronthatja a szkript olvashatóságát[4].

Álnevet a következő elemekhez rendelhetünk:

- PowerShell függvények
- PowerShell szkriptek
- Fájlok
- Bármilyen végrehajtható állomány (exe, com, cmd, vbs, stb.)

A PowerShell számos beépített álnévvel is rendelkezik, ezek közül néhánnyal (`dir`, `cd`, stb.) már a korábbi fejezetekben is találkozhattunk. Az álnevek teljes listáját a `Get-Alias` cmdlet jeleníti meg:

```
PS C:\> Get-Alias

CommandType     Name                           Definition
-----------     ----                           ----------
Alias           ac                             Add-Content
Alias           asnp                           Add-PSSnapin
```

[4] Ugyancsak a jobb olvashatóság miatt nem nagyon használunk aliasokat a könyvben szereplő mintapéldákban és szkriptekben sem.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Alias           clc                             Clear-Content
Alias           cli                             Clear-Item
Alias           clp                             Clear-ItemProperty
Alias           clv                             Clear-Variable
...
```

Rendelkezésünkre áll továbbá az „Alias:” meghajtó is, amely egy más módon teszi hozzáférhetővé a beceneveket:

```
PS C:\> Set-Location alias:
PS Alias:\> Get-ChildItem
```

Ha nem a teljes listára, hanem csak egy konkrét aliasra vagyunk kíváncsiak, akkor a következő parancsot használhatjuk:

```
PS C:\> Get-Alias -name dir

CommandType     Name                             Definition
-----------     ----                             ----------
Alias           dir                              Get-ChildItem
```

Kissé bonyolultabb a helyzet, ha az egy adott parancshoz használható aliasokat szeretnénk felderíteni (természetesen ilyenből több is lehet). Az alábbi parancs a Set-Location cmdlethez tartozó álneveket listázza ki[5]:

```
[27] PS C:\> Get-Alias | Where-Object {$_.Definition -eq "Set-Location"}

CommandType     Name                             Definition
-----------     ----                             ----------
Alias           cd                               Set-Location
Alias           chdir                            Set-Location
Alias           sl                               Set-Location
```

A már ismert Get-Alias cmdleten kívül az alábbi listában látható négy további parancs is az aliasokkal kapcsolatos különféle műveletek elvégzésére szolgál.

```
PS C:\> Get-Help *-alias

Name                    Category                Synopsis
----                    --------                --------
Export-Alias            Cmdlet                  Exports information a...
Get-Alias               Cmdlet                  Gets the aliases for ...
Import-Alias            Cmdlet                  Imports an alias list...
New-Alias               Cmdlet                  Creates a new alias.
Set-Alias               Cmdlet                  Creates or changes an...
```

A New-Alias cmdlet segítségével új álneveket definiálhatunk, paraméterként az alias nevét és a vele helyettesítendő parancsot kell megadnunk. Az alábbi parancs a „word” aliast hozza létre, amely a továbbiakban a szövegszerkesztő egyszerű indítását teszi lehetővé (természetesen a winword.exe valódi helyét kell megadnunk):

[5] A parancs egyes részeinek jelentéséről a következő fejezetben lesz szó.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> New-Alias -name word -Value "c:\program files\microsoft
office\office10\winword.exe"
PS C:\> word
```

A létrehozott aliasok csak az adott munkameneten belül élnek, a rendszeresen használni kívánt darabokat a PowerShell profilba kell felvennünk (lásd később). Az `Export-Alias` cmdlet segítségével a teljes alias listát fájlba menthetjük, a fájl pedig (például egy másik számítógépen) az `Import-Alias` parancs segítségével tölthető vissza.

1.3.8 PSDrive, PSProvider

Mivel a PowerShellnek kiterjedt saját parancskészlete van ezért logikus az elképzelés, hogy egy-egy parancs hasonló objektumokon is ugyanabban a formában futtatható lehessen. Például a fájlrendszer és a registry is hasonló mappastruktúra-szerűen épül fel. Ebből jöhetett a PowerShell alkotóinak az az ötlete, hogy kiterjesztették a meghajtó fogalmát, PSmeghajtó lett így a fájlrendszeren kívül a registry, a tanúsítványtár, de a függvényeinket és a környezeti és saját változóinkat is ilyen `PSDrive`-okon keresztül is megnézhetjük.

Ezekhez a különböző `PSDrive`-okhoz tartozik egy-egy provider (ami egy .NET alapú program), ami biztosítja az egységes felületet, amelyhez a PowerShell cmdletjei csatlakozhatnak. A meghajtók (illetve a mögöttük álló providerek) teszik lehetővé, hogy a `Set-Location`, a `Get-ChildItem`**,** stb. parancsok a fájlrendszerben, a registryben, a tanúsítványtárban és még számos más helyen is teljesen azonos módon működhessenek. A provider által támogatott cmdletek mindegyike használható a providerre alapuló meghajtókon, illetve vannak speciálisan egy meghatározott providerhez készült cmdletek is. Az egyes providerek úgynevezett dinamikus paramétereket is adhatnak a cmdletekhez, amelyek csak akkor használhatók, ha a cmdletet az adott provider segítségével létrehozott adatforráson használjuk. A PowerShell rendszerrel kapott providerek készlete tovább bővíthető, így újabb adatforrások (például az Active Directory!) is elérhetővé tehetők a szokásos cmdletek számára (lásd később).

A PowerShell meghajtók tehát olyan logikai adattárak, amelyek a megfelelő provider közreműködésével a fizikai meghajtókhoz hasonló módon érhetők el. A PowerShell meghajtók semmilyen módon nem használhatók a shell környezeten kívülről, de természetesen a fizikai és hálózati meghajtók csatlakoztatását, illetve eltávolítását a shell is érzékeli.

A PSmeghajtók kilistázásához használhatjuk a `Get-PSDrive` cmdletet, egy ilyen drive-on belül érvényes például a `get-childitem` parancs, azaz alias `dir`:

```
[1] PS C:\> Get-PSDrive

WARNING: column "CurrentLocation" does not fit into the display and was r
emoved.

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
A                                      FileSystem    A:\
Alias                                  Alias
C                   8,64         71,26 FileSystem    C:\
cert                                   Certificate   \
D                                      FileSystem    D:\
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
HKLM                                    Registry      HKEY_LOCAL_MACHINE
Variable                                Variable
WSMan                                   WSMan

[2] PS C:\> dir hkcu:


    Hive: HKEY_CURRENT_USER


SKC  VC Name                            Property
---  -- ----                            --------
  2   0 AppEvents                       {}
  0  36 Console                         {ColorTable00, ColorTable01, Co...
 11   0 Control Panel                   {}
  0   2 Environment                     {TEMP, TMP}
  3   0 Keyboard Layout                 {}
  0   0 Network                         {}
  3   0 Printers                        {}
  5   0 Software                        {}
  1   0 System                          {}
  1   9 Volatile Environment            {LOGONSERVER, USERDNSDOMAIN, US...

[3] PS C:\> dir cert:\CurrentUser

Name : SmartCardRoot
Name : UserDS
Name : AuthRoot
Name : CA
Name : Trust
Name : Disallowed
Name : My
Name : Root
Name : TrustedPeople
Name : ACRS
Name : TrustedPublisher
```

A PSProvidereket is kilistázhatjuk a `get-psprovider` cmdlet segítségével:

```
[12] PS C:\> Get-PSProvider

Name                 Capabilities               Drives
----                 ------------               ------
WSMan                Credentials                {WSMan}
Alias                ShouldProcess              {Alias}
Environment          ShouldProcess              {Env}
FileSystem           Filter, ShouldProcess      {C, A, D}
Function             ShouldProcess              {Function}
Registry             ShouldProcess, Transact... {HKLM, HKCU}
Variable             ShouldProcess              {Variable}
Certificate          ShouldProcess              {cert}
```

Ebben a listában láthatjuk, hogy az egyes providerek milyen képességekkel bírnak. Az összes lehetséges képességet le is kérdezhetjük[6]:

```
[13] PS C:\> [enum]::getValues("System.Management.Automation.Provider.ProviderC
```

[6] Ettől a kifejezéstől most ne ijedjen meg senki, később világossá válik a jelentése

```
apabilities")
None
Include
Exclude
Filter
ExpandWildcards
ShouldProcess
Credentials
Transactions
```

- **None**: az adott provider semmilyen újabb képességgel nem rendelkezik az alap PowerShell provider képességekhez képest.
- **Include, Exclude**: elemekre történő hivatkozás lehetősége dzsókerkarakterekkel (*, ?, stb.) . Az `Include` jelzi, hogy milyen elemekre szeretnénk kiterjeszteni a parancsunk hatását, az `Exclude`, hogy milyenekre nem. Természetesen a PowerShell akkor is támogathatja a dzsókerkarakterek használatát, ha maga a provider nem. Ilyenkor „saját hatáskörben” végzi el a szükséges műveleteket. Ha a provider támogatja ezeket a képességeket, akkor teljesítményben kapunk előnyt, azaz gyorsabban hajtódnak végre a műveletek, vagy akár kisebb hálózati forgalmat generál a művelet végrehajtása.
- **Filter**: támogatja a provider-specifikus szűrő kifejezéseket. Igazán „profi” filterekkel majd az Active Directory részben fogunk találkozni.
- **ExpandWildCards**: ez megint csak egy olyan képesség, ami teljesítménynövelő lehet, de azt a funkcionalitást alapban is nyújtja a PowerShell, így jelenleg még egyetlen provider sem rendelkezik ezzel.
- **ShouldProcess**: Ennek birtokában a provider meghívja a `ShouldProcess` metódust, mielőtt módosításokat végezne az adatokon. Így lehetővé válik a `-WhatIf` és a `-Confirm` paraméterek használatával óvatosan megnézni, hogy mi történne ha végrehajtanánk a változtatásokat, illetve szabályozhatjuk, hogy megerősítést kérjen-e minden változtatásnál.
- **Credentials**: Lehetőséget ad, hogy a műveleteket ne a saját nevünkben, hanem valamilyen más felhasználói fiók nevében végezzük el.
- **Tranzactions**: Engedi a tranzakcionált, azaz mindent vagy semmit elv alapján történő végrehajtást. Jelenleg ezt csak a Registry provider tudja, de várható, hogy majd a fájlrendszer is tudni fogja.

1.3.8.1 Meghajtók létrehozása és törlése (New-PSDrive, Remove-PSDrive)

PSDrive létrehozásával a gyakran használt helyek elérését jelentősen megkönnyíthetjük. Ha például szeretnénk készíteni egy `MSReg` nevű PS meghajtót, amely a registry `HKEY_CURRENT_USER\Software\Microsoft` helyére mutat, akkor a `New-PSDrive` paramétereként az új meghajtó nevét, a használandó providert és a gyökér elérési útját kell megadnunk az alábbiak szerint, és máris használhatjuk a frissen létrehozott meghajtót:

```
[13] PS C:\> New-PSDrive -Name MSReg -PSProvider Registry -Root HKEY_CURRE
NT_USER\Software\Microsoft

WARNING: column "CurrentLocation" does not fit into the display and was r
emoved.

Name           Used (GB)     Free (GB) Provider      Root
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
----            ---------     --------- --------      ----
MSReg                                   Registry      HKEY_CURRENT_USER...


[14] PS C:\> dir msreg:


    Hive: HKEY_CURRENT_USER\Software\Microsoft


SKC  VC Name                           Property
---  -- ----                           --------
  1   0 Active Setup                   {}
  1   0 ADs                            {}
  4   0 Advanced INF Setup             {}
  1   0 Assistance                     {}
  0   4 Command Processor              {CompletionChar, DefaultColor, ...
…
```

A létrehozott meghajtó csak az adott PowerShell munkamenetben használható, kilépéskor egyszerűen elvész. A rendszeresen használt meghajtókat mégsem kell minden egyes alkalommal újra létrehoznunk, a megfelelő parancsok beilleszthetők a PowerShell indításakor automatikusan lefutó profil-szkriptbe is (lásd később). Ha a munkameneten belül törölni szeretnénk a létrehozott meghajtót, ezt nagyon egyszerűen megtehetjük, csupán a nevét kell paraméterként megadnunk a `Remove-PSDrive` cmdletnek:

```
PS C:\> Remove-PSDrive MSReg
```

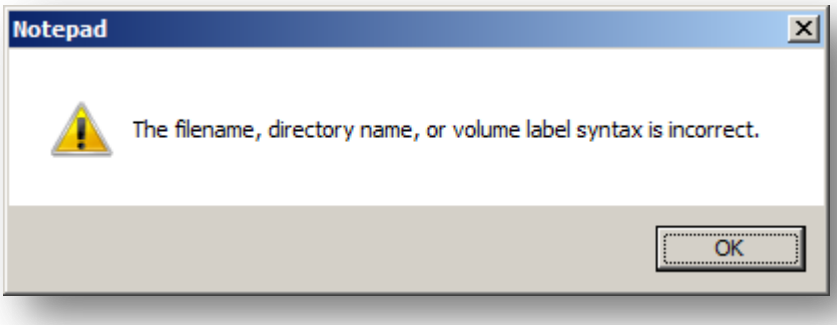
Korábban említettük, hogy a PSmeghajtók csak a PowerShellből érhetők el, a Windows, és más alkalmazások nem ismerik fel az ilyen módon létrehozott útvonalakat. Ha azonban egy külső programot a shellből indítunk el, akkor mégis van mód a korlátozás megkerülésére. Hozzuk létre a „`C:\munka`” mappát és készítsünk benne egy `szöveg.txt` nevű szöveges fájlt tetszőleges tartalommal, majd rendeljük a mappához a scripts nevű PowerShell-meghajtót!

```
[15] PS C:\> New-PSDrive -name munka -PSProvider FileSystem -Root 'C:\munk
a'

WARNING: column "CurrentLocation" does not fit into the display and was r
emoved.

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
munka                            71,26 FileSystem    C:\munka
```

Próbáljuk Notepad segítségével megnyitni a szöveg.txt fájlt!

```
PS C:\> notepad munka:\szöveg.txt
```

Akár tanfolyami felhasználás is lehet engedélyezett!



21. ábra Hibajelzés a Notepadban

Elég rosszul néz ki, igaz? Az a baj, hogy ilyen esetben a beírt útvonalat nem a PowerShell, hanem az elindított alkalmazás értelmezi (illetve esetünkben nem értelmezi). Az eredmény csakis egy szép hibaüzenet lehet[7]. Azt kéne megoldanunk, hogy a PowerShell (aki persze tudja az igazságot), ne a beírt karakterláncot, hanem az annak alapján megfejtett igazi útvonalat adja oda szegény, buta Notepadnak, hiszen az még soha nem is hallhatott a „`munka:`” meghajtó létezéséről. A PowerShell belső használatú útvonalainak megfejtésére, feloldására a `Resolve-Path` cmdlet szolgál, ami az alábbi módon használható:

```
[16] PS C:\> (Resolve-Path munka:\szöveg.txt).ProviderPath
C:\munka\szöveg.txt
```

Az így átalakított útvonalat már bármely külső alkalmazásnak, így a Notepadnak is odaadhatjuk.

```
[17] PS C:\> notepad (Resolve-Path munka:\szöveg.txt).ProviderPath
```

1.3.9 Változók, konstansok

Változóknak az olyan memóriaterületeket nevezzük, amelyeket szkriptünkben vagy programunkban névvel jelölünk meg, és a programnak lehetősége van az adott memóriaterületen található objektum tulajdonságait kiolvasni, megváltoztatni és a metódusait meghívni. A névvel ellátott memóriaterület tartalmazhatja közvetlenül az objektumot (érték típusú változók), illetve tartalmazhat egy hivatkozást (memóriacímet), ami az objektumok valódi helyét azonosítja (referencia típusú változók). Minden létrehozott változó számára program adatszegmensén lefoglalódik a megfelelő nagyságú memóriaterület, amelyet ezután a programból a névre való hivatkozással érhetünk el, kiolvashatjuk vagy beállíthatjuk annak értékét, illetve referencia esetén a név segítségével érhetjük el a mutatott objektumot is. Az érték- és referenciatípusok a PowerShellben gyakorlati szempontból nem különböznek egymástól lényegesen.

A típusok közötti konverzió minden esetben a .NET szabályai szerint történik, ahogyan a későbbi példákban látható. Mi tárolható tehát egy PowerShell változóban? Bármi, amit a .NET a memóriában tárolhat, méghozzá szigorúan típusos formában.

A PowerShell változónevei minden esetben a $ karakterrel kezdődnek (ha önmagában használjuk őket és nem valamilyen változókat kezelő cmdlet segítségével), betűket, számokat és speciális karaktereket is tartalmazhatnak.

[7] Nem a PowerShell, hanem a notepad jeleníti meg a hibaüzenetet.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Változókat PowerShellben legegyszerűbben a következő formában tudunk létrehozni, használni:

```
PS C:\> $a = "bcdefghijklmnopqrstuvwxyz"
PS C:\> $b = 12
PS C:\> $c = Get-Location
PS C:\> $a
bcdefghijklmnopqrstuvwxyz
PS C:\> $b
12
PS C:\> $c

Path
----
C:\
```

Látszik, hogy a változók értékadásához nem kell semmilyen kulcsszót használni, az egyenlőségjel elég az értékadáshoz. A változókhoz nem feltétlenül kell típust rendelni, azaz bármilyen típust, objektumosztályt tartalmazhatnak.

Azt, hogy egy változó éppen milyen típusú értéket tartalmaz a `GetType()` metódus meghívásával kérdezhetjük le:

```
PS C:\> $a.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object


PS C:\> $b.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType


PS C:\> $c.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    PathInfo                                 System.Object
```

Látszik, hogy a `$c` változónk nem egyszerűen egy szöveget tartalmaz, hanem egy `PathInfo` típusú objektumot.

A változók típusa nem rögzül, azaz egy `String` változónak később adhatunk `Int32` értéket minden további hiba nélkül, a változó típusa az értékének megfelelően módosul.

```
PS C:\> $a
bcdefghijklmnopqrstuvwxyz
PS C:\> $a = 22
```

Ez a megoldás persze nagyon kényelmes, de az automatikus változódeklaráció komoly problémákkal is járhat; minden elgépelésből új, rejtélyes változók születnek, érdekes, de legalábbis nehezen felderíthető futási hibákat okozva. Szkriptek esetén mindenképpen célszerű legalább a hibakeresés fázisában változtatni az alapértelmezett viselkedésen a következő módon:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> Set-PSDebug -strict
```

Ezután a PowerShell hibaüzenetet ad, ha olyan változónevet használunk, amihez korábban még nem rendeltünk értéket:

```
PS C:\> $a = 2
PS C:\> $a + $b
The variable '$b' cannot be retrieved because it has not been set.
At line:1 char:8
+ $a + $b <<<<
    + CategoryInfo          : InvalidOperation: (b:Token) [], RuntimeExceptio
   n
    + FullyQualifiedErrorId : VariableIsUndefined
```

Az alapértelmezett viselkedés a következő parancs használatával állítható vissza:

```
PS C:\> Set-PSDebug -off
```

A változók gyakran nem egyszerű elemeket tartalmaznak, hanem collection-öket (gyűjteményeket, vegyes elemű tömböket) vagy hagyományos tömböket:

```
[23] PS C:\> $alias = get-alias a*
[24] PS C:\> $alias

CommandType     Name                           Definition
-----------     ----                           ----------
Alias           ac                             Add-Content
Alias           asnp                           Add-PSSnapIn
```

A fenti példában az „a” kezdetű aliasok gyűjteménye lesz a `$alias` változó tartalma. A gyűjteményekkel és tömbökkel később foglalkozom részletesebben.

Változóknak még precízebben is tudunk értéket adni. A `set-variable` cmdlet nagyon sok opciót biztosít erre:

```
PS C:\> Set-Variable -Name PI -Value 3.14159265358979 -Option Constant
PS C:\> $PI
3.14159265358979
```

A fenti példában például a PI nevű változót konstansként hoztam létre, ami azt jelenti, hogy nem engedi felülírni más értékkel, illetve bárhonnan látható ez a változó (scope-pal, azaz láthatósággal később foglalkozom). Ezt még törölni sem lehet a `Remove-Variable` cmdlettel.

A `Set-Variable` cmdletnek van még egy érdekessége is, magyarázatot lehet adni ezzel a módszerrel a változóknak:

```
PS C:\> Set-Variable -Name Nevem -Value "Soós Tibor" -Description "Ez az én
nevem"
PS C:\> $nevem
Soós Tibor
```

De vajon hogyan olvashatjuk ki később ezt a magyarázó szöveget? Hát így nem:

```
PS C:\> $nevem.description
```

Ehhez a `get-variable` cmdletet kell használni:

```
PS C:\> (get-variable nevem).Description
Ez az én nevem
```

A `get-variable` kimeneteként nem maga a változó tartalma jön vissza, hanem egy `PSVariable` típusú objektum, aminek már kiolvasható a `Description` tulajdonsága is.

```
PS C:\> (get-variable nevem).gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    PSVariable                               System.Object
```

A korábban már mutatott `Set-Variable` cmdlethez hasonlóan a `new-variable` is használható új változók definiálásához, a két parancs között csak az a különbség, hogy a `set-variable` segítségével már meglevő változók tartalmát is meg lehet változtatni, a `new-variable` erre nem alkalmas:

```
PS C:\> $x = "körte"
PS C:\> New-Variable x -Value 55
New-Variable : A variable with name 'x' already exists.
At line:1 char:13
+ New-Variable <<<<  x -Value 55
    + CategoryInfo          : ResourceExists: (x:String) [New-Variable], Sess
   ionStateException
    + FullyQualifiedErrorId : VariableAlreadyExists,Microsoft.PowerShell.Comm
   ands.NewVariableCommand
```

Ha meguntunk egy változót, akkor tartalmát törölhetjük is a `Clear-Variable` cmdlet segítségével. Ez különösen akkor fontos, ha például egy nagyméretű gyűjtemény kerül bele a változóba, ami aztán szükségtelenné válik. Ha nem töröljük, akkor az továbbra is foglalja a memóriát. Ez magát a változót nem szünteti meg, csak az értékét törli:

```
[29] PS I:\>$x = "alma"
[30] PS I:\>Get-Variable x

Name                           Value
----                           -----
x                              alma


[31] PS I:\>Clear-Variable x
[32] PS I:\>Get-Variable x

Name                           Value
----                           -----
x


[33] PS I:\>Remove-Variable x
[34] PS I:\>Get-Variable x
Get-Variable : Cannot find a variable with name 'x'.
At line:1 char:13
```

```
+ Get-Variable <<<<  x
    + CategoryInfo          : ObjectNotFound: (x:String) [Get-Variable], Item
   NotFoundException
    + FullyQualifiedErrorId : VariableNotFound,Microsoft.PowerShell.Commands.
   GetVariableCommand
```

A [32]-es promptnál látjuk, hogy a `Clear-Variable` után az `$x` változóm még létezik, csak nincs értéke. A `Remove-Variable` után a [34]-es promptban viszont már nem létezik `$x`.

Megjegyzés

Változónévnek mindenféle csúnyaságot is használhatunk, de lehetőség szerint ne tegyünk ilyet, mert a szkriptünk, parancssorunk értelmezhetőségét nagyon megnehezíthetjük:

```
[37] PS I:\>$1 = 5
[38] PS I:\>$1
5
[39] PS I:\>$1+2
7

[40] PS I:\>${kód}="sor"
[41] PS I:\>"${kód}sor"
sorsor

[42] PS I:\>$$ = "Dollár"
[43] PS I:\>$$
Dollár

[44] PS I:\>$_a="bla"
[45] PS I:\>$_a
bla
```

A kapcsos zárójel a változó nevében különleges szerepet kap:

```
[80] PS C:\munka> $a = "ablak"
[81] PS C:\munka> ${a}
ablak
```

Látható, hogy bár a változó neve csak egy „a” betű, ennek ellenére a kapcsos zárójelben szerepeltetve is ugyanazt az eredményt kaptam. Így a kapcsos zárójelnek az a szerepe, hogy olyan változóneveket is használhassunk, amelyek olyan karaktereket is tartalmaznak, amelyek ott nem megengedettek:

```
[20] PS C:\> ${Ez egy nem szép nevű változó! Hanem csúnya. Bizony} = 1
[21] PS C:\> ${Ez egy nem szép nevű változó! Hanem csúnya. Bizony}
1
```

Ilyenkor a TAB kiegészítés úgy működik, hogy a `$Ez` után leütött TAB a teljes, kapcsoszárójeles neve írja ki.

1.3.10 Idézőjelezés, escape használat

Az idézőjelek használata arra szolgál, hogy a PowerShell számára egyébként valamiféle speciális jelentéssel bíró szövegdarabokat egyszerű karakterláncként olvastassuk be. A PowerShellben kétfajta idézőjel létezik. Az

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

egyik a macskaköröm (”), a másik az aposztróf (’). Ez utóbbi nem tévesztendő össze a „visszafele aposztrófal” (`– AltGr + 7), ami az Escape, azaz „hatástalanító” karakter.

```
PS C:\> $szöveg = "karaktersorozat"
PS C:\> $szöveg
karaktersorozat
PS C:\> "Ez itt egy $szöveg."
Ez itt egy karaktersorozat.
PS C:\> 'Ez itt egy $szöveg'
Ez itt egy $szöveg
PS C:\>
```

A fenti példában látszik, hogy a macskaköröm (”) közti szövegben a PowerShell észreveszi a változókat (a $ jel alapján) és behelyettesíti a változó értékét. Az aposztróf (’)-nál nem értékel ki semmit sem, minden karaktert szó szerint értelmez.

Hogyan kell például macskaköröm (”) között akkor $ jelet kiíratni?

```
PS C:\> "Ez most itt egy `$jel"
Ez most itt egy $jel
```

Itt jön jól az Escape karakter, ami hatástalanítja a $ alaphelyzet szerinti hatását.

Vigyázat, a $ jel hatása macskakörmök között csak az első szóelválasztó karakterig érvényes! Azaz nem működik így egy adott szöveg hossztulajdonságának kiíratása:

```
PS C:\> $szó = "ablak"
PS C:\> "Az ablak szó $szó.Length karakter hosszú."
Az ablak szó ablak.Length karakter hosszú.
```

Hogyan lehet az egész kifejezést kiértékeltetni a macskakörmök között?

```
PS C:\> "Az ablak szó $($szó.Length) karakter hosszú."
Az ablak szó 5 karakter hosszú.
```

Azaz a problematikus részt bezárójelezzük, és felhívjuk a PowerShell figyelmét, hogy a változóknál megszokott kiértékelés szerint járjon el ez egész zárójeles kifejezés esetében, ezért elé teszünk még egy $ jelet.

Többsoros szöveget is egyszerűen be tudunk írni:

```
PS C:\> $hosszú = "1. sor
>> 2. sor
>> 3. sor"
>>
PS C:\> $hosszú
1. sor
2. sor
3. sor
```

Egyszerűen Entert nyomunk a sortörésnél. A parancssor értelmező beindul és észreveszi, hogy egy idézőjel után vagyunk, így átvált egy alpromptra és várja a szöveget a következő idézőjelig.

Ha kombináljuk az idézőjeleket egész érdekes eredményre is juthatunk:

```
[8] PS C:\> $szöveg = "valami"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[9] PS C:\> "ide ágyazom be a szöveget: '$szöveg'"
ide ágyazom be a szöveget: 'valami'
```

Azaz a [9]-es sorban idézőjelek közötti aposztrófot alkalmaztam, ennek ellenére a változó kiértékelődött az aposztrófok között.

1.3.11 Sortörés, többsoros kifejezések

A PowerShell nagyon rugalmasan kezeli a többsoros kifejezéseket. Ha egy nyitó karakter (macskaköröm, aposztróf, minden fajta zárójel) után új sort kezdünk, akkor kapunk egy un. „nested prompt”-ot, azaz beágyazott promptot, melyet két > jel jelez, és ahol folytathatjuk a parancssorunk írását. A macskaköröm példáját már előzőleg mutattam, most nézzünk pár egyéb példát:

```
PS C:\> (get-service
>> ).count
>>
91
```

Szögletes zárójel:

```
PS C:\> $a = get-service
PS C:\> $a[
>> 1]
>>

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
```

Minden egyéb esetben, amikor tehát nem egyértelmű a PowerShell számára, hogy folytatódik a sor, ott segítsünk neki a már említett (`) karakterrel.

```
PS C:\> get-service -name `
>> Alerter
>>

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
```

A szövegek többsoros tárolására van még egy lehetőség, az un. „here string” alkalmazása, amit egy önálló sorban levő @” vagy @’ karakterpárral vezetünk be és egy önálló ”@ vagy ’@ karakterpárral fejezünk be. Ennél - hasonlóan a korábbiakhoz - a macskakörmös változat kiértékelődő, az aposztrófos pedig annyira „erős” idézőjelezésnek számít, hogy a köztes `, ” és ’ jelet sem veszi figyelembe:

```
PS C:\> $szöveg = @'
>> Itt aztán lehet bármi, sortörés
>> Escape karakter `
>> Aposztróf '
>> Macskaköröm "
>> '@
>>
```

```
PS C:\> $szöveg
Itt aztán lehet bármi, sortörés
Escape karakter `
Aposztróf '
Macskaköröm "
```

1.3.12 Kifejezés- és parancsfeldolgozás

A PowerShell két különböző üzemmódban képes értelmezni a beírt szöveget – kifejezés és parancs üzemmódban. Kifejezés üzemmódban a shell a legtöbb magas szintű programnyelvhez hasonlóan viselkedik: a számokat számnak tekinti, a karakterláncokat idézőjelek közé kell tennünk, stb. Kifejezések például a következők:

```
2+2
"Helló" + " világ!"
```

Parancs üzemmódban a karakterláncokhoz nincs szükség idézőjelekre, mivel a shell a változókon és a zárójelben lévő kifejezéseken kívül mindent karakterláncnak tekint. Ilyen módon értelmezi a parancsfeldolgozó az alábbi utasításokat:

```
write-host 2+2
copy-item egyik.txt masik.txt
```

Most már csak azt kellene tudnunk, hogy pontosan mi az, aminek hatására a PowerShell egyik vagy másik üzemmódot választja a parancsunk értelmezésére. Az üzemmódot az első beolvasott token (vagyis értelmezhető parancsdarab) fogja meghatározni az alábbiaknak megfelelően:

- Kifejezés üzemmódba vált a PowerShell, ha a beírt szöveg számmal (vagy egy pont karaktert követő számmal), idézőjelek közé tett karakterlánccal, vagy $ jellel kezdődik.
- Parancs üzemmódba vált a PowerShell, ha a beírt szöveg bármilyen betűvel, a `&` karakterrel, egy pont utáni szóközzel, vagy pont utáni betűvel kezdődik.

A két üzemmód vegyes használatát zárójelezéssel érhetjük el, a zárójelek közötti szövegre minden esetben újra megtörténik az üzemmód meghatározása a fenti szempontok szerint.

```
PS C:\> 2+2
4
PS C:\> Write-Host 2+2
2+2
PS C:\> Write-Host (2+2)
4
PS C:\> (Get-Date).Year + 3
2010
PS C:\> "Get-Date"
Get-Date
PS C:\> &"Get-Date"

2007. július 11. 15:37:51
```

tanfolyami felhasználása nem engedélyezett!

A fenti példákon jól látható a két üzemmód közötti különbség és az üzemmód váltás kényszerítése. Az első utasítást a PowerShell kifejezés üzemmódban értelmezte, elvégezte a műveletet és kiírta az eredményt. A második sor egy cmdlet nevével, vagyis betűvel kezdődik, ennek hatására aktiválódik a parancs üzemmód, és a shell már nem adja össze a két számot, egyszerű karakterláncként (pedig nincs idézőjelek között!) olvassa be. A harmadik sor betűvel kezdődik (parancs üzemmód), de a zárójelek miatt később újra megtörténik az üzemmód meghatározása, és a zárójelben lévő számok hatására a PowerShell az összeadás erejéig kifejezés üzemmódra vált. Az negyedik sorban a zárójeles szövegre parancs üzemmód indul, így a cmdlet kimenete kerül a teljes kifejezésbe. Az ötödik sorban egy karakterláncot adunk meg (ami éppen egy cmdlet neve), de az idézőjelek miatt a PowerShell kifejezés üzemmódra vált, így nem ismeri fel a cmdletet, a nevet egyszerű karakterláncnak tekinti. Az utolsó sorban a & karakter segítségével lefuttatjuk a karakterláncot, vagyis parancs üzemmódra kényszerítjük a shellt.

Ha meg akarjuk számolni, hogy hány parancs (cmdlet) van a PowerShellben, akkor ezt nagyon egyszerűen megtehetjük. A cmdletek listáját a `Get-Command` adja vissza egy objektumcsoport képében, a .NET-ben pedig minden objektumcsoporthoz tartozik egy `Count` tulajdonság, amely az elemek számát adja vissza. Vagyis:

```
[25] PS C:\> get-command.Count
The term 'get-command.Count' is not recognized as the name of a cmdlet, functi
on, script file, or operable program. Check the spelling of the name, or if a
path was included, verify that the path is correct and try again.
At line:1 char:18
+ get-command.Count <<<<
    + CategoryInfo          : ObjectNotFound: (get-command.Count:String) [],
   CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

A hibaüzenetből megtudhatjuk, hogy nem sikerült a „`Get-Command.Count`” utasítást végrehajtani, ahogyan az várható is volt. Természetesen nem egyszerre kellene végrehajtani az egész parancsot, hanem elsőként csak a `Get-Command`-nak kellene lefutnia, ezután pedig a kimeneten megjelenő gyűjteményre kellene `Count`-ot kérni. Ezt az alábbi szintaktika szerint érhetjük el:

```
[26] PS C:\> (get-command).Count
410
```

Láttuk a fenti példákban, hogy a PowerShell a cmdletek paramétereit alapvetően szövegként értelmezi (parancsmód), kivéve a változókat:

```
PS C:\> $bla="PowerShell"
PS C:\> Write-Output $bla
PowerShell
```

Sőt! Ilyenkor még a változó metódusaira és tulajdonságaira is lehet hivatkozni különösebb trükközés nélkül:

```
PS C:\> Write-Output $bla.Length
10
PS C:\> Write-Output $bla.split("S")
Power
hell
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.3.13 Utasítások lezárása

A PowerShell utasításainak lezárására két karakter is használható: az egyik a pontosvessző (;), a másik pedig az újsor karakter. Az újsor karakter azonban csak akkor jelenti az utasítás végét, ha az adott utasítást a PowerShell szintaktikailag teljesnek és befejezettnek tekinti, ellenkező esetben a shell megpróbálja bekérni a hiányzó befejezést.

Nézzünk egy-egy példát ezekre. Elsőként az egy sorban több kifejezés megadására:

```
PS C:\> 1+2; 54/12; "Ezek voltak az eredmények egy sorban kiszámolva"
3
4,5
Ezek voltak az eredmények egy sorban kiszámolva
```

A PowerShellnek hiányérzete van, alpromptot nyit a kifejezés korrekt lezárásához:

```
PS C:\> 2 +
>> 2
>>
4
PS C:\>
```

Ha a lezártnak is tekinthető utasításokat mégis folytatni szeretnénk, akkor ismét a korábban már megismert Escape (`) karaktert kell használnunk:

```
PS C:\> write-host `
>> Helló `
>> világ `
>>
Helló világ
```

1.3.14 Csővezeték, futószalag (Pipeline)

A hagyományos shellekhez hasonlóan a PowerShellnek is fontos eleme a csővezeték (pipeline), azzal a lényeges különbséggel, hogy a PowerShell-féle csővezetékben komplett objektumok (illetve objektumreferenciák) közlekednek, így a csővezeték minden eleme megkapja az előtte álló parancs által generált teljes adathalmazt, függetlenül attól, hogy az adott kimenet hogyan jelenne meg a képernyőn.

De mi is az a csővezeték, és hogyan működik? Talán nem is szerencsés a csővezeték elnevezés, hiszen ha valamit beöntünk egy csővezeték elején, az általában változatlan formában bukkan fel a végén, a PowerShell-féle csővezeték lényege pedig éppen a benne utazó dolgok (objektumok) átalakítása, megfelelő formára hozása. Sokkal szemléletesebb, ha csövek helyett egy futószalagot képzelünk el, amelyen objektumok utaznak, a szalag mellett álló munkásaink (a cmdletek) pedig szorgalmasan elvégzik rajtuk a megfelelő átalakításokat, mindegyikük azt, amihez éppen ő ért a legjobban. Egyikük lefaragja és eldobja a fölösleget, a következő kiválogatja a hibás darabokat, egy másik kiválogatja, és szépen becsomagolja az egybetartozókat, stb. A szalag végén pedig mi megkapjuk a késztermékeket, ha ügyesek voltunk (vagyis a megfelelő munkásokat állítottuk a szalag mellé, és pontosan megmondtuk nekik, hogy mit kell csinálniuk), akkor éppen azt, és olyan formában, amire és ahogyan szükségünk volt.

A csövezhetőség azzal is jár, hogy az egyik cmdlet kimenete a következő cmdlet bemeneteként megy tovább, anélkül hogy nekünk az első kimenetet el kellene mentenünk változóba. Ezt a paraméterátadást a

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

PowerShell helyettünk elvégzi. Ez - annak figyelembevételével - különösen praktikus, hogy a különböző cmdletek kimeneteként többfajta, előre nem biztos, hogy pontosan meghatározható típusú kimenet, általánosan gyűjtemény, azaz *collection* lehet.

Ráadásul, ha ilyen gyűjtemény a kimenet, akkor lehet, hogy annak első tagja hamar előáll. Ha ügyesen van megírva a következő csőszakasz helyén álló cmdlet, akkor az már a rendelkezésre álló első gyűjteményelemet el is kezdheti feldolgozni, sőt, akár tovább is adhatja az ő utána következő csőszakasznak. Ez jóval hatékonyabb feldolgozást tesz lehetővé memória-felhasználás tekintetében, hiszen nem kell bevárni az utolsó elem megérkezését és eltárolni az addigi elemeket.

Nézzünk egy nagyon egyszerű példát, a már többször alkalmazott `get-member` cmdlet példáján. Látható, hogy a csőszakaszok csatlakozását a | jel (Alt Gr + W) jelenti:

```
[4] PS C:\> "sztring" | get-member


   TypeName: System.String

Name             MemberType            Definition
----             ----------            ----------
Clone            Method                System.Object Clone()
CompareTo        Method                System.Int32 CompareTo(Object val...
Contains         Method                System.Boolean Contains(String va...
CopyTo           Method                System.Void CopyTo(Int32 sourceIn...
EndsWith         Method                System.Boolean EndsWith(String va...
Equals           Method                System.Boolean Equals(Object obj)...
...
```

A beírt „sztring” kimenete maga a „sztring”, ezt küldjük tovább a `get-member` cmdletnek, amely kilistázza a sztring objektum tagjait.

De vajon honnan tudja a PowerShell, hogy a `get-member` számos paramétere közül melyik legyen a csőből kieső kimenet?

Ehhez az adott cmdlet helpje (itt most kicsit megkurtítva) ad segítséget:

```
[5] PS C:\> get-help get-member -full

NAME
    Get-Member

SYNOPSIS
    Gets information about objects or collections of objects.

SYNTAX
    Get-Member [[-name] <string[]>] [-inputObject <psobject>] [-memberType
    {<AliasProperty> | <CodeProperty> | <Property> | <NoteProperty> | <Scri
    ptProperty> | <Properties> | <PropertySet> | <Method> | <CodeMethod> |
    <ScriptMethod> | <Methods> | <ParameterizedProperty> | <MemberSet> | <A
    ll>}] [-static] [<CommonParameters>]


DETAILED DESCRIPTION
    Gets information about the members of objects. Get-Member can accept in
...

PARAMETERS
    -name <string[]>
```

```
        Specifies the member names to retrieve information about.

        Required?                   false
        Position?                   1
        Default value               *
        Accept pipeline input?      false
        Accept wildcard characters?  true

    -inputObject <psobject>
        Specifies the objects to retrieve information about. Using this par
        ameter to provide input to Get-Member results in different output t
        han pipelining the same input. When you pipeline input to Get-Membe
        r, if the input is a container, the cmdlet returns information abou
        t each unique type of element in the container. If you provide the
        same input by using the InputObject parameter, the cmdlet returns i
        nformation about the container object itself. If you want to use pi
        pelining to retrieve information about a container, you must procee
        d the pipelined input by a comma (,). For example, if you informati
        on about processes stored in a variable named $process, you would t
        ype ,$process | get-member to retrieve information about the contai
        ner.

        Required?                   false
        Position?                   named
        Default value
        Accept pipeline input?      true (ByValue)
        Accept wildcard characters?  false
...
```

Tehát a cmdlet azon paramétere, amely fogadja a csővezetéken érkező adatot az általában „`inputObject`” névre hallgat, illetve a súgóban fel van tüntetve, hogy „`Accept pipeline input? true`”.

Látható tehát, hogy a `get-member` képes csőből adatokat feldolgozni, de vajon mi van akkor, ha egy cmdlet erre nem képes, vagy nincs is a mi céljainknak megfelelő cmdlet? Azaz a futószalag mellé nem tudunk beállítani egy kész, komplett gépet, hanem a gépet nekünk kell összerakni. Szerencsére erre is van lehetőség, számos ilyen „csőkezelő”, vagy „futószalag-melletti-gép-összerakó” cmdlet van, amelyekkel ezt meg lehet oldani. Most itt egyelőre csak kettőt vegyünk előre: a `ForEach-Object` és a `Where-Object` cmdleteket.

A `ForEach-Object` a futószalag melletti üres gépház, a belsejét kapcsoszárójel-pár közé kell beírni. A következő példában a futószalagon érkező elemeket duplázza az általam összerakott gép:

```
[63] PS C:\> 1,8,3 | ForEach-Object {$_ * 2}
2
16
6
```

Mi van a gép belsejében? Egy olyan utasítássor, hogy: „Vedd az aktuális futószalagon érkező elemet, és szorozd meg kettővel!”. Az „aktuális futószalagon érkező elemet” a `$_` furcsa nevű változó szimbolizálja. Ezzel a `ForEach-Object` belsejébe bekerül minden elem, azokon az előírt művelet végrehajtódik, majd az így módosított elem megy tovább a szalagon, jelen esetben itt vége is a szalagnak és az eredmény kihullott a képernyőre.

A másik fontos alapgép a futószalag mellett a szelektáló gép, ez csak azokat az elemeket engedi tovább, amelyek valamilyen általunk felállított kritériumot teljesítik. Ez a gép tehát magukat az elemeket nem

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

módosítja, csak kicsit „megegyeli” azokat, mint a konzervgyárban az alulméretes almákat. Az ezt megvalósító cmdlet neve `Where-Object`. Nézzünk erre is példát:

```
[64] PS C:\> 15,2,11,5 | Where-Object {$_ -gt 10}
15
11
```

Itt azokat az elemeket engedjük csak tovább, amelyek teljesítik azt a megadott feltételt, hogy nagyobbak, mint 10.

Az *1.7.8 Pipe kezelése, filter* c. fejezetben még részletesebben ismertetem a csövezést, hiszen ott majd mi magunk is írunk ilyen „csőképes” függvényt.

1.3.15 Kimenet (Output)

Az előző alfejezetben bemutattam a csövezés lehetőségét. Ez nem egyszerűen csak paraméterátadás, hanem el is kaphatjuk az éppen átadott paramétert, és annak tulajdonságaival, metódusaival is játszhatunk. Az így átadott paramétert a `$_` speciális, automatikusan generálódó változón keresztül érhetjük el.

Fontos fogalom még a PowerShellben az *output*, azaz a kimenet fogalma. Majdnem minden cmdletnek van kimenete. Ha mégsem lenne, akkor is van, merthogy ha nem készítünk explicit kimenetet, akkor az implicit módon alaphelyzetben az outputra adódik át. Ez az output lesz a következő csőszakaszban elérhető `$_` változó tartalma.

Nézzünk erre példát:

```
[1] PS C:\> "szöveg" | ForEach-Object{$_.Length}
6
[2] PS C:\> write-output "szöveg" | ForEach-Object{$_.Length}
6
[3] PS C:\> write-host "szöveg" | ForEach-Object{$_.Length}
szöveg
```

Az első és a második promptnál ugyanazt az eredményt kapjuk, azaz az első csőszakaszban megszületik a „szöveg” objektum, amit átadunk a következő csőszakasznak. Az első esetben implicit módon adjuk át, a második esetben pedig explicit módon.

A `write-output` cmdletet igazából nem „emberi fogyasztásra” szánják, ez tisztán a csőszakaszok közti paraméterátadásra szánták, nem pedig a csicsás, színes-szagos konzolos adatmegjelenítésre. Más kérdés, hogy ha a csővezeték legvégén nem rendelkezünk a csővezeték tartalmáról, akkor az „kifolyik” a konzol ablakba, azaz kiíródik a képernyőre.

Nézzük meg a `write-output` helpjéből a szintaxist:

```
[7] PS C:\> (get-help write-output).syntax

Write-Output [-inputObject] <PSObject[]> [<CommonParameters>]
```

Nincs túl sok extra lehetőségünk.

A fenti [3]-as promptnál viszont nem az outputra küldöm a „`szöveg`”-et, hanem a `write-host` cmdlettel a konzolra (képernyőre), így a következő csőszakasz nem kap semmit és ezért nem is jelenik meg semmilyen hosszadat, csak a `write-host` által kiírt szöveg.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nézzük meg a `write-host` szintaxisát is:

```
[8] PS C:\> (get-help write-host).syntax

Write-Host [[-Object] <Object>] [-BackgroundColor {Black | DarkBlue | DarkGree
n | DarkCyan | DarkRed | DarkMagenta | DarkYellow | Gray | DarkGray | Blue | G
reen | Cyan | Red | Magenta | Yellow | White}] [-ForegroundColor {Black | Dark
Blue | DarkGreen | DarkCyan | DarkRed | DarkMagenta | DarkYellow | Gray | Dark
Gray | Blue | Green | Cyan | Red | Magenta | Yellow | White}] [-NoNewline] [-S
eparator <Object>] [<CommonParameters>]
```

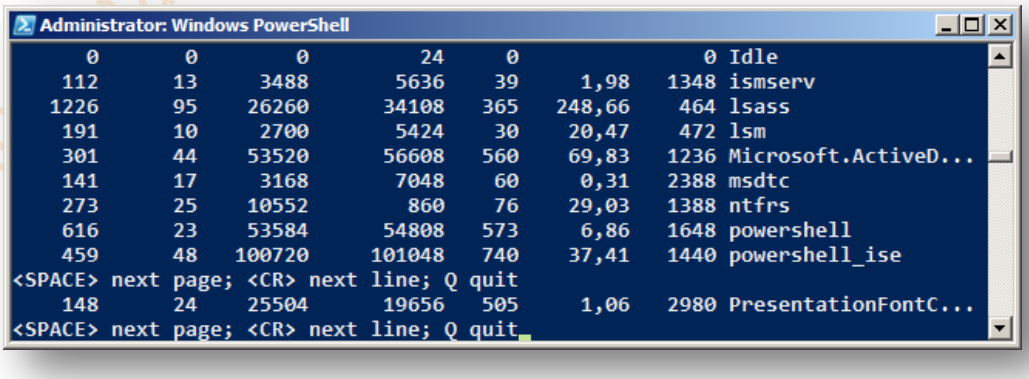
Ez már sok olyan lehetőséget is tartalmaz, amelyek tényleg az kiírt adatok élvezeti értékét fokozzák: háttér- és betűszínt lehet beállítani, valamint azt, hogy ne nyisson új sort a kiírás végén, valamint ha tömböt adunk neki kiíratásra, akkor megadhatjuk az elválasztó karaktert, aminek közbeiktatásával írja ki egymás mellé az elemeket soremelés helyett.

Amúgy számos egyéb helyekre is tudjuk irányítani a kimenetet:

```
[29] PS C:\> get-command write* -CommandType cmdlet

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Write-Debug                         Write-Debug [-Message] <Str...
Cmdlet          Write-Error                         Write-Error [-Message] <Str...
Cmdlet          Write-EventLog                      Write-EventLog [-LogName] <...
Cmdlet          Write-Host                          Write-Host [[-Object] <Obje...
Cmdlet          Write-Output                        Write-Output [-InputObject]...
Cmdlet          Write-Progress                      Write-Progress [-Activity] ...
Cmdlet          Write-Verbose                       Write-Verbose [-Message] <S...
Cmdlet          Write-Warning                       Write-Warning [-Message] <S...
```

A fenti egyéb `write-…` cmdleteknek a szkriptek hibakeresésénél van elsődlegesen jelentőségük, így majd a hibakeresés fejezetben fogok ezzel részletesen foglalkozni.

Még egy cmdlet kívánkozik ide, az `Out-Host`. Ez a cmdlet a kimenetét kiírja a konzolra. Tulajdonképpen elhagyható, hiszen a kimenet alaphelyzetben is a konzolon jelenik meg, de ennek a cmdletnek van egy `-paging` kapcsoló paramétere. Ennek alkalmazásával lapozni tudjuk a kimenetet, ami főleg sok adat megjelenítésekor lehet praktikus:



22. ábra Az Out-Host -Paging hatására megjelenő lapozás

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Látható, hogy a szóköz billentyű lenyomásával komplett képernyőnyi adatot tudunk lapozni, az Enter leütésével egy sort.

1.3.16 Jóváhagyás kezelése, óvatos végrehajtás

A PowerShell parancsai között lehetnek „veszélyesek” is. Például egy fájl törlése, ha azt véletlenül, egy hibás kifejezés miatt követjük el, akkor annak nem biztos, hogy örülünk. Ezért az egyes cmdleteknél gyárilag meg van határozva egy veszélyességi szint, ami lehet `Low`, `Medium`, `High`.

Nézzük is meg, hogy mi van beállítva! Ez egy elég bonyolult kifejezés, így kérem a tisztelt olvasót ne dobja itt el a könyvet! Ha majd a *2.4 Fejlett függvények – script cmdletek* fejezethez érünk minden világos lesz.

```
Get-Command -CommandType cmdlet |
foreach-object {New-Object System.Management.Automation.CommandMetaData ($_)} |
          select-object @{l="name"; e={$_.name}},
                @{l="confirmimpact"; e={$_.confirmimpact}} |
                      Sort-Object confirmimpact -Descending | ft -AutoSize
```

Ennek eredménye:

```
name                                confirmimpact
----                                -------------
Enable-PSSessionConfiguration                High
Enable-PSRemoting                            High
Set-PSSessionConfiguration                   High
Unregister-PSSessionConfiguration            High
Register-PSSessionConfiguration              High
Disable-PSSessionConfiguration               High
Remove-PSBreakpoint                        Medium
Remove-Module                              Medium
Remove-PSDrive                             Medium
…
```

Azaz a cmdletek zöme `Medium` veszélyességűnek van besorolva és csak néhány `High`. Ehhez hozzá kell venni a `$ConfirmPreference` automatikus változót:

```
[1] PS C:\> $ConfirmPreference
High
```

Aminek az értéke „`High`”. Ez azt jelenti, hogy alaphelyzetben a PowerShell csak a fenti lista első 6 elemében látható cmdlet futtatására kérdez rá, a többit szó nélkül lefuttatja. Nézzünk egy rákérdezésest:

```
[2] PS C:\> Enable-PSRemoting

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable this machine for remote
management through WinRM service.
 This includes:
    1. Starting or restarting (if already started) the WinRM service
    2. Setting the WinRM service type to auto start
    3. Creating a listener to accept requests on any IP address
    4. Enabling firewall exception for WS-Management traffic (for http only).

Do you want to continue?
```

```
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
WinRM already is set up to receive requests on this machine.
WinRM already is set up for remote management on this machine.
```

Hat különböző lehetőségünk van:

- `Y – Yes`: hadd fusson az adott elemre.
- `A – Yes to All`: minden elemre fusson, azaz ha egy csővezeték-feldolgozó cmdlettel van dolgunk, akkor az összes csőelemre Igen a válaszunk, amúgy elemenként rákérdezne.
- `N – No`: Ne fusson az adott elemre.
- `L – No to All`: egyik elemre se fusson, gyakorlatilag leállítjuk a futtatást
- `S – Suspend`: felfüggesztjük a futtatást, és egy alpromptot nyitunk, ahol mindenfélét ellenőrizhetünk, kiolvashatunk, és ha ezzel megvagyunk, akkor dönthetünk újra, hogy mi legyen.
- `? – Help`: felsorolja és bekéri a kötelezően kitöltendő paramétereket.

Ezek közül talán az „S” az érdekes, nézzük ezt:

```
[3] PS C:\> Enable-PSRemoting

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable this machine for remote
management through WinRM service.
 This includes:
    1. Starting or restarting (if already started) the WinRM service
    2. Setting the WinRM service type to auto start
    3. Creating a listener to accept requests on any IP address
    4. Enabling firewall exception for WS-Management traffic (for http only).

Do you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):s
[4] PS C:\>>> get-date

2010. február 10. 15:44:56


[5] PS C:\>>> exit

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable this machine for remote
management through WinRM service.
 This includes:
    1. Starting or restarting (if already started) the WinRM service
    2. Setting the WinRM service type to auto start
    3. Creating a listener to accept requests on any IP address
    4. Enabling firewall exception for WS-Management traffic (for http only).

Do you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):
```

Látható, hogy a [4]-es promptnál áttértem egy alpromptra. Itt tudok bármit futtatni anélkül, hogy az eredeti parancsomat megszakítottam volna. Ha kész vagyok, akkor ebből az alpromptból az `Exit` kulcsszóval léphetek ki és ott folytathatom a válaszadást, ahol az előbb abbahagytam.

Nézzünk egy „`Medium`” cmdletet:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[16] PS C:\> $a = 1
[17] PS C:\> Remove-Variable a
```

Semmit nem kért. Ha azt szeretnénk, hogy erre is rákérdezzen, akkor két lehetőségünk van. Az egyik egy „helyi” megoldás, ami csak erre az egy parancsra fog hatni:

```
[18] PS C:\> $a = 1
[19] PS C:\> Remove-Variable a -Confirm

Confirm
Are you sure you want to perform this action?
Performing operation "Remove Variable" on Target "Name: a".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
```

A másik lehetőség „globális”, azaz az egész PowerShell ügyködésünkre hatással lesz:

```
[21] PS C:\> $ConfirmPreference = "medium"
[22] PS C:\> $a = 1
[23] PS C:\> Remove-Variable a

Confirm
Are you sure you want to perform this action?
Performing operation "Remove Variable" on Target "Name: a".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
```

Ezután minden `Medium` és `High` veszélyességű cmdletre megerősítést fog kérni. Ez a beállítás azonban csak addig él, amíg be nem csukjuk a PowerShell alkalmazást. Újra megnyitva már újból visszaáll a „`High`” alapbeállítás.

1.3.17 Egyszerű formázás

Az output fogalmához tartozik hozzá, hogy három alapvető formázási módról beszéljek. Említettem, hogy a PowerShell objektumokat kezel, amelyeknek különböző kiolvasható tulajdonságaik (property) vannak. Egy-egy objektum típusnak nagyon sok ilyen tulajdonsága is lehet, így probléma, hogy akkor egy ilyen objektumot hogyan is jelenítsünk meg a képernyőn, hiszen ez az elsődleges interakciós felületünk a PowerShellel.

Nézzünk a problémára egy példát, listázzuk ki a gépen futó W-vel kezdődő szolgáltatásokat:

```
[47] PS C:\> get-service w*

Status   Name               DisplayName
------   ----               -----------
Running  W32Time            Windows Time
Running  WebClient          WebClient
Stopped  WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se...
Running  winmgmt            Windows Management Instrumentation
Stopped  WmdmPmSN           Portable Media Serial Number Service
Stopped  Wmi                Windows Management Instrumentation ...
Stopped  WmiApSrv           WMI Performance Adapter
Running  wscsvc             Security Center
Running  WSearch            Windows Search
Running  wuauserv           Automatic Updates
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Running  WZCSVC             Wireless Configuration
```

Látszik, hogy alaphelyzetben táblázatos nézetben látjuk a szolgáltatások három tulajdonságát: `Status`, `Name`, `DisplayName`. A „`DisplayName`” oszlopban nem minden szöveg fér ki, ezért van egy másik listázási lehetőségünk is:

```
[48] PS C:\> get-service w* | format-list


Name                  : W32Time
DisplayName           : Windows Time
Status                : Running
DependentServices     : {}
ServicesDependedOn    : {}
CanPauseAndContinue   : False
CanShutdown           : True
CanStop               : True
ServiceType           : Win32ShareProcess

Name                  : WebClient
DisplayName           : WebClient
Status                : Running
DependentServices     : {}
ServicesDependedOn    : {MRxDAV}
CanPauseAndContinue   : False
CanShutdown           : True
CanStop               : True
ServiceType           : Win32ShareProcess

Name                  : WinHttpAutoProxySvc
DisplayName           : WinHTTP Web Proxy Auto-Discovery Service
Status                : Stopped
DependentServices     : {}
ServicesDependedOn    : {Dhcp}
CanPauseAndContinue   : False
CanShutdown           : False
CanStop               : False
ServiceType           : Win32ShareProcess
…
```

Itt már jól kiférnek a hosszabb feliratok is, ezt a formázást hívjuk lista nézetnek, és az objektumok ilyen jellegű megjelenítését a `format-list` cmdlet végzi. Ezt olyan gyakran használjuk, hogy ennek rövid alias nevét érdemes mindenképpen megjegyezni: `fl`.

Van, hogy pont az ellenkezőjére van szükség, a listanézet helyett szeretnénk táblázatos nézetet. Ennek cmdletje a `format-table`, röviden az `ft`.

Mindkét esetben kapunk egy alaphelyzet szerinti tulajdonságlistát. Azaz nem szabad megijedni, ha pont azokat az információkat nem látjuk, amire szükségünk lenne. Először érdemes minden objektum esetében a `get-member`-rel ellenőrizni a tulajdonságok listáját és utána ezekre hivatkozhatunk a `format-` parancsoknál:

```
[11] PS C:\> Get-Service w* | Get-Member -MemberType properties


   TypeName: System.ServiceProcess.ServiceController
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name                MemberType    Definition
----                ----------    ----------
Name                AliasProperty Name = ServiceName
RequiredServices    AliasProperty RequiredServices = ServicesDependedOn
CanPauseAndContinue Property      System.Boolean CanPauseAndContinue {...
CanShutdown         Property      System.Boolean CanShutdown {get;}
CanStop             Property      System.Boolean CanStop {get;}
Container           Property      System.ComponentModel.IContainer Con...
DependentServices   Property      System.ServiceProcess.ServiceControl...
DisplayName         Property      System.String DisplayName {get;set;}
MachineName         Property      System.String MachineName {get;set;}
ServiceHandle       Property      System.Runtime.InteropServices.SafeH...
ServiceName         Property      System.String ServiceName {get;set;}
ServicesDependedOn  Property      System.ServiceProcess.ServiceControl...
ServiceType         Property      System.ServiceProcess.ServiceType Se...
Site                Property      System.ComponentModel.ISite Site {ge...
Status              Property      System.ServiceProcess.ServiceControl...

[12] PS C:\> Get-Service w* | Format-Table Name, Status, CanStop

Name                                    Status                CanStop
----                                    ------                -------
W32Time                                Running                   True
WcsPlugInService                       Stopped                  False
WdiServiceHost                         Stopped                  False
WdiSystemHost                          Stopped                  False
Wecsvc                                 Stopped                  False
wercplsupport                          Stopped                  False
WerSvc                                 Stopped                  False
WinHttpAutoProxySvc                    Stopped                  False
Winmgmt                                Running                   True
WinRM                                  Running                   True
wmiApSrv                               Stopped                  False
WPDBusEnum                             Stopped                  False
wuauserv                               Running                   True
wudfsvc                                Stopped                  False
```

Az [11]-es promptnál lekérdezem a szolgáltatások tulajdonságait. Látszik, hogy okos a `Get-Member`, hiszen több szolgáltatás-objektumot kap a bemeneteként, de mivel mindegyik egyforma típusú, teljesen egyforma a tulajdonságaik vannak, így csak egyszer adja meg a tulajdonságlistát.

Ezután a [12]-es promptban kiválasztottam, hogy a `Name, Status, CanStop` tulajdonságok kellenek nekem, ezeket átadom paraméterként a `Format-Table` cmdletnek és megkapom a kívánt táblázatot.

Ezt még szebbé lehet tenni az `-AutoSize` kapcsolóval, amellyel csak a szükséges minimális táblázatszélességet használja, és nem húzza szét az egészet a képernyőn:

```
[13] PS C:\> Get-Service w* | Format-Table Name, Status, CanStop -AutoSize

Name                 Status CanStop
----                 ------ -------
W32Time             Running    True
WebClient           Running    True
WinHttpAutoProxySvc Stopped   False
winmgmt             Running    True
WmdmPmSN            Stopped   False
Wmi                 Stopped   False
WmiApSrv            Stopped   False
wscsvc              Running    True
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
WSearch             Running    True
wuauserv            Running    True
WZCSVC              Running    True
```

A `Format-Table` olyan gyakran használatos, hogy nézzük meg néhány ügyes további lehetőségét. Például nézzük az „a” és „b” betűvel kezdődő nevű szolgáltatásokat:

```
[14] PS C:\> Get-Service [a-b]*

Status   Name               DisplayName
------   ----               -----------
Running  AeLookupSvc        Application Experience Lookup Service
Stopped  Alerter            Alerter
Running  ALG                Application Layer Gateway Service
Stopped  AppMgmt            Application Management
Stopped  aspnet_state       ASP.NET State Service
Running  AudioSrv           Windows Audio
Stopped  BITS               Background Intelligent Transfer Ser...
Stopped  Browser            Computer Browser
```

Csoportosítsuk ezeket `Status` szerint:

```
[15] PS C:\> Get-Service [a-b]* | ft -GroupBy Status


   Status: Running

Status   Name               DisplayName
------   ----               -----------
Running  AeLookupSvc        Application Experience Lookup Service


   Status: Stopped

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter


   Status: Running

Status   Name               DisplayName
------   ----               -----------
Running  ALG                Application Layer Gateway Service
...
```

Hoppá! Ez nem valami jó. Itt tetten érhetjük a csövezést. Ugye a `format-table` szépen kapja egymás után a szolgáltatásokat és nyit egy aktuális csoportot a `Status` alapján. Ha a következő szolgáltatás is ugyanilyen státusú, akkor szépen mögé biggyeszti, de ha nem, akkor nyit egy új csoportot. Azaz kicsit összevissza az eredményünk.

A megoldás az lenne, hogy ideiglenesen összegyűjtenénk a csővezetéken átmenő összes objektumot egy pufferben, és utána berendezzük az elemeket `Status` alapján, majd utána jelenítjük meg a csoportosított nézetet. Ezt a pufferelést és sorbarendezést végzi a `Sort-Object` cmdlet:

```
[16] PS C:\> Get-Service [a-b]* | sort-object status, name |ft -GroupBy Status
```

```
   Status: Stopped

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
Stopped  AppMgmt            Application Management
Stopped  aspnet_state       ASP.NET State Service
Stopped  BITS               Background Intelligent Transfer Ser...
Stopped  Browser            Computer Browser


   Status: Running

Status   Name               DisplayName
------   ----               -----------
Running  AeLookupSvc        Application Experience Lookup Service
Running  ALG                Application Layer Gateway Service
Running  AudioSrv           Windows Audio
```

Ha pedig a hosszú nevű szolgáltatások neveit is szeretnénk látni teljesen, akkor a `-wrap` kapcsolót érdemes használni:

```
[17] PS C:\> Get-Service [a-b]* | sort-object status, name |ft -GroupBy Status
-wrap


   Status: Stopped

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
Stopped  AppMgmt            Application Management
Stopped  aspnet_state       ASP.NET State Service
Stopped  BITS               Background Intelligent Transfer Servic
                            e
Stopped  Browser            Computer Browser


   Status: Running

Status   Name               DisplayName
------   ----               -----------
Running  AeLookupSvc        Application Experience Lookup Service
Running  ALG                Application Layer Gateway Service
Running  AudioSrv           Windows Audio
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés:

Kicsit ízlelgessük ezt a kifejezésrészt: „`Format-Table Name, Status, CanStop`”. Most ez itt mi? Miért van itt vessző? Nem elég a szóköz? Majd később nézzük részletesen a kifejezések argumentumait, elöljáróban annyit, hogy a `Format-Table` itt egy háromelemű tömböt kap argumentumként, ezért kell vessző.
Melyik argumentuma kaphat tömböt?
A súgóból ez látszik:

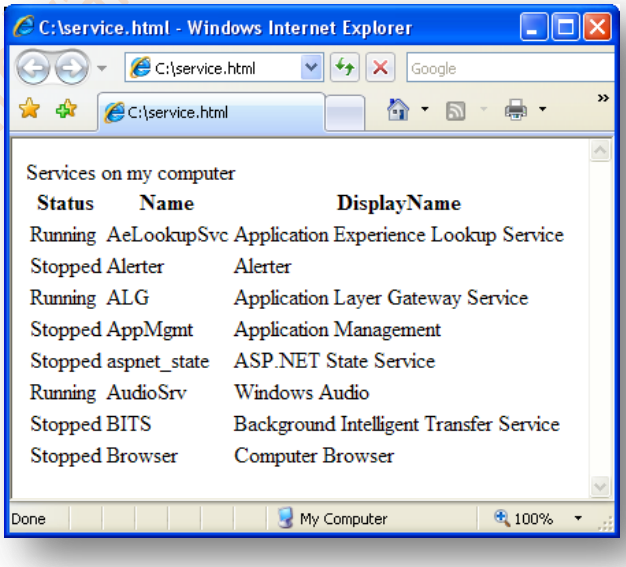
```
PS C:\> Get-Help Format-Table -Parameter property

-Property <Object[]>
    Specifies the object properties that appear in the display and the order i
    n which they appear. Type one or more property names (separated by commas)
    , or use a hash table to display a calculated property. Wildcards are perm
    itted.
…
```

Az `<Object[]>` jelzés végén a szögletes zárójelpár jelöli a tömböt. Azaz a PowerShellben nagyon fontos, hogy a több paramétert nem a vessző jelzi, hanem a szóköz! Viszont a tömbök esetében az elemeket vesszővel kell elválasztani.

1.3.18 HTML output

Ha már ilyen szép táblázatokat tudunk készíteni, akkor jó lenne ezeket valamilyen még szebb formában megjeleníteni. Ezt szolgálja a PowerShell beépített HTML támogatása:

```
[7] PS C:\> Get-Service [a-b]* | ConvertTo-Html -Property
Status,Name,DisplayName -head "Services on my computer" > service.html
```



23. ábra Convert-HTML eredménye böngészőben

A könyv tanfolyami felhasználása nem engedélyezett!

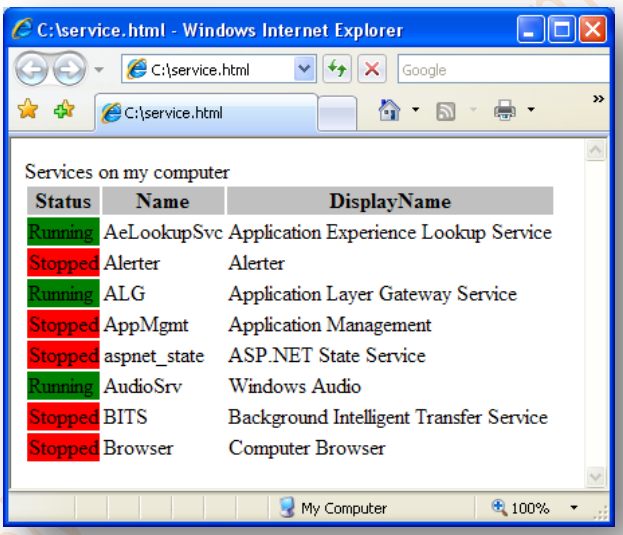
© Soós Tibor – soost@ IQJB.hu

Természetesen itt a kimenetet nem a konzolon érdemes nézni, mert ott a html forráskódot látjuk, hanem érdemes átirányítani egy fájlba, amit a böngészőben lehet megnézni.

Látjuk, hogy szép táblázatos formátumot ad, kis ügyességgel lehet ezt még fokozni. Itt most még nem célom a teljes kód elmagyarázása, csak egy kis kedvcsinálóként írom ide, a PowerGUI Script editorából kimásolva:

```
Get-Service [a-b]* | ConvertTo-Html -Property Status,Name,DisplayName `
-head "Services on my computer" |
foreach {
    if ($_ -like "*<td>Running</td>*")
          {$_ -replace "<td>Running", "<td bgcolor=green>Running"}
    elseif ($_ -like "*<td>Stopped</td>*")
          {$_ -replace "<td>Stopped", "<td bgcolor=red>Stopped"}
    else
          {$_ -replace "<tr>", "<tr bgcolor=#C0C0C0>"}
    } > c:\service.html
```

És a végeredmény:



24. ábra Kicsit kibővített HTML kimenet a böngészőben

(Ugyan ez fekete-fehér nyomtatásban nem látszik, de a futó szolgáltatások Status-a zöld, a leálltaké pedig piros hátterű.)

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.4 Típusok

A PowerShellben a típusoknak (vagy ha valaki inkább a programozós terminológiát szereti, akkor osztályoknak) nagyon nagy jelentősége van, hiszen a parancsok kimenete általában nem egyszerű sztring, hanem valamilyen egyéb objektum. Ennek ellenére az a cél, hogy a típuskezelés minél egyszerűbb legyen, azaz hogy a PowerShell alkalmas legyen hatékony parancssori felhasználásra.

Ebben a fejezetben a PowerShell által kezelt legfontosabb típusokat és azok kezelését mutatom be.

1.4.1 Típusok, típuskezelés

Láthattuk, hogy ha nem adunk meg konkrét típust, akkor a PowerShell a változónak adott érték alapján önállóan határozza meg, hogy milyen típusú változót is hoztunk létre. A parancssorba beirkált néhány utasítás esetén ez teljesen rendben is van, a (látszólagos) típustalanság rendszerint nem okoz különösebb problémát. Más a helyzet azonban hosszabb, bonyolultabb szkriptek esetén. Ekkor a kódolást, de még inkább a hibakeresést nagymértékben megkönnyíti, ha biztosak lehetünk benne, hogy milyen típusú változókat is használunk. A változó típusának meghatározása a következőképpen történik:

```
[1] PS C:\> [int] $szám = 1
[2] PS C:\> $szám = "blabla"
Cannot convert value "blabla" to type "System.Int32". Error: "Input strin
g was not in a correct format."
At line:1 char:6
+ $szám <<<<  = "blabla"
    + CategoryInfo          : MetadataError: (:) [], ArgumentTransformat
   ionMetadataException
    + FullyQualifiedErrorId : RuntimeException
```

A `$szam` változót `Int` típusként hozzuk létre, ezután csak számot (vagy számként értelmezhető karakterláncot) adhatunk neki értékként, más típusú változók, vagy objektumreferenciák már nem kerülhetnek bele. Az alábbi értékadás „típustalan” esetben egy karakterláncot eredményezne, de így a karakterlánc szám megfelelője lesz a változó értéke.

```
[3] PS C:\> [int] $szám = "123"
[4] PS C:\> $szám
123
```

A PowerShell változóinak típusaként tehát a .NET valamennyi érték-, illetve referenciatípusát megadhatjuk.

Most tegyük egy változóba a `Get-ChildItem` cmdlet kimenetét:

```
PS C:\_munka> $lista = Get-ChildItem
```

Tudjuk, hogy a `Get-ChildItem` kimenete objektumokból áll, de vajon a változóba ez milyen formában kerül? Lehet, hogy csak a képernyőre kiírt lista van benne egyszerű szövegként? Hogy biztosak lehessünk a dologban, hívjuk segítségül a rendszergazda legjobb barátját, a `get-member` cmdletet:

```
PS C:\_munka> $lista | Get-Member
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
   TypeName: System.IO.DirectoryInfo

Name                        MemberType      Definition
----                        ----------      ----------
Mode                        CodeProperty    System.String Mode{get=Mode;}
Create                      Method          System.Void Create(System.Security...
CreateObjRef                Method          System.Runtime.Remoting.ObjRef Cre...
CreateSubdirectory          Method          System.IO.DirectoryInfo CreateSubd...
…

   TypeName: System.IO.FileInfo

Name                        MemberType      Definition
----                        ----------      ----------
Mode                        CodeProperty    System.String Mode{get=Mode;}
AppendText                  Method          System.IO.StreamWriter AppendText()
CopyTo                      Method          System.IO.FileInfo CopyTo(string d...
Create                      Method          System.IO.FileStream Create()
CreateObjRef                Method          System.Runtime.Remoting.ObjRef Cre...
CreateText                  Method          System.IO.StreamWriter CreateText()
Decrypt                     Method          System.Void Decrypt()
…
```

Láthatjuk, hogy a változó teljesen az eredeti formában, `System.IO.DirectoryInfo` és `System.IO.FileInfo` objektumok alakjában tartalmazza a cmdlet kimenetét, még az sem okozott problémát, hogy a gyűjtemény két különböző típust is tartalmazott.

Interaktív üzemmódban a `Get-Member` tökéletesen alkalmas a típusok felderítésére, de a típus meghatározására nem csak itt, hanem szkriptek kódjában is szükség lehet. Ebben az esetben több megoldás közül is választhatunk, talán a legegyszerűbb az `-is` és `-isnot` operátorok használata. Kérdezzük meg, hogy milyen objektumok alkotják a `Get-Process` kimenetét:

```
PS C:\> $a = get-process
PS C:\> $a -is "System.Diagnostics.Process"
False
```

Nem `Process` objektumok?! Persze azok, csak nem jól kérdeztünk. A `Get-Process` kimenete ugyanis egy gyűjtemény, a `Process` objektumok pedig ennek belsejében vannak. A `Get-Member` ezek szerint, bár teljesen érthető okok miatt, de mégis hamis eredményt ad. Hogyan tudhatjuk meg az igazi típust? A `GetType()` metódus az `Object` osztályból öröklődik, vagyis minden elképzelhető objektum és változó esetén meghívható:

```
PS C:\> $a = get-process
PS C:\> $a.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

A kimenet tehát egyszerűen egy tömb, amelynek elemei `Object` típusú objektumok, vagyis bármi beletehető. Ha beleindexelünk a tömbbe, akkor kivehetjük belőle az egyik `Process` objektumot, hogy annak típusát is lekérdezhessük (a `Get-Member` ezt előzékenyen megteszi helyettünk):

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> $a = get-process
PS C:\> $a[0] -is "System.Diagnostics.Process"
True
```

Mit kell tennünk akkor, ha korlátozni szeretnénk ugyan a változóba kerülő típusok körét, de olyan módon, hogy mégis több különböző (bár hasonló) típus is beleférjen? A megoldást az osztályok közötti öröklődés környékén kell keresnünk. Minden objektum ugyanis a saját konkrét típusán kívül valamennyi ősosztály típusába is beletartozik, mindent „tud" amit az ősei, de ezen felül van még néhány speciális tulajdonsága és képessége is. Egy `FileInfo` osztályú objektum tehát nemcsak `FileInfo`, hanem `FileSystemInfo` (az ősosztály) és `Object` (mindenki ősosztálya) is egyben. A változó típusát tehát úgy kell meghatároznunk, hogy az a tárolni kívánt objektumok közös őse legyen. Ha ez csak az `Object` osztály, akkor nincs mód korlátozásra, a klubnak mindenki tagja lehet. Az alábbi változóba például csak és kizárólag `DirectoryInfo` objektumot tehetünk:

```
PS C:\> [System.IO.DirectoryInfo]$mappa = Get-Item c:\windows
PS C:\> $mappa

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       2007.07.25.      8:06            windows
```

Ha például `FileInfo`-val próbálkozunk (aki pedig elég közeli rokon☺), csak hibaüzenetet kaphatunk:

```
PS C:\> $mappa = Get-Item c:\windows\notepad.exe
Cannot convert the "C:\windows\notepad.exe" value of type "System.IO.FileInfo"
 to type "System.IO.DirectoryInfo".
At line:1 char:7
+ $mappa <<<<  = Get-Item c:\windows\notepad.exe
    + CategoryInfo          : MetadataError: (:) [], ArgumentTransformationMe
   tadataException
    + FullyQualifiedErrorId : RuntimeException
```

Ha mindkét típust tárolni szeretnénk (de semmi mást!), akkor közös őstípusú változót kell létrehoznunk:

```
PS C:\> [System.IO.FileSystemInfo]$mappa = Get-Item c:\windows
PS C:\> [System.IO.FileSystemInfo]$mappa = Get-Item c:\windows\notepad.exe
```

Honnan tudhatjuk meg, hogy egy adott osztálynak mi az őse? Természetesen a .NET SDK dokumentációjából bármikor, de szerencsére nem kell feltétlenül ilyen messzire mennünk. Az ősosztály egyszerűen a PowerShellből is lekérdezhető a következő módon (`BaseType` oszlop):

```
PS C:\> [System.IO.FileInfo]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     FileInfo                                 System.IO.FileSystemInfo
```

Vagy egy objektum megragadása után annak `PSObject` rejtett tulajdonságának, vagy más szóval nézetének `TypeName` tulajdonságával:

```
PS C:\> (Get-Item c:\windows\notepad.exe).psobject.typenames
System.IO.FileInfo
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
System.IO.FileSystemInfo
System.MarshalByRefObject
System.Object
```

Itt egy tömböt kaptunk, amiben az elemeket alulról fölfele kell értelmezni olyan módon, hogy a legősibb osztály a legalsó, azaz a `System.Object`, ennek leszármazottja a `System.MarshalByRefObject` és így tovább. A `PSObject` nézetről kicsit bővebben a *1.4.14 PSBase, PSAdapted, PSExtended, PSObject nézetek* fejezetben lesz szó részletesebben.

1.4.2 Számtípusok

Nézzük akkor részletesebben a típusokat, azok közül is a leggyakoribb számtípusokat és azok jelölését:

Példa (értéktartomány)	.NET teljes típusnév	PowerShell rövid név
12 (±2147483648)	System.Int32	[int]
3.12 (±1.79769313486232e308)	System.Double	[double]
12345678901 (±9223372036854775807)	System.Int64	[long]
15d (±79228162514264337593543950335)	System.Decimal	[decimal]

Az informatikában még gyakran alkalmazunk hexadecimális számokat. Erre külön nincs PowerShellben típus, viszont használhatunk egy nagyon egyszerű jelölést:

```
[20] PS C:\> 0x1000
4096
[21] PS C:\> 0xbaba
47802
[22] PS C:\> 0xfababa
16431802
```

A lebegőpontos számok megadására használhatjuk a matematikában és a számológépeken megszokott formátumot:

```
[23] PS C:\> $lebegő = 1.234e10
[24] PS C:\> $lebegő
12340000000
```

1.4.3 Tömbök

A programnyelvek egyik legalapvetőbb adattípusa a tömb, ami ugye egy olyan változó, amely értékek egy halmazát tartalmazza. Nézzük meg az egyszerű tömböktől kezdve a többdimenziós tömbökig a lehetőségeket!

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.4.3.1 Egyszerű tömbök

A PowerShell nagyon rugalmasan, akár a parancssorban gyorsan begépelhető módon kezeli a tömböket:

```
[17] PS C:\> $egésztömb = 1,2,11,22,100
[18] PS C:\> $egésztömb
1
2
11
22
100
[19] PS C:\> $egésztömb.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Látszik, hogy a tömbök létrehozásához és adatainak megadásához legegyszerűbben a vessző (,) karakter használatos.

A PowerShellben a tömbök nem csak egyforma típusú elemeket tartalmazhatnak:

```
[21] PS C:\> $vegyestömb = "szöveg", 123, 666d, 3.1415
[22] PS C:\> $vegyestömb
szöveg
123
666
3,1415
```

Hogyan lehet egyelemű tömböt létrehozni?

```
[23] PS C:\> $nemegyelemű = "elem"
[24] PS C:\> $nemegyelemű
elem
[25] PS C:\> $nemegyelemű.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object
```

A fenti példában a [23]-as promptnál látszik, hogy egyszerűen egy tagot megadva természetesen – ahogy korábban is láttuk – nem jön létre egyelemű tömb. De egy kis trükkel ez is megoldható:

```
[26] PS C:\> $egyelemű = ,"elem"
[27] PS C:\> $egyelemű
elem
[28] PS C:\> $egyelemű.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Ilyenkor tehát a [26]-os promptban alkalmazott trükköt érdemes használni, azaz az elem elé egy vesszőt kell rakni.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Mi van akkor, ha üres tömböt akarunk létrehozni, mert majd később, egy ciklussal akarjuk feltölteni elemekkel? Ehhez ezt a formátumot lehet használni:

```
[29] PS C:\> $ürestömb = @()
[30] PS C:\> $ürestömb.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Ezt a „`@()`” jelölést természetesen egy- és többelemű tömbök létrehozására is felhasználhatjuk:

```
[31] PS C:\> $eet = @(1)
[32] PS C:\> $eet
1
[33] PS C:\> $eet.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array


[34] PS C:\> $tet = @(1,2,3,4)
[35] PS C:\> $tet
1
2
3
4
[36] PS C:\> $tet.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Ha már ennyit foglalkoztunk tömbökkel, nézzük meg, hogy milyen tulajdonságaik és metódusaik vannak:

```
[37] PS C:\> $tet | Get-Member


   TypeName: System.Int32

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     System.Int32 CompareTo(Object value), System.Int3...
Equals      Method     System.Boolean Equals(Object obj), System.Boolean...
GetHashCode Method     System.Int32 GetHashCode()
GetType     Method     System.Type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode()
ToString    Method     System.String ToString(), System.String ToString(...
```

Hoppá! Ez valahogy nem jó! A négy számot tartalmazó tömbünk esetében a `Get-Member` nem magának a tömbnek, hanem a tömb egyes elemeinek adta meg a tagjellemzőit. Hogyan lehetne rábírni, hogy magának a tömbnek a tagjellemzőit adja vissza? Segíteni kell az előbb látott, egyelemű tömbre vonatkozó (`,`) trükkel:

```
[38] PS C:\> ,$tet | Get-Member
```

```
   TypeName: System.Object[]

Name                MemberType     Definition
----                ----------     ----------
Count               AliasProperty  Count = Length
Address             Method         System.Object& Address(Int32 )
Clone               Method         System.Object Clone()
CopyTo              Method         System.Void CopyTo(Array array, Int32 i...
Equals              Method         System.Boolean Equals(Object obj)
Get                 Method         System.Object Get(Int32 )
GetEnumerator       Method         System.Collections.IEnumerator GetEnume...
GetHashCode         Method         System.Int32 GetHashCode()
GetLength           Method         System.Int32 GetLength(Int32 dimension)
GetLongLength       Method         System.Int64 GetLongLength(Int32 dimens...
GetLowerBound       Method         System.Int32 GetLowerBound(Int32 dimens...
GetType             Method         System.Type GetType()
GetUpperBound       Method         System.Int32 GetUpperBound(Int32 dimens...
GetValue            Method         System.Object GetValue(Params Int32[] i...
get_IsFixedSize     Method         System.Boolean get_IsFixedSize()
get_IsReadOnly      Method         System.Boolean get_IsReadOnly()
get_IsSynchronized  Method         System.Boolean get_IsSynchronized()
get_Length          Method         System.Int32 get_Length()
get_LongLength      Method         System.Int64 get_LongLength()
get_Rank            Method         System.Int32 get_Rank()
get_SyncRoot        Method         System.Object get_SyncRoot()
Initialize          Method         System.Void Initialize()
Set                 Method         System.Void Set(Int32 , Object )
SetValue            Method         System.Void SetValue(Object value, Int3...
ToString            Method         System.String ToString()
IsFixedSize         Property       System.Boolean IsFixedSize {get;}
IsReadOnly          Property       System.Boolean IsReadOnly {get;}
IsSynchronized      Property       System.Boolean IsSynchronized {get;}
Length              Property       System.Int32 Length {get;}
LongLength          Property       System.Int64 LongLength {get;}
Rank                Property       System.Int32 Rank {get;}
SyncRoot            Property       System.Object SyncRoot {get;}
```

Vajon miért nem ez az alapértelmezett működése a `Get-Member`-nek? A PowerShell alkotói próbáltak mindig olyan megoldásokat kitalálni, ami a gyakoribb felhasználási esetekre ad jó megoldást, márpedig inkább az a gyakoribb, hogy egy tömb elemeinek keressük a tagjellemzőit, nem pedig magának a tömbnek.

Van egy másik módszer is, amivel az „egész” objektum tagjellemzőit tudjuk megnézni:

```
[40] PS C:\> Get-Member -InputObject $tet


   TypeName: System.Object[]

Name            MemberType     Definition
----            ----------     ----------
Count           AliasProperty  Count = Length
Address         Method         System.Object&, mscorlib, Version=2.0.0.0, Cul...
Clone           Method         System.Object Clone()
…
```

A fenti tagjellemzőkből nézzük meg a fontosabbakat:

```
[46] PS C:\> $tet.count # elemszám
4
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[47] PS C:\> $tet.length # elemszám
4
[48] PS C:\> $tet.rank # dimenzió
1
[49] PS C:\> $tet.isfixedsize # fix méretű?
True
```

(A '#' jel megjegyzést jelöl, az ez utáni részt a parancsértelmező nem veszi figyelembe.) A fenti listában látjuk, hogy kétféle szintaxissal is lekérhetjük a tömb elemszámát, lekérhetjük, hogy hány dimenziós a tömb és hogy bővíthetjük-e a tömbünk elemszámát. Ez utóbbi szomorú eredményt ad, hiszen azt mondja, hogy ez fixméretű tömb, nem lehet elemeket hozzáadni. Vajon tényleg?

```
[54] PS C:\> $tet
1
2
3
4
[55] PS C:\> $tet += 11
[56] PS C:\> $tet
1
2
3
4
11
```

Azt láthatjuk, hogy az eredetileg négyelemű tömböt minden nehézség nélkül tudtuk öteleműre bővíteni a += operátorral. Ez azonban valójában nem ilyen egyszerűen ment végbe a felszín alatt, hanem a PowerShell létrehozott egy új, üres tömböt és szépen átmásolta az eredeti tömbünk elemeit, majd hozzábiggyesztette az új tagot. Természetesen ezt az új tömböt továbbra is a régi név alatt érjük el, de ez már valójában nem ugyanaz a tömb. Ez akkor érdekes, ha nagyon nagyméretű tömbökkel dolgozunk, hiszen akkor az új tömb felépítése során a művelet végrehajtásának végéig ideiglenesen kétszer is tárolódik a tömb, ami jelentős memória-felhasználást igényelhet.

Megjegyzés

Vigyázzunk a tömbök másolásakor! Merthogy a másolás valójában referencia alapján, azaz a memóriacím alapján történik a háttérben, azaz valójában ugyanaz a tömbünk van kétszer:

```
[1] PS C:\> $tömb1 = "a", "b", "c", "d"
[2] PS C:\> $tömb2 = $tömb1
[3] PS C:\> $tömb2
a
b
c
d
[4] PS C:\> $tömb1[0] = 1
[5] PS C:\> $tömb2
1
b
c
d
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Azaz készítek egy tömböt [1], majd azt „átmásolom” egy másik változóba [2-3], majd módosítom az első tömb első elemét [4]. És mi történt? A második tömb első eleme is változott! [5]
Itt most kapóra jön nekünk viszont ennek a tömbtípusnak a „hiányossága”, azaz hogy nem lehet bővíteni, hanem az elem hozzáadása valójában új tömböt hoz létre:

```
[6] PS C:\> $tömb1 += "e"
[7] PS C:\> $tömb2
1
b
c
d
[8] PS C:\> $tömb1[0] = 2
[9] PS C:\> $tömb2
1
b
c
d
```

Látható, hogy mihelyst új elemet raktam az első tömbbe, az függetlenedett a másodiktól.

A .NET Frameworkben van ennél „okosabb” tömb is, azt is használhatjuk PowerShellben, ez pedig a `System.Collections.ArrayList` típus. Nézzük meg ennek a tagjellemzőit:

```
[60] PS C:\> $scal = New-Object system.collections.arraylist
[61] PS C:\> ,$scal | Get-Member


   TypeName: System.Collections.ArrayList

Name                MemberType            Definition
----                ----------            ----------
Add                 Method                System.Int32 Add(Object value)
AddRange            Method                System.Void AddRange(ICollectio...
BinarySearch        Method                System.Int32 BinarySearch(Int32...
Clear               Method                System.Void Clear()
Clone               Method                System.Object Clone()
Contains            Method                System.Boolean Contains(Object ...
CopyTo              Method                System.Void CopyTo(Array array)...
Equals              Method                System.Boolean Equals(Object obj)
GetEnumerator       Method                System.Collections.IEnumerator ...
GetHashCode         Method                System.Int32 GetHashCode()
GetRange            Method                System.Collections.ArrayList Ge...
GetType             Method                System.Type GetType()
get_Capacity        Method                System.Int32 get_Capacity()
get_Count           Method                System.Int32 get_Count()
get_IsFixedSize     Method                System.Boolean get_IsFixedSize()
get_IsReadOnly      Method                System.Boolean get_IsReadOnly()
get_IsSynchronized  Method                System.Boolean get_IsSynchroniz...
get_Item            Method                System.Object get_Item(Int32 in...
get_SyncRoot        Method                System.Object get_SyncRoot()
IndexOf             Method                System.Int32 IndexOf(Object val...
Insert              Method                System.Void Insert(Int32 index,...
InsertRange         Method                System.Void InsertRange(Int32 i...
LastIndexOf         Method                System.Int32 LastIndexOf(Object...
Remove              Method                System.Void Remove(Object obj)
RemoveAt            Method                System.Void RemoveAt(Int32 index)
RemoveRange         Method                System.Void RemoveRange(Int32 i...
Reverse             Method                System.Void Reverse(), System.V...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
SetRange            Method                 System.Void SetRange(Int32 inde...
set_Capacity        Method                 System.Void set_Capacity(Int32 ...
set_Item            Method                 System.Void set_Item(Int32 inde...
Sort                Method                 System.Void Sort(), System.Void...
ToArray             Method                 System.Object[] ToArray(), Syst...
ToString            Method                 System.String ToString()
TrimToSize          Method                 System.Void TrimToSize()
Item                ParameterizedProperty  System.Object Item(Int32 index)...
Capacity            Property               System.Int32 Capacity {get;set;}
Count               Property               System.Int32 Count {get;}
IsFixedSize         Property               System.Boolean IsFixedSize {get;}
IsReadOnly          Property               System.Boolean IsReadOnly {get;}
IsSynchronized      Property               System.Boolean IsSynchronized {...
SyncRoot            Property               System.Object SyncRoot {get;}
```

A [60]-as promptban létrehozok egy új objektumot a `new-object` cmdlettel, melynek típusa a `System.Collections.ArrayList`, majd ennek kilistázom a tagjellemzőit. Itt már egész fejlett lehetőségeket találunk. Van itt `Add` metódus, amivel lehet elemet hozzáadni, meg van `Contains`, amivel lehet megvizsgálni, hogy egy adott érték benne van-e a tömbben, az `Insert` metódussal be lehet szúrni elemeket, a `Remove`-val el lehet távolítani, a `Reverse`-zel meg lehet fordítani az elemek sorrendjét, a `Sort`-tal sorba lehet rendezni:

```
[67] PS C:\> $scal = [system.collections.arraylist] (1,2,3,4,5)
[68] PS C:\> $scal
1
2
3
4
5
[69] PS C:\> $scal.Contains(4)
True
[70] PS C:\> $scal.add(1000)
5
[71] PS C:\> $scal
1
2
3
4
5
1000
[72] PS C:\> $scal.insert(3, 200)
[73] PS C:\> $scal
1
2
3
200
4
5
1000
[74] PS C:\> $scal.reverse()
[75] PS C:\> $scal
1000
5
4
200
3
2
1
```

```
[76] PS C:\> $scal.sort()
[77] PS C:\> $scal
1
2
3
4
5
200
1000
```

Az `Add`-nél vigyázni kell, hogy kimenetet ad, méghozzá azt a számot, amelyik elemként tette hozzá az éppen hozzáadott elemet. Ha ez nem kell, akkor el lehet „fojtani” a kimenetet az alábbi két módszer valamelyikével:

```
[78] PS C:\> [void] $scal.add(1234)

[79] PS C:\> $scal.add(2345) > $null
```

A [78]-es promptban átkonvertáljuk a kimenetet `[void]` típussá, azaz semmivé. A [79]-as sorban pedig átirányítjuk a semmibe a kimenetet.

Megjegyzés

Nem mindegy, hogy hogyan adunk elemet az „okos” tömbünkhöz. Az előző megjegyzésben láthattuk, hogy a tömbök másolása változók között valójában csak a tömb címét másolja, mindkét változó ugyanarra a tömbre mutat. Ugyanez igaz a `collections.arraylist` tömbre is. Ez az ugyanoda mutatás megszűnik, ha a `+=` operátorral bővítem a tömböt, viszont nem szűnik meg, ha az Add() metódust használjuk:

```
[11] PS C:\> $ot1 = [collections.arraylist] (1,2,3,4,5)
[12] PS C:\> $ot2 = $ot1
[13] PS C:\> $ot1[0]="a"
[14] PS C:\> $ot2
a
2
3
4
5
[15] PS C:\> $ot1 += 6
[16] PS C:\> $ot2
a
2
3
4
5
[17] PS C:\> $ot1 = [collections.arraylist] (1,2,3,4,5)
[18] PS C:\> $ot2 = $ot1
[19] PS C:\> $ot1.add(6)
5
[20] PS C:\> $ot2
1
2
3
4
5
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
6
```

A fenti példában látható a [19]-es sorban, hogy az `Add()` metódus hatása az `$ot2`-re is megvolt.

Most már csak egy dolgot nem mutattam meg, hogy hogyan lehet hivatkozni a tömbelemekre:

```
[91] PS C:\> $scal[0] # első elem
1
[92] PS C:\> $scal[2..5] # harmadiktól hatodik elemig
3
4
5
200
[93] PS C:\> $scal[5..2] # hatodiktól harmadik elemig
200
5
4
3
```

A fenti példákban látható, hogy a tömbök első elemére a 0-s indexszel lehet hivatkozni. Egyszerre több egymás utáni elemre a (`..`) *range*, azaz tartomány operátorral. Ugyan operátorokkal később foglalkozom, de ez a „range” operátor annyira kötődik a tömbökhöz, hogy érdemes itt tárgyalni. Az alábbi példa mutatja az alapvető működését:

```
[94] PS C:\> 1..10
1
2
3
4
5
6
7
8
9
10
```

A `range` segítségével egy másik, érdekes módon is lehet hivatkozni a tömbelemekre, de az csak a „hagyományos” `array`-re működik:

```
[95] PS C:\> $a=1,2,3,4,10,8
[96] PS C:\> $a
1
2
3
4
10
8
[97] PS C:\> $a[-1..-3]
8
10
4
```

A [97]-es prompt azt mutatja, hogy értelmezett a negatív index, amit a PowerShell a tömb utolsó elemétől visszafele számol. A -1. elem az utolsó elem, -2. az utolsó előtti és így tovább.

tanfolyami felhasználása nem engedélyezett!

Ha több elem kellene egyszerre? Adjunk meg bátran több indexet egyszerre! Ebben az esetben természetesen a visszakapott érték is egy tömb lesz.

```
PS C:\> $b = 1..100
PS C:\> $b[5,7,56]
6
8
57
```

Az már csak hab a tortán, hogy nemcsak konkrét indexet, hanem intervallumot (sőt akár több intervallumot) adhatunk meg ebben az esetben is.

```
PS C:\> $b = 1..100
PS C:\> $b[2..4 + 56..58]
3
4
5
57
58
59
```

1.4.3.2 Többdimenziós tömbök

Természetesen egy tömbnek nem csak egyirányú kiterjedése lehet, tudunk többdimenziós tömböket is létrehozni. Az alábbi példában úgy érem el a kétirányú kiterjedést, hogy a `$tábla` tömb elemeiként szintén tömböket teszek:

```
[1] PS C:\> $tábla = (1,2,3,4),("a","b","c","d")
[2] PS C:\> $tábla
1
2
3
4
a
b
c
d
[3] PS C:\> $tábla[0][0]
1
[4] PS C:\> $tábla[0][1]
2
[5] PS C:\> $tábla[1][0]
a
```

Látszik, hogy ilyenkor két indexszel hivatkozhatunk a „dimenziókra”. Sőt! Nem csak egyforma „hosszú” sorokból állhat egy „kvázi” kétdimenziós tömb:

```
[7] PS C:\> $vegyes = (1,2,3),("a","b"),("egy","kettő","három","négy")
[8] PS C:\> $vegyes[0][0]
1
[9] PS C:\> $vegyes[0][2]
3
[10] PS C:\> $vegyes[0][3]
[11] PS C:\> $vegyes[1][1]
b
[12] PS C:\> $vegyes[1][2]
```

```
[13] PS C:\> $vegyes[2][3]
négy
```

Az „igazi” többdimenziós tömböt az alábbi szintaxissal lehet hivatalosan létrehozni, és ilyenkor másként kell hivatkozni a tömbelemekre:

```
[20] PS C:\> $igazi = new-object 'object[,]' 3,2
[21] PS C:\> $igazi[2,1]="kakukk"
```

Természetesen nem csak kettő, hanem akárhány dimenziós lehet egy tömb, de ilyet valószínű csak a robottechnikában használnak. Példa egy tízdimenziós tömbre:

```
[23] PS C:\> $igazi = new-object 'object[,,,,,,,,,]' 8,3,7,5,6,7,8,9,10,3
```

1.4.3.3 Típusos tömbök

Tudunk létrehozni típusos tömböket, amelyek csak az adott típusú elemeket tartalmazhatnak:

```
[24] PS C:\> $t = New-Object int[] 20
[25] PS C:\> $t[1]="szöveg"
Array assignment to [1] failed: Cannot convert value "szöveg" to type "Sy
stem.Int32". Error: "Input string was not in a correct format.".
At line:1 char:4
+ $t[ <<<< 1]="szöveg"
    + CategoryInfo          : InvalidOperation: (szöveg:String) [], Runt
   imeException
    + FullyQualifiedErrorId : ArrayAssignmentFailed
[26] PS C:\> $t += 2
```

A fenti példában látszik, hogy létrehozunk előre egy 20 elemű `int` típusú tömböt, amibe ha szöveget akarunk betölteni, akkor hibát kapunk. Azonban ha új elemet biggyesztünk hozzá, akkor az már lehet akármilyen típusú.

Természetesen típusos tömbökből többdimenziósakat is létre tudunk hozni:

```
[27] PS C:\> $ttdt = New-Object 'int[,]' 5,2
[28] PS C:\> $ttdt[0,0] = 12
[29] PS C:\> $ttdt[1,1] = "próba"
Array assignment to [1,1] failed: Cannot convert value "próba" to type "System
.Int32". Error: "Input string was not in a correct format.".
At line:1 char:7
+ $ttdt[ <<<< 1,1] = "próba"
    + CategoryInfo          : InvalidOperation: (próba:String) [], RuntimeExc
   eption
    + FullyQualifiedErrorId : ArrayAssignmentFailed
```

Itt tehát annyi a különbség, hogy nem általános `object`-ek a tömb elemei, hanem itt a példában `int` típus. A [29]-es sorban nem is szerette, ha sztringet akartam beletenni.

1.4.3.4 Generic adattípus – paraméterezhető típusos tömb

A .NET keretrendszer tartalmaz egy olyan tömböt, aminek paraméterként adható át, hogy milyen elemeket tartalmazhasson:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[1] PS C:\> $genlist = New-Object collections.generic.list[string]
[2] PS C:\> get-member -i $genlist


   TypeName: System.Collections.Generic.List`1[[System.String, mscorlib, Versi
on=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]

Name           MemberType            Definition
----           ----------            ----------
Add            Method                System.Void Add(string item)
AddRange       Method                System.Void AddRange(System.Collections...
AsReadOnly     Method                System.Collections.ObjectModel.ReadOnly...
BinarySearch   Method                int BinarySearch(int index, int count, ...
Clear          Method                System.Void Clear()
Contains       Method                bool Contains(string item)
ConvertAll     Method                System.Collections.Generic.List[TOutput...
CopyTo         Method                System.Void CopyTo(string[] array), Sys...
Equals         Method                bool Equals(System.Object obj)
Exists         Method                bool Exists(System.Predicate[string] ma...
Find           Method                string Find(System.Predicate[string] ma...
FindAll        Method                System.Collections.Generic.List[string]...
FindIndex      Method                int FindIndex(System.Predicate[string] ...
FindLast       Method                string FindLast(System.Predicate[string...
FindLastIndex  Method                int FindLastIndex(System.Predicate[stri...
ForEach        Method                System.Void ForEach(System.Action[strin...
GetEnumerator  Method                System.Collections.Generic.List`1+Enume...
GetHashCode    Method                int GetHashCode()
GetRange       Method                System.Collections.Generic.List[string]...
GetType        Method                type GetType()
IndexOf        Method                int IndexOf(string item), int IndexOf(s...
Insert         Method                System.Void Insert(int index, string item)
InsertRange    Method                System.Void InsertRange(int index, Syst...
LastIndexOf    Method                int LastIndexOf(string item), int LastI...
Remove         Method                bool Remove(string item)
RemoveAll      Method                int RemoveAll(System.Predicate[string] ...
RemoveAt       Method                System.Void RemoveAt(int index)
RemoveRange    Method                System.Void RemoveRange(int index, int ...
Reverse        Method                System.Void Reverse(), System.Void Reve...
Sort           Method                System.Void Sort(), System.Void Sort(Sy...
ToArray        Method                string[] ToArray()
ToString       Method                string ToString()
TrimExcess     Method                System.Void TrimExcess()
TrueForAll     Method                bool TrueForAll(System.Predicate[string...
Item           ParameterizedProperty string Item(int index) {get;set;}
Capacity       Property              System.Int32 Capacity {get;set;}
Count          Property              System.Int32 Count {get;}
```

Látható, hogy a `$genlist` objektum létrehozásakor meghatároztam, hogy ez egy olyan típusú tömb legyen, amelynek elemei sztringek lehetnek. A [2]-es sorban kilistáztam a tagjellemzőit ennek az objektumnak és láthatjuk, hogy ez is „okos” tömb, azaz itt is van `Add()` metódus és sok más hasznos dolog.

1.4.4 Szótárak (hashtáblák) és szótártömbök

A strukturált adatszerkezetek esetében nagyon praktikusan használható adattípus a hashtable, vagy magyarul talán szótárnak vagy asszociatív tömbnek lehetne hívni.

```
[9] PS C:\> $hash = @{ Név = "Soós Tibor"; Cím = "Budapest"; "e-mail"="so
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
ostibor@citromail.hu"}
[10] PS C:\> $hash

Name                           Value
----                           -----
Név                            Soós Tibor
e-mail                         soostibor@citromail.hu
Cím                            Budapest
```

A hashtábla jelölése tehát egy kukac-kapcsos zárójel pár (@{}) jellel és egy lezáró kapcsos zárójellel (}) jellel történik. Belül *kulcs=érték* párokat kell elhelyezni. A kulcs nevét csak akkor kell idézőjelezni, ha az valami speciális karaktert (pl. szóköz, kötőjel, stb.) tartalmaz. Az érték megadásánál az eddig megszokott formákat kell alkalmazni.

Hogyan tudok vajon egy újabb személyt felvenni ebbe a hashtáblába? Nézzük meg ehhez a hashtábla tagjellemzőit:

```
[12] PS C:\> $hash | Get-Member | ft -wrap


   TypeName: System.Collections.Hashtable

Name                MemberType            Definition
----                ----------            ----------
Add                 Method                System.Void Add(Object key, Object
                                           value)
Clear               Method                System.Void Clear()
Clone               Method                System.Object Clone()
Contains            Method                System.Boolean Contains(Object key
                                           )
ContainsKey         Method                System.Boolean ContainsKey(Object
                                           key)
ContainsValue       Method                System.Boolean ContainsValue(Objec
                                           t value)
CopyTo              Method                System.Void CopyTo(Array array, In
                                           t32 arrayIndex)
Equals              Method                System.Boolean Equals(Object obj)
GetEnumerator       Method                System.Collections.IDictionaryEnum
                                           erator GetEnumerator()
GetHashCode         Method                System.Int32 GetHashCode()
GetObjectData       Method                System.Void GetObjectData(Serializ
                                           ationInfo info, StreamingContext c
                                           ontext)
GetType             Method                System.Type GetType()
get_Count           Method                System.Int32 get_Count()
get_IsFixedSize     Method                System.Boolean get_IsFixedSize()
get_IsReadOnly      Method                System.Boolean get_IsReadOnly()
get_IsSynchronized  Method                System.Boolean get_IsSynchronized(
                                           )
get_Item            Method                System.Object get_Item(Object key)
get_Keys            Method                System.Collections.ICollection get
                                           _Keys()
get_SyncRoot        Method                System.Object get_SyncRoot()
get_Values          Method                System.Collections.ICollection get
                                           _Values()
OnDeserialization   Method                System.Void OnDeserialization(Obje
                                           ct sender)
Remove              Method                System.Void Remove(Object key)
set_Item            Method                System.Void set_Item(Object key, O
```

```
                                        bject value)
ToString            Method              System.String ToString()
Item                ParameterizedProperty System.Object Item(Object key) {ge
                                        t;set;}
Count               Property            System.Int32 Count {get;}
IsFixedSize         Property            System.Boolean IsFixedSize {get;}
IsReadOnly          Property            System.Boolean IsReadOnly {get;}
IsSynchronized      Property            System.Boolean IsSynchronized {get
                                        ;}
Keys                Property            System.Collections.ICollection Key
                                        s {get;}
SyncRoot            Property            System.Object SyncRoot {get;}
Values              Property            System.Collections.ICollection Val
                                        ues {get;}
```

Láthatjuk, az `Add` metódust, így nézzük meg, azzal mit kapunk:

```
[13] PS C:\> $hash.Add("Név","Fájdalom Csilla")
Exception calling "Add" with "2" argument(s): "Item has already been added. Ke
y in dictionary: 'Név'  Key being added: 'Név'"
At line:1 char:10
+ $hash.Add <<<< ("Név","Fájdalom Csilla")
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
```

Hát erre bizony hibajelzést kaptam, mert nem szereti ez az adatszerkezet, ha ugyanolyan kulccsal még egy értéket akarok felvenni. Csak megváltoztatni tudom az adott kulcshoz tartozó értéket:

```
[17] PS C:\> $hash.set_Item("Név","Fájdalom Csilla")
[18] PS C:\> $hash

Name                           Value
----                           -----
e-mail                         soostibor@citromail.hu
Cím                            Budapest
Név                            Fájdalom Csilla


[19] PS C:\> $hash["Név"]="Beléd Márton"
[20] PS C:\> $hash

Name                           Value
----                           -----
e-mail                         soostibor@citromail.hu
Cím                            Budapest
Név                            Beléd Márton


[21] PS C:\> $hash.Név = "Pandacsöki Boborján"
[22] PS C:\> $hash

Name                           Value
----                           -----
e-mail                         soostibor@citromail.hu
Cím                            Budapest
Név                            Pandacsöki Boborján
```

A fenti példában látszik, hogy három különböző szintaxissal is lehet módosítani értékeket ([17], [19] és [21]-es sorok).

Lekérdezni az értékeket is a fenti lehetőségekhez hasonlóan lehet:

```
[27] PS C:\> $hash.Név
Pandacsöki Boborján
[28] PS C:\> $hash["Név"]
Pandacsöki Boborján
[29] PS C:\> $hash.get_item("Név")
Pandacsöki Boborján
```

Szóval lehetőségek elég széles tárháza áll rendelkezésre. Külön lekérhetjük a hashtábla kulcsait és értékeit is:

```
[32] PS C:\> $hash.keys
e-mail
adat
Cím
Név
[33] PS C:\> $hash.values
soostibor@citromail.hu
12345
Budapest
Pandacsöki Boborján
```

De visszatérve a problémámhoz, hogyan lehet még egy ember adatait berakni ebbe a hashtáblába? Nagyon egyszerűen, hashtábla-tömböt kell létrehozni, méghozzá abból a `System.Collections.ArrayList` fajtából, amit tudunk bővíteni:

```
[37] PS C:\> $névjegyek = New-Object system.collections.arraylist
[38] PS C:\> $névjegyek.Add(@{Név="Soós Tibor";
"e-mail"="soostibor@citromail.hu";Cím="Budapest"})
0
[39] PS C:\> $névjegyek.Add(@{Név="Fájdalom Csilla";
"e-mail"="fcs@citromail.hu";Cím="Zamárdi"})
1
[40] PS C:\> $névjegyek

Name                           Value
----                           -----
e-mail                         soostibor@citromail.hu
Cím                            Budapest
Név                            Soós Tibor
e-mail                         fcs@citromail.hu
Cím                            Zamárdi
Név                            Fájdalom Csilla


[41] PS C:\> $névjegyek.count
2
[42] PS C:\> $névjegyek[0]

Name                           Value
----                           -----
e-mail                         soostibor@citromail.hu
Cím                            Budapest
Név                            Soós Tibor
```

```
[43] PS C:\> $névjegyek[1]

Name                           Value
----                           -----
e-mail                         fcs@citromail.hu
Cím                            Zamárdi
Név                            Fájdalom Csilla


[44] PS C:\> $névjegyek[1].Név
Fájdalom Csilla
[45] PS C:\> $névjegyek[1]."e-mail"
fcs@citromail.hu
```

Így ezzel a hashtábla-tömbbel adatbázis-szerű alkalmazási területek adatleképezési igényeit is nagyon jól ki lehet elégíteni.

Hashtáblák definiálására a 2.0-ás PowerShellben egy új cmdlet is rendelkezésünkre áll, hogy még egyszerűbben definiálhassunk ilyen adattípust, ez pedig a `ConvertFrom-StringData`:

```
[58] PS C:\> $hashstring = @"
>> első = Ez itt az első érték
>> második = az érték részt nem is kell idézőjelezni
>> harmadik = soostibor@citromail.hu
>> negyedik = 4
>> ötödik = 2009.11.21
>> "@
>>
[59] PS C:\> $hash = ConvertFrom-StringData $hashstring
[60] PS C:\> $hash

Name                           Value
----                           -----
negyedik                       4
második                        az érték részt nem is kell idézőjelezni
ötödik                         2009.11.21
harmadik                       soostibor@citromail.hu
első                           Ez itt az első érték


[61] PS C:\> $hash.első
Ez itt az első érték
[62] PS C:\> ($hash.negyedik).gettype().fullname
System.String
[63] PS C:\> ($hash.ötödik).gettype().fullname
System.String
```

Mint ahogy látható, a hashtábla egy többsoros sztringből képződik. A sztring egyes sorait úgy kell felépíteni, hogy bal oldalon legyen egy kulcsnév, aztán egy egyenlőségjel és jobb oldalon egy érték. Vigyázat, ezen a módon csak szöveges értékeket tudunk a hashtáblába felvenni. Hiába adtam meg számnak és dátumnak kinéző sorokat a negyedik és ötödik kulcsként, ahogy a [62]-es ás [63]-as sorokban látszik ezek mind egyszerű szövegként lettek betöltve a hashtáblába.

Fontos megjegyezni, hogy a hashtáblák kiíratásakor az egyes kulcsok nem előre meghatározott módon íródnak ki. Maga a „hash” szó, azaz magyarul „kivonat” onnan jön az adattípus nevében, hogy a belső tárolási

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

módja ezeknek úgy történik, hogy a kulcsból képződik egy kivonat, ami alapján lesznek az adatok ténylegesen tárolva, és ez alapján gyorsan tudja ellenőrizni a .NET keretrendszer, hogy az adott kulcs már létezik-e, illetve az adatok kiolvasásában, a kulcsok megkeresésében is van szerepe.

Alaphelyzetben tehát össze-vissza az elemek sorrendje:

```
 [46] PS C:\> $hash = @{"ab" = 1; "bb" = 2; "cb" = 3; "db" =4}
[47] PS C:\> $hash

Name                           Value
----                           -----
cb                             3
ab                             1
bb                             2
db                             4
```

Hogyan lehetne ABC rendben kiíratni a hashtábla adatait?

```
[48] PS C:\> $hash.keys | Sort-Object | select-object -Property @{n="Name"; e={
$_}}, @{n="Value"; e={$hash.$_}}

Name                                                                   Value
----                                                                   -----
ab                                                                         1
bb                                                                         2
cb                                                                         3
db                                                                         4
```

Nem túl szép kifejezés ez… Ha mindenképpen fontos a sorrendiség, akkor használhatjuk a .NET keretrendszer egy másik osztályát, a `System.Collections.SortedList`-et, ami nagyon hasonlít a hashtáblához:

```
[54] PS C:\> $sl = new-object system.collections.sortedlist
[55] PS C:\> $sl.ab = 1
[56] PS C:\> $sl.bb = 2
[57] PS C:\> $sl.cd = 3
[58] PS C:\> $sl.db = 4
[59] PS C:\> $sl

Name                           Value
----                           -----
ab                             1
bb                             2
cd                             3
db                             4
```

A fő különbség, hogy itt a kulcsok abc rendjében jelennek meg az elemek, sőt, ha beillesztek egy új elemet, akkor azt is természetesen az abc sorrendben jeleníti meg:

```
[62] PS C:\> $sl.aa = "új elem"
[63] PS C:\> $sl

Name                           Value
----                           -----
aa                             új elem
ab                             1
bb                             2
```

```
cd                                3
db                                4
```

1.4.4.1 A szótár általános adattípusa

Ahogy a típusos tömbnél láttuk az általános (generic) adattípust, úgy a hashtáblának is van ilyenje. Ezzel olyan speciális hashtáblát tudunk létrehozni, melynek akár a kulcs mezőjének, akár az érték mezőjének típusát meg tudjuk szabni.

```
[14] PS C:\> $genhash = New-Object "collections.generic.dictionary[string,int]"

[15] PS C:\> $genhash.elem1 = 1
[16] PS C:\> $genhash.elem2 = "valami"
The value "valami" is not of type "System.Int32" and cannot be used in this ge
neric collection.
Parameter name: value
At line:1 char:10
+ $genhash. <<<< elem2 = "valami"
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : PropertyAssignmentException
```

A fenti példában olyan hashtáblát hoztam létre, ahol a kulcs mező csak sztring lehet, és az értékek meg csak egészek. Ha egész helyett egy szöveget akartam betölteni, akkor hibát kaptam.

1.4.5 Dátumok ([datetime], Get-Date, Set-Date)

A dátumok kezelésével kapcsolatban egy alapszabályt mindenképpen be kell tartanunk: **soha** ne kezdjünk el kézihajtány módszerrel dátumokat faragni, a beépített metódusok és tulajdonságok használata nem csak egyszerűbb megoldáshoz vezet, de a számtalan hibalehetőség elkerülésének érdekében egyenesen kötelező.

Mit tud a PowerShell a dátumokkal kapcsolatban? Igazság szerint nem túl sokat, mindössze két egyszerű cmdletet kapunk: A `Get-Date` segítségével az aktuális rendszerdátumot és időt kérdezhetjük le, valamint `[datetime]` típusú adatot állíthatunk elő; a `Set-Date` cmdlet pedig ezek beállítását képes elvégezni. A háttérben tehát mindig ott van a .NET `DateTime` osztálya; ennek segítségével már bármit megtehetünk.

Próbáljunk meg készíteni dátumot a következő karakterláncból: „2009. november 18.”, és alakítsuk át „2009.11.18” alakra (természetesen úgy, hogy bármilyen dátumra működjön)!

Dátumot leíró karakterláncok beolvasására a `DateTime` osztály statikus `Parse()` metódusa szolgál. A `Parse()` egy mindenevő metódus, ismeri és beolvassa az összes szokásos dátumformátumot:

```
PS C:\> PS C:\> [DateTime]::Parse("2009. november 18.")

2009. november 18. 0:00:00
```

A fenti kifejezés egy `DateTime` objektumot ad vissza (aki nem hiszi, annak `Get-Member` segít), ennek egy metódusát kell meghívnunk, hogy az egyszerűbb alakot előállítsuk (az eredmény már nem dátum, hanem karakterlánc!):

```
[29] PS C:\> ([DateTime]::Parse("2009. november 18.")).ToShortDateString()

2009.11.18.
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

A `[datetime]` konstruktora is képes némi dátumértelmezésre, azonban nem olyan okos, mint a `Parse` metódus:

```
[31] PS C:\> $d = [datetime] "2009.11.18."
[32] PS C:\> $d

2009. november 18. 0:00:00
[33] PS C:\> $d = [datetime] "2009. május 18."
Cannot convert value "2009. május 18." to type "System.DateTime". Error:
"The string was not recognized as a valid DateTime. There is a unknown wo
rd starting at index 6."
At line:1 char:16
+ $d = [datetime] <<<<  "2009. május 18."
    + CategoryInfo          : NotSpecified: (:) [], RuntimeException
    + FullyQualifiedErrorId : RuntimeException
```

A [31]-es sorban létrehozott `[datetime]` típusú változóban az ottani, magyar nyelvű formában szövegként megadott dátumot képes volt értelmezni, de a [33]-as sor szövegét már nem.

Hogyan lehetne megállapítani, hogy milyen napra esik az aktuális dátum 13 év múlva? A 13 évnyi időutazás a PowerShellben nem lehet probléma: egy `Get-Date` eredményére meg kell hívnunk az `AddYears()` metódust. A hét napjának nevét pedig a `DayOfWeek` tulajdonság adja vissza:

```
[36] PS C:\> (get-date).AddYears(13).DayOfWeek
Friday
```

A `Get-Date` használható `[datetime]` objektumok konstruktoraként is:

```
[37] PS C:\> $d = Get-Date -year 2009 -month 11 -day 18
[38] PS C:\> $d

2009. november 18. 11:12:06
```

A fenti példában a `Get-Date` cmdlettel majdnem ugyanazt értem el, mint a megjegyzés [31]-es sorában. Egy fontos különbség van: míg a korábbi példában a nem megadott óra, perc, másodperc érték nulla lett, addig a `Get-Date` használatával az aktuális óra, perc, másodperc érték helyettesítődik be, ami esetleg nem kívánatos a programunk működése szempontjából.

Most próbáljuk meg kilistázni azokat a folyamatokat, amelyek az elmúlt 1 órán belül indultak el a számítógépen! Először is le kell gyártanunk az egy órával ezelőtti időt, a változatosság kedvéért használjuk most a `DateTime` osztály statikus `Now` tulajdonságát. (Statikus tagokkal részletesebben majd a *1.4.9 .NET típusok, statikus tagok* fejezetben lesz szó.) A második sor azokat a `Process` objektumokat válogatja le, amelyeknek `StartTime` tulajdonságában ennél későbbi időpont szerepel.

```
[54] PS C:\> $ora = [DateTime]::Now.AddHours(-1)
[55] PS C:\> Get-Process | Where-Object {$_.StartTime -ge $ora}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     52       6     1208       3948    61     0,06   1644 notepad
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A második sorban az `$ora` változó helyére természetesen beírhattuk volna magát a kifejezést is, a két parancs csak a jobb áttekinthetőség miatt került külön sorba.

Hogyan lehetne visszaállítani a számítógép óráját 10 perccel? A rendszeridőt a `Set-Date` cmdlet segítségével tologathatjuk, paraméterként `TimeSpan` objektumot (lásd később), vagy az adott területi beállítások mellett időintervallumként értelmezhető karakterláncot is megadhatunk. A megoldás tehát:

```
[56] PS C:\> Set-Date -adjust -0:10:0

2009. november 18. 11:18:38
```

Nézzük meg kicsit részletesebben a [datetime] adattípus tagjellemzőit:

```
[64] PS C:\> $d = get-date
[65] PS C:\> $d | Get-Member


   TypeName: System.DateTime

Name                 MemberType     Definition
----                 ----------     ----------
Add                  Method         System.DateTime Add(System.TimeSpan value)
AddDays              Method         System.DateTime AddDays(double value)
AddHours             Method         System.DateTime AddHours(double value)
AddMilliseconds      Method         System.DateTime AddMilliseconds(double ...
AddMinutes           Method         System.DateTime AddMinutes(double value)
AddMonths            Method         System.DateTime AddMonths(int months)
AddSeconds           Method         System.DateTime AddSeconds(double value)
AddTicks             Method         System.DateTime AddTicks(long value)
AddYears             Method         System.DateTime AddYears(int value)
CompareTo            Method         int CompareTo(System.Object value), int...
Equals               Method         bool Equals(System.Object value), bool ...
GetDateTimeFormats   Method         string[] GetDateTimeFormats(), string[]...
GetHashCode          Method         int GetHashCode()
GetType              Method         type GetType()
GetTypeCode          Method         System.TypeCode GetTypeCode()
IsDaylightSavingTime Method         bool IsDaylightSavingTime()
Subtract             Method         System.TimeSpan Subtract(System.DateTim...
ToBinary             Method         long ToBinary()
ToFileTime           Method         long ToFileTime()
ToFileTimeUtc        Method         long ToFileTimeUtc()
ToLocalTime          Method         System.DateTime ToLocalTime()
ToLongDateString     Method         string ToLongDateString()
ToLongTimeString     Method         string ToLongTimeString()
ToOADate             Method         double ToOADate()
ToShortDateString    Method         string ToShortDateString()
ToShortTimeString    Method         string ToShortTimeString()
ToString             Method         string ToString(), string ToString(stri...
ToUniversalTime      Method         System.DateTime ToUniversalTime()
DisplayHint          NoteProperty   Microsoft.PowerShell.Commands.DisplayHi...
Date                 Property       System.DateTime Date {get;}
Day                  Property       System.Int32 Day {get;}
DayOfWeek            Property       System.DayOfWeek DayOfWeek {get;}
DayOfYear            Property       System.Int32 DayOfYear {get;}
Hour                 Property       System.Int32 Hour {get;}
Kind                 Property       System.DateTimeKind Kind {get;}
Millisecond          Property       System.Int32 Millisecond {get;}
Minute               Property       System.Int32 Minute {get;}
Month                Property       System.Int32 Month {get;}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Second                Property       System.Int32 Second {get;}
Ticks                 Property       System.Int64 Ticks {get;}
TimeOfDay             Property       System.TimeSpan TimeOfDay {get;}
Year                  Property       System.Int32 Year {get;}
DateTime              ScriptProperty System.Object DateTime {get=if ((& { Se...
```

Látható, hogy számos metódus és tulajdonság áll rendelkezésünkre a dátum-idő típusú adatok kezelésére. A rendszer mélyén ezek az adatok egész számként, un. ketyegésekben vannak tárolva, ehhez a `Ticks` tulajdonságon keresztül férünk hozzá:

```
[66] PS C:\> $d.ticks
634029707852086250
```

Ez nem egy apró szám! Az időszámításunk óta eltelt 100 ns-okban mért idő.

Megjegyzés

Érdekes módon ez más algoritmus az AD-ben használatos „longint” formátumú időtároláshoz képest, hiszen ott 1601. január 1. 0:00 a kiindulási időpont és ehhez képest veszi 100 ns-okban az eltelt időt.

1.4.6 Időtartam számítás (New-TimeSpan)

A `New-TimeSpan` cmdlet segítségével dátum, illetve időintervallumokat adhatunk meg. A alábbi parancs például a 2009. január 1-e óta eltelt időt adja vissza:

```
[2] PS C:\> New-TimeSpan (Get-Date -month 1 -day 1 -year 2009) (get-date)


Days              : 321
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 0
Ticks             : 277344000000000
TotalDays         : 321
TotalHours        : 7704
TotalMinutes      : 462240
TotalSeconds      : 27734400
TotalMilliseconds : 27734400000
```

A cmdletnek két `DateTime` objektumot kell paraméterként adnunk, a visszaadott érték pedig egy `TimeSpan` objektum, amely a két dátum közötti különbséget tartalmazza. A `TimeSpan` objektumtól természetesen egyesével is elkérhetjük a fenti értékek bármelyikét a megfelelő tulajdonság nevére való hivatkozással:

```
[3] PS C:\> (New-TimeSpan (Get-Date -month 1 -day 1 -year 2009) (get-date)
).ticks
277344000000000
```

A `Ticks` a korábban már látott 100 ns-okban jelzi az eltelt időt.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.4.7 Automatikus típuskonverzió

Korábban már láttuk, hogy a PowerShell megpróbálja automatikusan megváltoztatni az objektumok típusát, ha szükséges, a minél kényelmesebb, egyszerűbb parancsbevitel érdekében:

```
[1] PS I:\>1+2.0+"3"
6
[2] PS I:\>(1+2.0+"3").GetType().FullName
System.Double
```

Az [1]-es promptban látszik, hogy össze tudok adni egy egész számot, egy lebegőpontos számot egy „szöveg” formátumú számmal anélkül, hogy nekem kellene típuskonverziót végezni. A PowerShell ezt helyettem elvégzi. Megnézi, hogy a művelet tagjait vajon át lehet-e alakítani olyan típusra, amellyel egyrészt a művelet elvégezhető, másrészt nem történik adatvesztés. Erre a célra ebben az esetben a `System.Double` típus alkalmas, így a PowerShell minden tagot erre konvertál, illetve a végeredményt is ilyen formában adja meg.

Ez nem csak a matematikai műveletekre igaz, hanem az összehasonlításokra is:

```
[12] PS I:\>15 -eq 15d
True
[13] PS I:\>15.0 -eq 15d
True
[14] PS I:\>15 -eq "15"
True
[15] PS I:\>(15).GetType().FullName
System.Int32
[16] PS I:\>(15d).GetType().FullName
System.Decimal
[17] PS I:\>(15.0).GetType().FullName
System.Double
[18] PS I:\>("15").GetType().FullName
System.String
```

Azaz az egyenlőségvizsgálat (`-eq`) nem alkalmas arra, hogy a nem egyforma típusú objektumokat kiszűrhessük segítségével, hiszen a PowerShell típuskonverziót végezhet. Részletesebben az összehasonlítási lehetőségekkel az *1.5.3 Összehasonlító operátorok* fejezetben foglalkozom.

Az automatikus típuskonverziónál fontos észben tartani, hogy a kifejezések szigorúan balról jobbra értékelődnek ki, így nem mindegy, hogy milyen sorrendben adjuk meg a műveleteink paramétereit:

```
[20] PS C:\> 1+"2"
3
[21] PS C:\> "2"+1
21
```

A [20]-as sorban először egy `int` típusú számmal találkozik a parancselemző, ehhez próbálja hozzáigazítani a szöveges formátumban megadott „2”-t és így végzi el a műveletet, amelynek eredménye 3 lett.

A [21]-es sorban a szöveges „2”-höz igazítja az 1-et, amit szintén szöveggé alakít, és a két szöveg összeadásának, azaz összeillesztésének eredményét adja meg, ami „21” lett. Láthatjuk, hogy ennek eredménye tényleg sztring típusú:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[22] PS C:\> ("2"+1).gettype().fullname
System.String
```

1.4.8 Típuskonverzió

Előzőekben mutattam, hogy a PowerShell a kifejezésekben megpróbálja a triviális típuskonverziókat elvégezni, olyan típusúvá próbálja konvertálni a tagokat, amely nem jár információveszteséggel.

Ha mi magunk akarunk típuskonverziót elvégezni és nem a PowerShell automatizmusára bízni a kérdést, akkor erre is van lehetőség:

```
[1] PS C:\> $a = "1234"
[2] PS C:\> $b = [int] $a
[3] PS C:\> $b.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType


[4] PS C:\> $c = 4321
[5] PS C:\> $d = [string] $c
[6] PS C:\> $d.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object
```

Ennél trükkösebb dolgokat is lehet típuskonverzió segítségével elvégezni. Például az Exchange Server 2007-ben van egy „rejtélyes” karaktersorozat az Administrative Group objektumnál. Próbáljuk ezt megfejteni típuskonverzió segítségével. Elsőként alakítsuk át a sztringet karaktertömbbé:

```
[10] PS C:\> $s = "FYDIBOHF23SPDLT"
[11] PS C:\> [char[]] $s
F
Y
D
I
B
O
H
F
2
3
S
P
D
L
T
```

Majd ebből csináljunk egy egésztömböt, amely a karakterkódokat tartalmazza:

```
[12] PS C:\> [int[]][char[]] $s
70
89
68
```

```
73
66
79
72
70
50
51
83
80
68
76
84
```

Majd ezt a számsort töltsük bele egy csővezetékbe és vonjunk ki minden eleméből egyet:

```
[13] PS C:\> [int[]][char[]] $s | ForEach-Object{$_ -1}
69
88
67
72
65
78
71
69
49
50
82
79
67
75
83
```

Az így megkapott számsort alakítsuk vissza karaktersorozattá:

```
[14] PS C:\> [char[]]([int[]][char[]] $s | ForEach-Object{$_ -1})
E
X
C
H
A
N
G
E
1
2
R
O
C
K
S
```

Itt már az éles szeműek felismerik a megoldást, a többieknek segítsünk azzal, hogy a karaktersorozatból sztringet gyúrunk össze:

```
[15] PS C:\> [string][char[]]([int[]][char[]] $s | ForEach-Object{$_ -1})
E X C H A N G E 1 2 R O C K S
```

Ez majdnem tökéletes, azzal a különbséggel, hogy felesleges a sok szóköz.

Megjegyzés

Alaphelyzetben a PowerShell, amikor egy karaktertömbből sztringet rak össze, akkor az összefűzött elemek közé egy szóközt tesz elválasztó karakterként. De szerencsére ez testre szabható a gyári `$ofs` változóval, ami az `Output Field Separator`. Ha ennek a változónak adunk egy üres sztring értéket (””), akkor ez pont célra vezet:

```
[16] PS C:\> $ofs=""
[17] PS C:\> [string][char[]]([int[]][char[]] $s | ForEach-Object{$_-1})
EXCHANGE12ROCKS
```

Az `$ofs` változó – ellentétben a többi, „igazi” automatikus változóval – alaphelyzetben nem létezik. Ha létrehozzuk, akkor a benne tárolt sztring lesz az elválasztó karakter.

Még a dátum típussal kapcsolatban szoktunk gyakran típuskonverziót végezni. Például szeretnék készíteni egy olyan függvényt, ami kiszámolja, hogy hány nap van még a születésnapomig:

```
[9] PS I:\>function szülinap ([string] $mikor)
>> {
>>  ([datetime] ([string]((get-date).Year) + "-$mikor") - (get-date)).Days
>> }
>>
[10] PS I:\>szülinap "10-14"
178
```

A függvény működése: sztring formátumban beírom a születési hó, nap értékét kötőjelesen (`10-14`), ehhez az aktuális dátum (`Get-Date`) sztringgé alakított évszám részét hozzátoldom, ebből az új, teljes dátumot kiadó sztringből csinálok egy dátumot `[datetime]`, ebből a dátumból kivonom az aktuális dátumot, majd ennek az egésznek veszem a napokban kifejezett értékét (`.Days`).

1.4.9 .NET típusok, statikus tagok

A következő táblázatban összefoglaltam a PowerShell által rövid névvel is hivatkozott típusokat és azok .NET Frameworkbeli típusmegnevezését:

PowerShell rövid név	.NET típusnév
[adsi]	System.DirectoryServices.DirectoryEntry
[adsisearcher]	System.DirectoryServices.DirectorySearcher
[array]	System.Array
[bool]	System.Boolean
[byte]	System.Byte
[char]	System.Char
[datetime]	System.DateTime
[decimal]	System.Decimal
[double]	System.Double

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

[float]	System.Single
[hashtable]	System.Collections.Hashtable
[int]	System.Int32
[ipaddress]	System.Net.IPAddress
[long]	System.Int64
[powershell]	System.Management.Automation.PowerShell
[psobject]	System.Management.Automation.PSObject
[ref]	System.Management.Automation.PSReference
[regex]	System.Text.RegularExpressions.Regex
[scriptblock]	System.Management.Automation.ScriptBlock
[single]	System.Single
[string]	System.String
[switch]	System.Management.Automation.SwitchParameter
[type]	System.Type
[void]	System.Void
[wmi]	System.Management.ManagementObject
[wmiclass]	System.Management.ManagementClass
[wmisearcher]	System.Management.ManagementClass
[xml]	System.Xml.XmlDocument

Ezen kívül természetesen használható más típus is a .NET Frameworkből, de akkor a típus hivatkozásánál a teljes névvel kell hivatkozni, mint ahogy ezt tettük korábban a `System.Collections.ArrayList` típusnál. Az adott típusú (vagy más szóval osztályba tartozó) objektumot, azaz annak a típusnak egy példányát a `new-object` cmdlettel tudjuk létrehozni.

Megjegyzés

Valójában a „system” előtagot nem feltétlenül kell kiírni, mert ezt a PowerShell a típus elé biggyeszti, ha az a kifejezés, amit használtunk nem található szó szerint a típuskönyvtárban:

```
[12] PS C:\> $al = New-Object collections.arraylist
[13] PS C:\> $al.gettype().fullname
System.Collections.ArrayList
```

Ezen kívül nagyon sok olyan típus van a .NET Frameworkben, amelyek estében nem csak akkor profitálhatunk az osztály metódusaiból, ha azok egy példányát, objektumát hozzuk létre, hanem maga az osztály (típus) is rendelkezik metódusokkal, tagjellemzőkkel. Például ilyen a `[math]` - matematika osztály. Ebből nem csinálunk tényleges objektumot, hanem magának a `[math]` típusnak hívjuk meg a metódusait, tulajdonságait. Ehhez speciális szintaxist kell alkalmazni:

```
[1] PS C:\> [math]::pi
3,14159265358979
```

A fenti példában a `[math]` osztály `PI` tulajdonságát olvasom ki. Ehhez a (`::`) un. static member hivatkozást kell alkalmazni.

A könyv tanfolyami felhasználása nem engedélyezett!

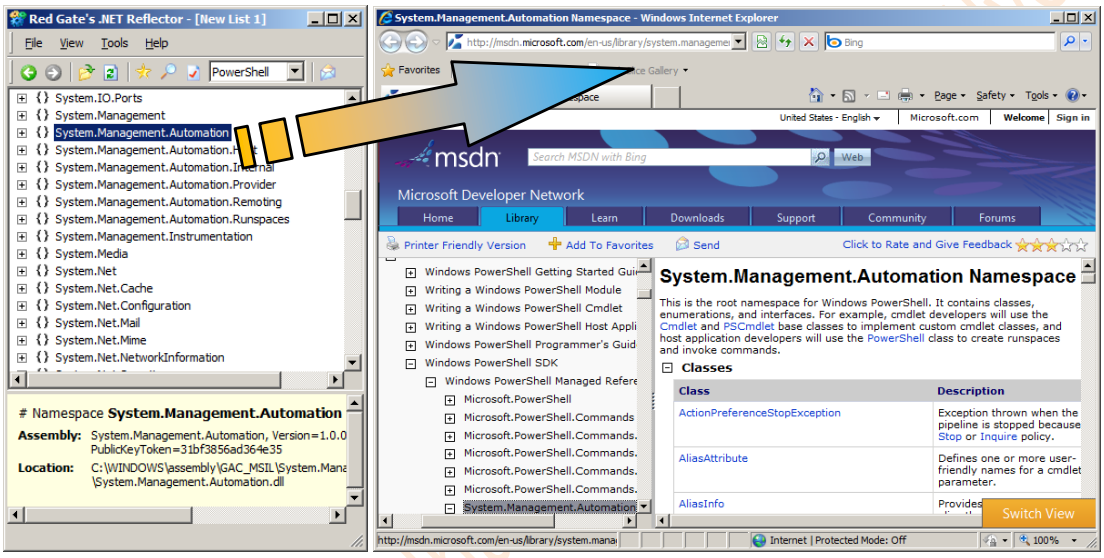
Nézzük meg, hogy hogyan lehet egy statikus metódust meghívni:

```
[2] PS C:\> [math]::Pow(3,2)
9
```

Itt a hatványozás statikus metódust hívtam meg: $3^2 = 9$.

1.4.10 A .NET osztályok felderítése

Az osztályok (típusok) felderítéséhez, a használatuk magyarázatához nagyon jól használhatjuk a Reflector programot (részletesebben a *2.2.3 Reflector* fejezetben):



25. ábra A Reflector közvetlenül megnyitja a .NET osztály magyarázatát

Látható, hogy a helyi menüben rögtön meg tudja hívni nekünk az adott osztály vagy metódus magyarázatát az MSDN weboldalról.

Természetesen a `get-member` cmdlettel is kilistázhatók a statikus tagjellemzők a `-static` kapcsolóval. Alább például a `[system.convert]` osztály `ToInt32` metódusa látszik:

```
[19] PS C:\> [system.convert] | get-member ToInt32 -MemberType methods -stat
ic | ft -wrap


   TypeName: System.Convert

Name     MemberType Definition
----     ---------- ----------
ToInt32 Method     static System.Int32 ToInt32(String value, Int32 fromBase
                   ), static System.Int32 ToInt32(Object value), static Sys
                   tem.Int32 ToInt32(Object value, IFormatProvider provider
                   ), static System.Int32 ToInt32(Boolean value), static Sy
                   stem.Int32 ToInt32(Char value), static System.Int32 ToIn
                   t32(SByte value), static System.Int32 ToInt32(Byte value
                   ), static System.Int32 ToInt32(Int16 value), static Syst
                   em.Int32 ToInt32(UInt16 value), static System.Int32 ToIn
                   t32(UInt32 value), static System.Int32 ToInt32(Int32 val
                   ue), static System.Int32 ToInt32(Int64 value), static Sy
```

A könyv tanfolyami felhasználása nem engedélyezett!

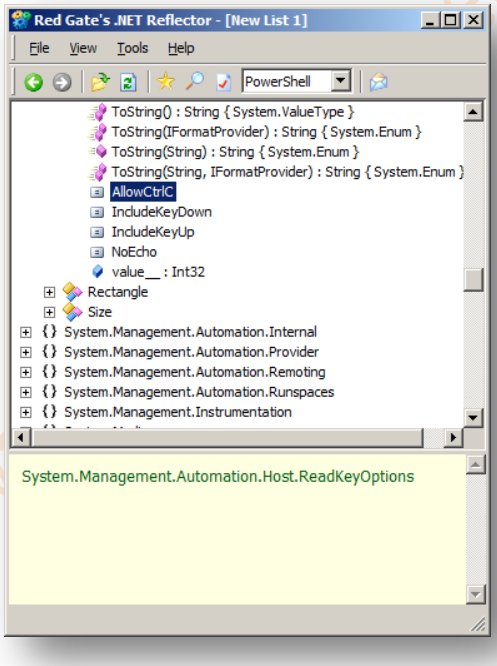
© Soós Tibor – soost@ IQJB.hu

```
                    stem.Int32 ToInt32(UInt64 value), static System.Int32 To
                    Int32(Single value), static System.Int32 ToInt32(Double
                    value), static System.Int32 ToInt32(Decimal value), stat
                    ic System.Int32 ToInt32(String value), static System.Int
                    32 ToInt32(String value, IFormatProvider provider), stat
                    ic System.Int32 ToInt32(DateTime value)
```

Ezzel a metódussal a 2-es, 8-as és 16-os számrendszer számait lehet egésszé konvertálni:

```
[14] PS C:\> [system.convert]::ToInt32("F", 16)
15
[15] PS C:\> [system.convert]::ToInt32("111111", 2)
63
[16] PS C:\> [system.convert]::ToInt32(111111, 2)
63
```

Nézzük kicsit jobban meg a Reflectorral a különböző .NET osztályokat! Keressük például meg a PowerShell osztályait, amelyeket a `System.Management.Automation` névtérben keresendők. Igazából nem gyakran kell ide járkálni, néhány típus belső világáról azonban gyűjthetünk hasznos információkat:



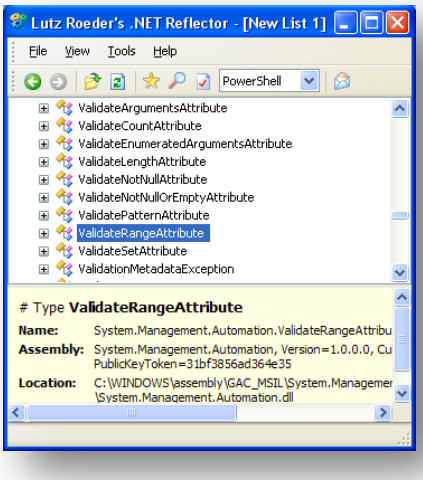
26. ábra A ReadKeyOption felderítése

A fenti példában például meg tudjuk nézni, hogy a `ReadKey` metódusnak milyen opciói vannak. A `ReadKey` metódussal a *2.1.2.3 Lépjünk kapcsolatba a konzolablakkal ($host)* fejezetben lesz részletesen szó.

Vagy a függvények paramétereinek ellenőrzését lehetővé tevő osztályokat deríthetjük fel:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



27. ábra Validálási osztályok felderítése

Ezekről részletesen a *1.7.2.9 Paraméterek, változók ellenőrzése (validálás)* fejezetben lesz szó.

Sok esetben egy frissen létrehozott objektum tagjellemzőit szeretnénk felderíteni. Például egy új `system.collections.arraylist` típusú objektumét:

```
[1] PS C:\> $o = New-Object collections.arraylist
[2] PS C:\> $o | get-member
Get-Member : No object has been specified to the get-member cmdlet.
At line:1 char:8
+ $o | gm <<<<
    + CategoryInfo          : CloseError: (:) [Get-Member], InvalidOperationE
   xception
    + FullyQualifiedErrorId : NoObjectInGetMember,Microsoft.PowerShell.Comman
   ds.GetMemberCommand
```

Ha ezt a csövezős módszert használjuk, akkor hibát kaptunk. Viszont ha az `InputObject` paramétereként adjuk át az objektumot, akkor már megkapjuk a keresett tagjellemzőket:

```
[3] PS C:\> Get-Member -InputObject $o


   TypeName: System.Collections.ArrayList

Name            MemberType            Definition
----            ----------            ----------
Add             Method                int Add(System.Object value)
AddRange        Method                System.Void AddRange(System.Collection...
BinarySearch    Method                int BinarySearch(int index, int count,...
Clear           Method                System.Void Clear()
Clone           Method                System.Object Clone()
Contains        Method                bool Contains(System.Object item)
…
```

Ivanyi Peter nem engedélyezett!

1.4.11 PowerShell objektumok vizsgálata (Format-Custom)

Az objektumok részletesebb vizsgálatára felhasználható a `format-custom` cmdlet. Ez a bemeneteként kapott objektum tulajdonságait részletesen kiírja, ezen kívül az objektum és tulajdonságainak típusát (osztályát) is megjeleníti:

```
[1] PS C:\> Get-Service wuauserv | Format-Custom -Property *

class ServiceController
{
  Name = wuauserv
  RequiredServices =
    [
      class ServiceController
      {
        Status = Running
        Name = rpcss
        DisplayName = Remote Procedure Call (RPC)
      }
    ]

  CanPauseAndContinue = False
  CanShutdown = True
  CanStop = True
  DisplayName = Windows Update
  DependentServices =
    [
    ]

  MachineName = .
  ServiceName = wuauserv
  ServicesDependedOn =
    [
      class ServiceController
      {
        Status = Running
        Name = rpcss
        DisplayName = Remote Procedure Call (RPC)
      }
    ]

  ServiceHandle =
    class SafeServiceHandle
    {
      IsInvalid = False
      IsClosed = False
    }
  Status = Running
  ServiceType = Win32ShareProcess
  Site =
  Container =
}
```

Óvatosan bánjunk ezzel a megjelenítési lehetőséggel, hiszen ha nagyon összetett adattípust iratunk így ki, a kimenet nagyon szószátyár is lehet. Óvatosságból használjuk a `Depth` paramétert:

```
[10] PS C:\> $f = Get-Item C:\munka\futók.txt
[11] PS C:\> $f | Format-Custom -Depth 1
```

```
class FileInfo
{
  LastWriteTime =
    class DateTime
    {
      Date = 2009. 11. 21. 0:00:00
      Day = 21
      DayOfWeek = Saturday
      DayOfYear = 325
      Hour = 19
      Kind = Local
      Millisecond = 815
      Minute = 45
      Month = 11
      Second = 20
      Ticks = 633944295208158750
      TimeOfDay = 19:45:20.8158750
      Year = 2009
      DateTime = 2009. november 21. 19:45:20
    }
  Length = 254
  Name = futók.txt
}
```

Enélkül a fenti példában minden egyes `datetime` és `timespan` tulajdonság külön kibontásra került volna.

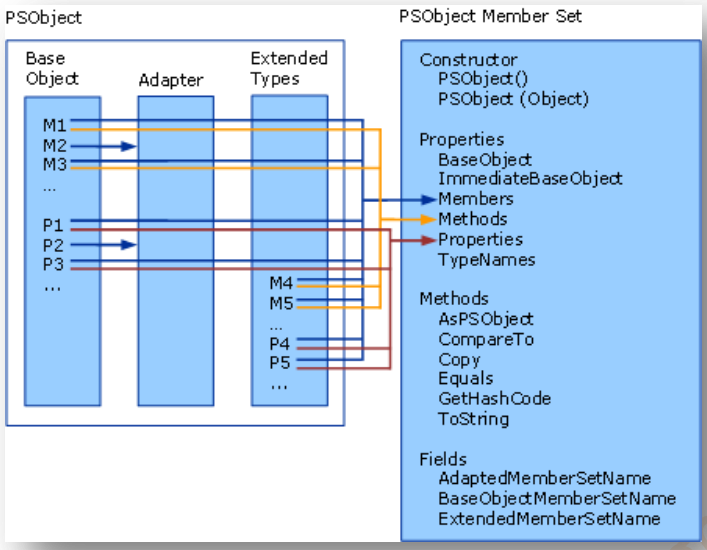
1.4.12 Objektumok testre szabása, kiegészítése

Elöljáróban annyit meg kell jegyezni, hogy természetesen a PowerShell parancssori értelmezőjétől nem várhatjuk el, hogy magát a .NET osztályokat módosítsa. A PowerShell csak a saját „*generic*” osztályát, a `PSObject` osztályt és annak példányait képes módosítani.

Vegyünk egy általános objektumot, hívjuk őt „Base Object”-nek. A PowerShell elképzelhető, hogy nem minden metódust és tulajdonságot tesz elérhetővé, ezeket egy adapter réteggel elfedi. Ha bármilyen módosítást, kiegészítést teszünk ehhez az objektumhoz, például definiálunk mi magunk valamilyen tulajdonságot vagy metódust hozzá, akkor azt egy újabb, „Extended Types” rétegben tesszük meg. Így alakul ki a végleges tulajdonság- és metóduslistánk, amit az alábbi ábra illusztrál:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



28. ábra PowerShell adaptált objektummodellje
(forrás: http://msdn2.microsoft.com/en-us/library/cc136098(VS.85).aspx)

Az így elérhető objektum kettős jelleggel bír: egyrészt hordozza magában az eredeti „Base Object” jellegét, de valójában ez már egy új, `PSObject` típusú objektum lesz.

Ennek megértéséhez nézzük meg az alábbi példát, melyben az `Add-Member` cmdlettel bővítek ki egy objektumot újabb tulajdonsággal, vagy egyéb tagjellemzővel. Ilyen tagjellemzők lehetnek:

Tagjellemző típus	Leírás
AliasProperty	Álnév egy már meglevő tulajdonságra.
All	Minden lehetséges taglehetőség típus.
CodeMethod	Olyan metódus, amellyel hivatkozunk egy .NET osztály statikus metódusára.
CodeProperty	Olyan tulajdonság, amellyel hivatkozunk egy .NET osztály statikus tulajdonságára.
MemberSet	Tulajdonságok és metódusok halmaza egy közös néven.
Method	A PSObject alapjaként szolgáló objektumosztály egyik metódusa.
Methods	Minden metódus.
NoteProperty	Tulajdonságnév és tulajdonságérték páros, nem számított, hanem fix tag.
ParameterizedProperty	Olyan tulajdonság, ami paraméterezhető.
Properties	Minden tulajdonság.
Property	A PSObject alapjaként szolgáló objektumosztály egyik tulajdonsága.
PropertySet	Tulajdonsághalmaz.
ScriptMethod	Szkripttel megfogalmazott metódus.
ScriptProperty	Szkripttel megfogalmazott tulajdonság.

Ezek közül leggyakrabban `NoteProperty`, `ScriptProperty`, `ScriptMethod` testre szabási lehetőséget alkalmazzuk.

Akkor nézzük a példát:

```
[30] PS C:\> $a = 23
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[31] PS C:\> $a.GetType().FullName
System.Int32
[32] PS C:\> $a -is [PSObject]
False
[33] PS C:\> $a | add-member -memberType Scriptproperty -Name Dupla -value
{$this*2}
[34] PS C:\> $a.dupla
[35] PS C:\> $a -is [PSObject]
True
[36] PS C:\> $a| Get-Member -MemberType scriptproperty
[37] PS C:\> $a
23
[38] PS C:\> $a.dupla
[39] PS C:\>
```

Nézzük mi történt: a [30]-ban van egy egyszerű `Int32` változóm, [32]-ben rákérdezek, hogy vajon nem `PSObject`-e a szegény? Válasz az, hogy nem.

[33]-ban hozzáadok egy szkript alapján kiszámolódó tulajdonságot `Dupla` néven. Az érték (`value`) úgy generálódik, hogy az adott objektum értékét (`$this` változó tartalmazza a szkript számára majd ezt futási időben) megszorzom 2-vel.

Ezután ki is próbálom, lekérem a [34]-ben a dupla tulajdonságot, de nem kapok vissza semmit. A [35]-ben meg is vizsgálom, és hiába lett a `$a` változóm már `PSObject`, az `add-member` a csővezeték végén nem hat vissza az `$a` változóra. Szegény PowerShell annyit csinált csak, hogy érzékelte, hogy itt az `$a` tulajdonságainak bővítése történik, ezért az automatikus típuskonverzió miatt `[PSObject]`-té tette, de ténylegesen az új tulajdonságot nem integráltuk bele az `$a`-ba. Ezért [36]-ban hiába kérdezem le, hogy mi az `$a` `scriptproperty`-je, nem kapok választ. És természetesen ugyanilyen némaság a válasz a nem létező tulajdonság lekérdezésére is a [38]-ban.

Nézzük akkor meg, hogy hogyan lesz jó:

```
[40] PS C:\> $a = add-member -inputobject $a -memberType Scriptproperty -Name
Dupla -value {$this*2} -PassThru
[41] PS C:\> $a.dupla
46
[42] PS C:\> $a| Get-Member -MemberType scriptproperty


   TypeName: System.Int32

Name  MemberType     Definition
----  ----------     ----------
Dupla ScriptProperty System.Object Dupla {get=$this*2;}


[43] PS C:\> $a -is [PSObject]
True
```

[40]-ben már helyesen alkalmazom az `add-member` cmdletet. Értékadással kombinálom, `$a` vegye fel az `add-member` kimenetén kijövő értéket. De baj van! Az `add-member`-nek alaphelyzetben nincs kimenete! Ezért találták ki a `-PassThru` kapcsolót, hogy mégis legyen. Az esetek zömében ugyanis nem az alap .NET osztályok objektumait szoktuk testre szabni, hanem `[PSObject]` osztály objektumait, ott meg nem kell semmit visszaadnia az `add-member`-nek, hiszen „helyben” hozzá tudja adni a tulajdonságot, metódust. Itt viszont az új, kibővített `PSObject` objektumot vissza kell tölteni az eredeti változóba.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

[41]-ben már jól működik a `Dupla` tulajdonság, a [42]-ben meg azt látjuk, hogy ott a `ScriptProperty`-nk, és annak ellenére, hogy látjuk `$a` egy `TypeName: System.Int32`, emellett a [42]-ben azt is látjuk még `[PSObject]` is!

Nézzünk egy olyan példát is, ahol nincs szükség ennyi trükközésre, hanem egy már eleve `PSObject` objektumot módosítok. Először tehát érdemes mindig megvizsgálni, hogy egy adott objektum már eleve `PSObject`-e vagy sem, hiszen ha nem az, akkor másként kell eljárni, mintha igen.

Vegyünk egy fájlt:

```
[2] PS C:\> $f = Get-Item C:\filemembers.txt
[3] PS C:\> $f


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.03.01.    21:09           10 filemembers.txt


[4] PS C:\> $f -is [PSObject]
True
[5] PS C:\> $f.gettype().FullName
System.IO.FileInfo
```

Látszik, hogy amellett, hogy ez egy `PSObject`, mellette még `System.IO.FileInfo` típusú objektum. Így erre már közvetlenül alkalmazhatjuk az `add-member` cmdletet:

```
[6] PS C:\> $f | Add-Member -MemberType scriptproperty -Name Típus -Value
{if($this -is "System.IO.FileInfo"){"File"} else{"Directory"}}
[7] PS C:\> $f.Típus
File
```

A hiányossága az `add-member`-rel kibővített objektumoknak, hogy a bővítményüket „elfelejtik” mihelyst új értéket adunk nekik:

```
[11] PS C:\> $f = get-item c:\old
[12] PS C:\> $f.Típus
[13] PS C:\> $f | Add-Member -MemberType scriptproperty -Name Típus -Value
{if($this -is "System.IO.FileInfo"){"File"} else{"Directory"}}
[14] PS C:\> $f.Típus
Directory
```

A [12]-es promptban hiába kérem le újra a `Típus` tulajdonságot, nem kapok választ, csak ha újra hozzáadom ezt a tagjellemzőt.

Erre megoldást jelent magának az osztálynak (típusnak) a testre szabása, amit a következő fejezetben mutatok be.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.4.13 Osztályok (típusok) testre szabása

Az előző fejezetben láttuk, hogy egy osztály egy konkrét objektumpéldányának hogyan adhatunk újabb tagjellemzőket. Ez nagyon jó lehetőség, csak az a baja – mint ahogy láttuk is – hogy minden újabb objektum példánynál újra létre kell hozni saját tagjellemzőinket. Milyen jó lenne, ha magát az osztályt (típust) tudnánk módosítani és akkor az adott osztály minden objektuma már eleve rendelkezne az általunk definiált tagjellemzővel. Szerencsére ezt is lehetővé teszi a PowerShell!

Az objektumtípusok a PowerShell számára egy `types.ps1xml` fájlban vannak definiálva a `C:\WINDOWS\system32\windowspowershell\v1.0` könyvtárban. Azt senki sem ajánlja, hogy ezt átszerkesszük, de hasonló fájlokat mi is készíthetünk, amelyekkel mindenféle dolgot tehetünk az objektumtípusokhoz: új tulajdonságokat, metódusokat és még azt is, hogy mely tulajdonságait mutassa meg magáról az objektum alapban. Ez utóbbi is nagyon fontos, mert engem zavart, hogy például a `get-services` cmdlet alapban miért pont a `Status`, `Name` és `DisplayName` tulajdonságokat adja ki? Hiszen van neki jó néhány egyéb tulajdonsága is:

```
PS C:\Documents and Settings\SoosTibi> Get-Service

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
Running  ALG                Application Layer Gateway Service
Running  AppMgmt            Application Management
Stopped  aspnet_state       ASP.NET State Service
...
```

Erre válasz ez az előbb említett types.ps1xml file. Keressük meg benne a szolgáltatások `System.ServiceProcess.ServiceController` adattípusát:

```
...
    <Type>
        <Name>System.ServiceProcess.ServiceController</Name>
        <Members>
            <MemberSet>
                <Name>PSStandardMembers</Name>
                <Members>
                    <PropertySet>
                        <Name>DefaultDisplayPropertySet</Name>
                        <ReferencedProperties>
                            <Name>Status</Name>
                            <Name>Name</Name>
                            <Name>DisplayName</Name>
                        </ReferencedProperties>
                    </PropertySet>
                </Members>
            </MemberSet>
            <AliasProperty>
                <Name>Name</Name>
                <ReferencedMemberName>ServiceName</ReferencedMemberName>
            </AliasProperty>
        </Members>
    </Type>
...
```

A kiemelésben látszik, hogy azért ezeket a tulajdonságokat mutatja meg alapban a `get-service`, mert ezek vannak `DefaultDisplayPropertySet`-ként definiálva.

Na, de minket most nem ez érdekel, hanem hogy hogyan tudok típust módosítani.

Létrehoztam egy `typemember.ps1xml` fájlt:

```
<Types>
        <Type>
                <Name>System.IO.FileInfo</Name>
                <Members>
                        <NoteProperty>
                                <Name>Tipus</Name>
                                <Value>
                                        File
                                </Value>
                        </NoteProperty>
                </Members>
        </Type>
        <Type>
                <Name>System.IO.DirectoryInfo</Name>
                <Members>
                        <NoteProperty>
                                <Name>Tipus</Name>
                                <Value>
                                        Directory
                                </Value>
                        </NoteProperty>
                </Members>
        </Type>
</Types>
```

A szerkezet magáért beszél. Definiáltam két különböző típusban is egy-egy `NoteProperty` tulajdonságot, hiszen itt magából a típusból következik, hogy fájlról vagy könyvtárról van szó, így nem kell futási időben kiszámolni semmit sem.

Most már csak be kell etetni a rendszerbe az én típusmódosításomat és már nézhetjük is az eredményt:

```
[1] PS C:\> Update-TypeData C:\powershell2\Tananyag\typemember.ps1xml
[2] PS C:\> $f = Get-Item C:\filemembers.txt
[3] PS C:\> $f.tipus
File
[4] PS C:\> $f = Get-Item C:\old
[5] PS C:\> $f.tipus
Directory
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzések:

A ps1xml fájlok a PowerShellben olyan elbírálás alá esnek, mint a szkriptek, azaz a rendszerbe való felvételükhöz a *1.8.1 Szkriptek engedélyezése és indítása* fejezetben leírt engedélyezésre van szükség.
Másik fontos tulajdonsága ezeknek a fájloknak, hogy kis-nagybetű érzékenyek, azaz az XML címkéknél és a .NET osztályokra való hivatkozásnál fontos, hogy a kis- és nagybetűket helyesen írjuk.
Továbbá ebben az XML fájlban nem használhatunk ékezetes karaktereket, így a definiált tulajdonság nem `Típus`, hanem `Tipus` lett.

Természetesen ez csak egy kis ízelítő volt az osztályok, típusok testre szabásából, a gyakorlati részben visszatérek majd erre gyakorlatiasabb példákkal.

1.4.14 PSBase, PSAdapted, PSExtended, PSObject nézetek

Mint ahogy az fejezet elején bemutattam, a PowerShell igazából a .NET osztályokat nem közvetlenül kezeli, hanem néha kicsit átalakítja annak érdekében, hogy még egységesebb, egyszerűbb, bizonyos esetekben biztonságosabb legyen ezen objektumok kezelése.

Különböző nézetek segítségével mi is láthatjuk azt, hogy milyen „csalafintaságokat” követett el ezeken az osztályokon a PowerShell:

Nézet neve	Nézet tartalma
PSBASE	A .NET-es osztály eredeti állapotban
PSADAPTED	A PowerShell által adaptált nézet (ezt látjuk alaphelyzetben)
PSEXTENDED	Csak a kibővített tagok
PSOBJECT	Magának az adapternek a nézete

Nézzünk ezekre néhány példát. Elsőként az XML adattípust mutatom, mert ott elég jól láthatóak ezen nézetek közti különbségek. Nézzük meg egy XML adat tagjellemzőit:

```
[20] PS C:\> $x = [xml] "<elem>érték<szint1><szint2>mélyadat</szint2></szint
1></elem>"
[21] PS C:\> $x | Get-Member


   TypeName: System.Xml.XmlDocument

Name                        MemberType            Definition
----                        ----------            ----------
ToString                    CodeMethod            static System.String X...
add_NodeChanged             Method                System.Void add_NodeCh...
add_NodeChanging            Method                System.Void add_NodeCh...
...
Validate                    Method                System.Void Validate(V...
WriteContentTo              Method                System.Void WriteConte...
WriteTo                     Method                System.Void WriteTo(Xm...
Item                        ParameterizedProperty System.Xml.XmlElement ...
elem                        Property              System.Xml.XmlElement ...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nagyon sok jellemzője van, az egyszerűbb áttekinthetőség miatt kicsit megvágtam a közepén. A legutolsó jellemző egy Property típusú, `elem` nevű tag. Hát ilyet biztos nem tettek bele a .NET keretrendszerbe. Erről meg is győződhetünk:

```
[22] PS C:\> $x.psbase | Get-Member


   TypeName: System.Management.Automation.PSMemberSet

Name                        MemberType            Definition
----                        ----------            ----------
add_NodeChanged             Method                System.Void add_NodeCh...
add_NodeChanging            Method                System.Void add_NodeCh...
...
ChildNodes                  Property              System.Xml.XmlNodeList...
DocumentElement             Property              System.Xml.XmlElement ...
DocumentType                Property              System.Xml.XmlDocument...
FirstChild                  Property              System.Xml.XmlNode Fir...
HasChildNodes               Property              System.Boolean HasChil...
...
Value                       Property              System.String Value {g...
XmlResolver                 Property              System.Xml.XmlResolver...
```

A fenti listában tényleg nincs `elem` nevű tulajdonság. Miért tették bele ezt az elem tulajdonságot vajon a PowerShell alkotói? Azért, hogy egyszerűen lehessen hivatkozni az XML adathalmaz különböző elemeire, hiszen az XML egy hierarchikus felépítésű adattípus, így könnyen adódik az ötlet az ilyen jellegű hivatkozási lehetőségre:

```
[23] PS C:\> $x.elem

#text                                szint1
-----                                ------
érték                                szint1


[24] PS C:\> $x.elem.szint1

szint2
------
mélyadat


[25] PS C:\> $x.elem.szint1.szint2
mélyadat
```

Nézzük, hogy a PS1XML fájlban történt-e típusbővítés az XML típus esetében?

```
[53] PS C:\> $x.psextended | Get-Member


   TypeName: System.Management.Automation.PSMemberSet

Name     MemberType Definition
----     ---------- ----------
ToString CodeMethod static System.String XmlNode(PSObject instance)
```

Egyetlen egy `CodeMethod` lett csak definiálva, amellyel sztringgé alakíthatjuk az XML adatot.
Nézzük meg az általam létrehozott `Tipus` tulajdonságot a fájl és könyvtár objektumoknál:

```
[27] PS C:\> $fo = Get-Item C:\powershell2\demo\demo1.ps1
[28] PS C:\> $fo.Tipus
File
[29] PS C:\> $fo.psextended | Get-Member


   TypeName: System.Management.Automation.PSMemberSet

Name           MemberType     Definition
----           ----------     ----------
PSChildName    NoteProperty   System.String PSChildName=demo1.ps1
PSDrive        NoteProperty   System.Management.Automation.PSDriveInfo PS...
PSIsContainer  NoteProperty   System.Boolean PSIsContainer=False
PSParentPath   NoteProperty   System.String PSParentPath=Microsoft.PowerS...
PSPath         NoteProperty   System.String PSPath=Microsoft.PowerShell.C...
PSProvider     NoteProperty   System.Management.Automation.ProviderInfo P...
Tipus          NoteProperty   System.String Tipus=File
BaseName       ScriptProperty System.Object BaseName {get=[System.IO.Path...
Mode           ScriptProperty System.Object Mode {get=$catr = "";...
ReparsePoint   ScriptProperty System.Object ReparsePoint {get=if($this.At...
```

Ott látható a listában az általam, a PS1XML fájlon keresztül történt típusbővítésnek a nyoma.
Nézzünk még egy példát a `PSBase` nézet használatára:

```
[30] PS C:\> PS C:\> $user = [ADSI] "WinNT://$env:computername/$env:username"
[31] PS C:\> $user.name
tibi
[32] PS C:\> $user | Get-Member


   TypeName: System.DirectoryServices.DirectoryEntry

Name                          MemberType Definition
----                          ---------- ----------
ConvertDNWithBinaryToString   CodeMethod static string ConvertDNWithBinaryToS...
ConvertLargeIntegerToInt64    CodeMethod static long ConvertLargeIntegerToInt...
AutoUnlockInterval            Property   System.DirectoryServices.PropertyVal...
BadPasswordAttempts           Property   System.DirectoryServices.PropertyVal...
Description                   Property   System.DirectoryServices.PropertyVal...
FullName                      Property   System.DirectoryServices.PropertyVal...
HomeDirDrive                  Property   System.DirectoryServices.PropertyVal...
HomeDirectory                 Property   System.DirectoryServices.PropertyVal...
LastLogin                     Property   System.DirectoryServices.PropertyVal...
LockoutObservationInterval    Property   System.DirectoryServices.PropertyVal...
LoginHours                    Property   System.DirectoryServices.PropertyVal...
LoginScript                   Property   System.DirectoryServices.PropertyVal...
MaxBadPasswordsAllowed        Property   System.DirectoryServices.PropertyVal...
MaxPasswordAge                Property   System.DirectoryServices.PropertyVal...
MaxStorage                    Property   System.DirectoryServices.PropertyVal...
MinPasswordAge                Property   System.DirectoryServices.PropertyVal...
MinPasswordLength             Property   System.DirectoryServices.PropertyVal...
Name                          Property   System.DirectoryServices.PropertyVal...
objectSid                     Property   System.DirectoryServices.PropertyVal...
Parameters                    Property   System.DirectoryServices.PropertyVal...
PasswordAge                   Property   System.DirectoryServices.PropertyVal...
PasswordExpired               Property   System.DirectoryServices.PropertyVal...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PasswordHistoryLength       Property   System.DirectoryServices.PropertyVal...
PrimaryGroupID              Property   System.DirectoryServices.PropertyVal...
Profile                     Property   System.DirectoryServices.PropertyVal...
UserFlags                   Property   System.DirectoryServices.PropertyVal...
```

Egy helyi felhasználót betöltöttem a `$user` nevű változómba, szépen le is tudtam kérdezni a nevét. Majd amikor kilistázom a felhasználóm tagjellemzőit, meglepődve láthatjuk, hogy nincs köztük igazi, praktikus metódus sem! Márpedig nehezen hihető el, hogy tényleg semmi értelmeset nem tud egy felhasználói fiók magával kezdeni. Nézzünk az objektumunk mögé:

```
[34] PS C:\> $user.psbase | Get-Member


   TypeName: System.Management.Automation.PSMemberSet

Name                      MemberType Definition
----                      ---------- ----------
Disposed                  Event      System.EventHandler Disposed(System.Ob...
Close                     Method     System.Void Close()
CommitChanges             Method     System.Void CommitChanges()
CopyTo                    Method     adsi CopyTo(adsi newParent), adsi Copy...
CreateObjRef              Method     System.Runtime.Remoting.ObjRef CreateO...
DeleteTree                Method     System.Void DeleteTree()
Dispose                   Method     System.Void Dispose()
Equals                    Method     bool Equals(System.Object obj)
GetHashCode               Method     int GetHashCode()
GetLifetimeService        Method     System.Object GetLifetimeService()
GetType                   Method     type GetType()
InitializeLifetimeService Method     System.Object InitializeLifetimeService()
Invoke                    Method     System.Object Invoke(string methodName...
InvokeGet                 Method     System.Object InvokeGet(string propert...
InvokeSet                 Method     System.Void InvokeSet(string propertyN...
MoveTo                    Method     System.Void MoveTo(adsi newParent), Sy...
RefreshCache              Method     System.Void RefreshCache(), System.Voi...
Rename                    Method     System.Void Rename(string newName)
ToString                  Method     string ToString()
AuthenticationType        Property   System.DirectoryServices.Authenticatio...
Children                  Property   System.DirectoryServices.DirectoryEntr...
Container                 Property   System.ComponentModel.IContainer Conta...
Guid                      Property   System.Guid Guid {get;}
Name                      Property   System.String Name {get;}
NativeGuid                Property   System.String NativeGuid {get;}
NativeObject              Property   System.Object NativeObject {get;}
ObjectSecurity            Property   System.DirectoryServices.ActiveDirecto...
Options                   Property   System.DirectoryServices.DirectoryEntr...
Parent                    Property   System.DirectoryServices.DirectoryEntr...
Password                  Property   System.String Password {set;}
Path                      Property   System.String Path {get;set;}
Properties                Property   System.DirectoryServices.PropertyColle...
SchemaClassName           Property   System.String SchemaClassName {get;}
SchemaEntry               Property   System.DirectoryServices.DirectoryEntr...
Site                      Property   System.ComponentModel.ISite Site {get;...
UsePropertyCache          Property   System.Boolean UsePropertyCache {get;s...
Username                  Property   System.String Username {get;set;}
```

Hoppá! Mindjárt más a helyzet. A metódusok zöme természetesen tartományi környezetben használható, de például a `rename()` vagy a `set_password()` metódus helyi gépen is praktikus szolgáltatás.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Vajon ezek a metódusok miért nincsenek alaphelyzetben adaptálva a PowerShell környezetre? Erre az igazi választ nem tudom, valószínű ez egy biztonsági megfontolás volt, hogy a szkriptelők ne írogassanak felelőtlenül olyan szkripteket, amelyekkel a felhasználói objektumokat módosítanak. Vagy az is lehet a magyarázat, hogy egy másik interfészt szánt volna igazából a Microsoft a felhasználó menedzsment céljaira, amin keresztül módosítani lehetett volna, de ez a PowerShell aktuális verziójába már nem fért bele.

Megjegyzése

A PowerShell 2.0-ban már a `Get-Member` is tud az objektumok mögé látni, azaz a [34]-es sor eredményét így is megkaphatjuk:

```
PS C:\> $user | Get-Member -View base
```

Hasonlóan megkaphatjuk a további nézeteket a `-View` paraméterhez megadott `Extended`, `Adapted` vagy `All` szavakkal.

Utoljára nézzük a `PSObject` nézetet. Vegyünk például megint egy fájlt, és nézzük meg tulajdonságait:

```
[1] PS C:\> $f = Get-Item C:\munka\a.txt
[2] PS C:\> $f


    Directory: C:\munka


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---     2010. 01. 16.     10:59        402 a.txt


[3] PS C:\> $f.psobject


Members              : {System.String PSPath=Microsoft.PowerShell.Core\FileSyst
                       em::C:\munka\a.txt, System.String PSParentPath=Microsoft
                       .PowerShell.Core\FileSystem::C:\munka, System.String PSC
                       hildName=a.txt, System.Management.Automation.PSDriveInfo
                        PSDrive=C...}
Properties           : {System.String PSPath=Microsoft.PowerShell.Core\FileSyst
                       em::C:\munka\a.txt, System.String PSParentPath=Microsoft
                       .PowerShell.Core\FileSystem::C:\munka, System.String PSC
                       hildName=a.txt, System.Management.Automation.PSDriveInfo
                        PSDrive=C...}
Methods              : {string get_Name(), long get_Length(), string get_Direct
                       oryName(), System.IO.DirectoryInfo get_Directory()...}
ImmediateBaseObject  : C:\munka\a.txt
BaseObject           : C:\munka\a.txt
TypeNames            : {System.IO.FileInfo, System.IO.FileSystemInfo, System.Ma
                       rshalByRefObject, System.Object}
```

Látható a [2]-es sorban, hogy az `$f` változóban tényleg a fájl van, míg a [3]-as sorban a `PSObject` nézete ennek az objektumnak már nem a fájl jelleget adja vissza, hanem általában egy PowerShellbeli objektum

jellegzetességeit, azaz hogy az objektumnak vannak tagjellemzői, tulajdonságai, metódusai. A `TypeNames` tulajdonságból kiolvasható, hogy az objektum milyen osztályok leszármazottja.

1.4.15 Új típusok létrehozása (Add-Type)

Bizonyos esetekben szükség lehet, hogy teljesen új típust hozzunk létre. Egyszerűbb esetben elég, ha ez nem is „igazi” új típus, hanem a `[PSObject]` típus nekünk tetsző tulajdonságokkal kiegészített változata. Ezt a korábban látott `Add-Member` cmdlettel is megtehetjük, vagy a PowerShell 2.0 új lehetőségével, amikor a `New-Object` cmdlet `-Property` paraméterének adunk át egy olyan hashtáblát, amelyben a tulajdonságnév-érték párok vannak, mint ahogy a következő példa mutatja:

```
$p = @{
    név = "Soós Tibor"
    életkor = 39
    felvét = (get-date)
}
$user = New-Object -TypeName PSObject -Property $p
```

Nézzük, hogy mi lett ennek az eredménye:

```
[76] PS C:\munka> $user

                   életkor név                          felvét
                   ------- ---                          ------
                        39 Soós Tibor                   2010. 01. 10. 23:13:07


[77] PS C:\munka> $user | gm


   TypeName: System.Management.Automation.PSCustomObject

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       bool Equals(System.Object obj)
GetHashCode Method       int GetHashCode()
GetType     Method       type GetType()
ToString    Method       string ToString()
felvét      NoteProperty System.DateTime felvét=2010. 01. 10. 23:13:07
név         NoteProperty System.String név=Soós Tibor
életkor     NoteProperty System.Int32 életkor=39
```

Látható, hogy bár az általam megadott objektumtípus `PSObject` volt, de ezzel a paraméterezéssel a `new-object PSCustomObject` típussá alakította ezt, és ehhez `NoteProperty` típusú tagjellemzőket vett fel.

Ennél „profibb” objektumtípusokat is létre tudunk hozni az `Add-Type` cmdlet segítségével. Az alábbiakban látható egy többsoros sztringként egy C# kódú osztálydefiníció:

```
$code = @"
public class email {
    public string alias {get; set;}
    public string domain {get; set;}
    public email (string a, string d)
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    {
        alias = a; domain = d;
    }
    public string fullemail
        {
            get { return alias + "@" + domain; }
        }
    public static string create (string alias, string domain)
        {
            return alias + "@" + domain;
        }
}
"@
```

A fenti kód egy `email` adattípust hoz létre, egy publikus `alias` és `domain` adatmezővel, melyeket lekérdezni és beállítani is lehet. Van még egy `fullemail` tulajdonsága is, melyet csak lekérdezni lehet, hiszen ez az `alias` és a `domain` tulajdonságból generálódik. Van még egy konstruktora is `email` néven, melynek mind az `alias`, mind a `domain` paramétert meg kell adni. Van még egy `create` statikus metódusa is, amellyel email objektum létrehozása nélkül tudunk e-mail címet létrehozni. Nézzük, hogy hogyan tudunk ebből PowerShellben is használható típust generálni?

```
Add-Type -TypeDefinition $code -Language CSharpVersion3
```

Ezzel létre is jött a saját típusom. Most létre is hozok ebből egy objektumot, amit le is kérdezek:

```
[93] PS C:\munka> $address = New-Object email soost, iqjb.hu
[94] PS C:\munka> $address

alias                    domain                   fullemail
-----                    ------                   ---------
soost                    iqjb.hu                  soost@iqjb.hu
```

Nézzük meg ennek az objektumnak a tulajdonságait is:

```
[95] PS C:\munka> $address | gm


   TypeName: email

Name        MemberType Definition
----        ---------- ----------
Equals      Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
ToString    Method     string ToString()
alias       Property   System.String alias {get;set;}
domain      Property   System.String domain {get;set;}
fullemail   Property   System.String fullemail {get;}
```

Látható, hogy ennek tagjellemzői tényleg úgy néznek ki mint az „igazi” .NET-es típusoké. Az általam létrehozott tulajdonságok mellett az objektum mivoltából következő alap metódusok is megtalálhatók.

Ha megnézzük a statikus tagjellemzőket is, akkor láthatjuk a `create` metódusunkat is:

```
[96] PS C:\munka> [email] | Get-Member -Static
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
   TypeName: email

Name            MemberType Definition
----            ---------- ----------
create          Method     static string create(string alias, string domain)
Equals          Method     static bool Equals(System.Object objA, System.Ob...
ReferenceEquals Method     static bool ReferenceEquals(System.Object objA, ...
```

Ez a lehetőség igazán azok számára jelent nagy könnyebbséget, akik amúgy is rendelkeznek valamely .NET programnyelvben létrehozott osztálydefiníciókkal, így azokat nem csak a Visual Studio projektjeikben tudják felhasználni, hanem közvetlenül PowerShellből is.

Megjegyzés

Van egy „nem hivatalos”, de praktikus módja egyedi objektumok létrehozásának:

```
[46] PS C:\> $ember=""  | Select-Object -Property Név, Életkor, Lábméret
[47] PS C:\> $ember.név="Soós Tibor"
[48] PS C:\> $ember.Életkor = 39
[49] PS C:\> $ember.Lábméret = 38
[50] PS C:\> $ember

Név                                             Életkor                 Lábméret
---                                             -------                 --------
Soós Tibor                                           39                       38


[51] PS C:\> $ember | Get-Member


   TypeName: Selected.System.String

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       bool Equals(System.Object obj)
GetHashCode Method       int GetHashCode()
GetType     Method       type GetType()
ToString    Method       string ToString()
Lábméret    NoteProperty System.Int32 Lábméret=38
Név         NoteProperty System.String Név=Soós Tibor
Életkor     NoteProperty System.Int32 Életkor=39
```

Definiálok egy bármilyen, nem `$null`-t tartalmazó változót, és továbbtolom a `Select-Object` cmdletnek, és kiválasztok olyan tulajdonságokat, amelyek nincsenek is az eredeti objektumban. Ezután ez az objektum „`Selected`” altípusa lesz az eredeti objektumtípusnak, jelen esetben `Selected.System.String` típus lett. Ez el is veszítette az eredeti tulajdonságait, helyette az általam kiválasztottakkal rendelkezik, melyeknek értéket is lehet adni.

Másik, még gyakran előforduló igény, hogy egy változó csak bizonyos értékeket vehessen fel, például csak a hét napjai kerülhessenek bele. Ha ez a változó egy függvény paramétere, akkor erre majd látni fogjuk a

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

validációs attribútumok lehetőségét a *2.4.2 Paraméterek ellenőrzése* fejezetben. De van más lehetőségünk is, ez pedig a .NET háttérnek köszönhető, azon belül is az `enum` típusnak:

```
[1] PS C:\> $napok = "hétfő, kedd, szerda, csütörtök, péntek, szombat, vasárnap
"
[2] PS C:\> Add-Type -TypeDefinition "public enum Napok {$napok}"
```

A fenti két sorral létrehoztam egy új típust, ami Napok névre hallgat, és amely csak a hét napjait veheti fel értéknek.

Megjegyzés

Fontos, hogy itt a „public” és az „enum” szavak kisbetűsek! Ez nem a PowerShellnek köszönhető, hanem a C# nyelvnek, amit itt tulajdonképpen használtunk.

Hogyan tudjuk ezt használni? Nézzünk erre példákat:

```
[8] PS C:\> $nap = [napok] "hétfő"
[9] PS C:\> $nap
hétfő
[10] PS C:\> $nap | Get-Member


   TypeName: Napok

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object target)
Equals      Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode()
ToString    Method     string ToString(), string ToString(string format, Sy...
value__     Property   System.Int32 value__ {get;set;}


[11] PS C:\> $másiknap = [napok] "blabla"
Cannot convert value "blabla" to type "Napok" due to invalid enumeration value
s. Specify one of the following enumeration values and try again. The possible
 enumeration values are "hétfő, kedd, szerda, csütörtök, péntek, szombat, vasá
rnap".
At line:1 char:20
+ $másiknap = [napok] <<<<  "blabla"
    + CategoryInfo          : NotSpecified: (:) [], RuntimeException
    + FullyQualifiedErrorId : RuntimeException
```

A [8]-as sorban a `$nap` változónak adtam egy értéket a lehetséges napok közül. Ez hiba nélkül lement, még a tagjellemzők is látszanak a [10]-es sor kimeneteként.

Ezzel szemben, ha a `$másiknap` változónak olyan napot próbálok adni, ami nincs a listában, akkor hibajelzést kapunk.

Természetesen ezzel a `[napok]` típussal már valójában nem sztringgel dolgozunk, így a sztringekre jellemző metódusokról és tulajdonságokról is le kell mondanunk. (Valamit valamiért.) Viszont könnyen lehet sztringgé konvertálni:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[14] PS C:\> ([string] $nap).length
5
[15] PS C:\> ([string] $nap).toupper()
HÉTFŐ
```

Azaz itt is nagy a szabadságunk arra vonatkozólag, hogy egy problémát milyen módszerrel oldunk meg.

1.4.16 Formázás testre szabása (Export-FormatData, Get-FormatData, Update-FormatData)

A PowerShell parancsainak, kifejezéseinek és csővezetékeinek a kimenetét valójában az `Out-Default` cmdletnek köszönhetjük, amelyet a parser a háttérben hozzábiggyeszt a kimenetet adó kifejezések végére. Ha a kimenet egyszerű sztring, akkor az továbbadódik az `Out-Host`-nak, és az megjeleníti a szöveget a képernyőn.

Ha a kimenet nem sztring, akkor `Out-Default` megvizsgálja az objektumot és megnézni, hogy van-e nézet definiálva az objektumtípushoz. Ilyen nézetdefiníciók már „gyárilag” is vannak a rendszerben, de mi magunk is tudunk ilyeneket létrehozni XML adatformátumban, amelyeket az `Update-FormatData` cmdlet segítségével tudunk regisztrálni a PowerShell környezettel. Ebben az XML fájlban definiáljuk a `Format-Wide`, `-List`, `-Table` és `-Custom` cmdletek futtatására megjelenő formátumot, azaz hogy melyik tulajdonságokat írja ki és milyen formában.

Ha nincs az adattípusnak regisztrált nézete, akkor az `Out-Default` megnézi, hogy a kimenet első objektumának vajon van-e legalább öt tulajdonsága. Ha igen, akkor lista nézetet használ, ha nincs, akkor táblázatos nézetet:

```
[10] PS C:\> $p = @{e=1;k=2}
[11] PS C:\> $x = New-Object -Property $p -TypeName psobject
[12] PS C:\> $x

                                        k                                       e
                                        -                                       -
                                        2                                       1
[13] PS C:\> $p =@{e=1;k=2;h=3;n=4;o=5;t=6}
[14] PS C:\> $y = New-Object -Property $p -TypeName psobject
[15] PS C:\> $y


o : 5
h : 3
k : 2
t : 6
e : 1
n : 4
```

A [10]-[12] sorokban egy egyedi objektumot hoztam létre két tulajdonsággal. A kimenet táblázatos lett. A [13]-[15] sorokban hasonló módon hat tulajdonsággal hoztam létre objektumot, így alaphelyzetben lista nézetet kaptam. Fontos hangsúlyozni, hogy az első objektum dönt:

```
[20] PS C:\> $x, (Get-Process powershell) | Format-Table

                                        k                                       e
                                        -                                       -
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                                                    2                                                    1
```

A fenti példában még az is látszik, hogy szegény PowerShell processz meg sem jelent a `Format-Table` kimenetén, mert nincs se „`k`”, se „`e`” tulajdonsága.

Nézzük, hogyan lehet az egyéni típusokat kicsit szebben megjeleníteni. Az előző fejezetben létrehozott email típus egy objektuma alaphelyzetben láthattuk, hogy így néz ki. Most definiálok ilyen e-mail címek egy tömbjét:

```
[4] PS C:\> $addrs = @()
[5] PS C:\> $addrs += New-Object email soost, iqjb.hu
[6] PS C:\> $addrs += New-Object email bakaigy, iqjb.hu
[7] PS C:\> $addrs += New-Object email soostibor, citromail.hu
[8] PS C:\> $addrs += New-Object email valaki, citromail.hu
[9] PS C:\> $addrs += New-Object email tanfolyam, iqjb.hu
[10] PS C:\> $addrs

alias                       domain                    fullemail
-----                       ------                    ---------
soost                       iqjb.hu                   soost@iqjb.hu
bakaigy                     iqjb.hu                   bakaigy@iqjb.hu
soostibor                   citromail.hu              soostibor@citromail.hu
valaki                      citromail.hu              valaki@citromail.hu
tanfolyam                   iqjb.hu                   tanfolyam@iqjb.hu
```

Ha például nekem jobban tetszene az, hogy az első oszlopban jelenjen meg a `fullemail` cím tulajdonság, és e-mail tartományonként csoportosítva legyenek a címek, akkor ehhez készíthetek egy speciális formázó XML kifejezést, ami nagyon hasonlít a típusok testre szabásánál látott PS1XML fájlokhoz:

```
<?xml version="1.0" encoding="utf-8" ?>

<Configuration>

    <ViewDefinitions>
                <View>
            <Name>email</Name>
            <ViewSelectedBy>
                <TypeName>email</TypeName>
            </ViewSelectedBy>
            <GroupBy>
                <PropertyName>domain</PropertyName>
            </GroupBy>

            <TableControl>
                <TableHeaders>
                    <TableColumnHeader>
                        <Width>30</Width>
                    </TableColumnHeader>
                    <TableColumnHeader>
                        <Width>15</Width>
                    </TableColumnHeader>
                    <TableColumnHeader>
                        <Width>15</Width>
                    </TableColumnHeader>
                </TableHeaders>
                <TableRowEntries>
                    <TableRowEntry>
```

```
                        <TableColumnItems>
                            <TableColumnItem>
                                <PropertyName>Fullemail</PropertyName>
                            </TableColumnItem>
                            <TableColumnItem>
                                <PropertyName>Alias</PropertyName>
                            </TableColumnItem>
                            <TableColumnItem>
                                <PropertyName>Domain</PropertyName>
                            </TableColumnItem>
                        </TableColumnItems>
                    </TableRowEntry>
                </TableRowEntries>
            </TableControl>
        </View>
    </ViewDefinitions>
</Configuration>
```

Ez is egy PS1XML kiterjesztésű fájl, ebben tehát meghatároztam a `<TypeName>` címkénél, hogy melyik típushoz tartozik, majd a táblázat csoportosítását adtam meg, majd a fejlécét és oszlopait definiáltam. Ezután ezt a be kell tölteni a PowerShell alá az `Update-FormatData` cmdlet segítségével:

```
[37] PS C:\> Update-FormatData -AppendPath C:\munka\email.format.ps1xml
```

Lehet a PowerShell saját típusdefinícióihoz képest később (`-AppendPath`) vagy előbb (`-PrependPath`) betölteni ezeket a formátumdefiníciókat. Ha később töltjük be, akkor ezzel felül lehet bírálni a „gyári” formátumokat, ha előbb töltjük be, akkor csak akkor jut érvényre a formátum, ha azt később „gyári” formátum nem írja felül.

Nézzük, hogy ezután hogyan jelenik meg a táblázatom:

```
[12] PS C:\> $addrs


   domain: iqjb.hu

Fullemail                       Alias           Domain
---------                       -----           ------
soost@iqjb.hu                   soost           iqjb.hu
bakaigy@iqjb.hu                 bakaigy         iqjb.hu


   domain: citromail.hu

Fullemail                       Alias           Domain
---------                       -----           ------
soostibor@citromail.hu          soostibor       citromail.hu
valaki@citromail.hu             valaki          citromail.hu


   domain: iqjb.hu

Fullemail                       Alias           Domain
---------                       -----           ------
tanfolyam@iqjb.hu               tanfolyam       iqjb.hu
```

Használata nem Engedélyezett!

Látható, hogy itt is a csoportosítás önmagában nem rendezi az elemeket, azaz csak egymáshoz képest nézi, hogy vajon új csoportot kell nyitni, vagy sem.

A „gyári” formátumfájlok a `$PSHome` elérési úton találhatók, nevükben benne van a `format` kifejezés. A legtöbb objektumtípus formázása a `DotNetTypes.format.ps1xml` és a `PowerShellCore.format.ps1xml` fájlban található.

Ezeket formázási információkat a `Get-FormatData` cmdlettel lehet kiolvasni:

```
[21] PS C:\> Get-FormatData

TypeName                                  FormatViewDefinition
--------                                  --------------------
Microsoft.PowerShell.Commands.GetCou... {Counter , TableControl}
Microsoft.PowerShell.Commands.GetCou... {Counter , TableControl}
System.Xml.XmlElement#http://schemas... {System.Xml.XmlElement#http://schem...
Microsoft.WSMan.Management.WSManConf... {Microsoft.WSMan.Management.WSManCo...
…
email                                     {email , TableControl}
```

A lista alján található az én email osztályom megjelenítési adata. Ezt nézzük kicsit részletesebben, miután ez egy hierarchikus XML adat, ezért a legpraktikusabb a `Format-Custom` cmdlet használata:

```
[30] PS C:\> Get-FormatData email | Format-Custom

class ExtendedTypeDefinition
{
  TypeName = email
  FormatViewDefinition =
    [
      class FormatViewDefinition
      {
        Name = email
        Control =
          class TableControl
          {
            Headers =
              [
                class TableControlColumnHeader
                {
                  Label =
                  Alignment = Undefined
                  Width = 30
                }
                class TableControlColumnHeader
                {
                  Label =
                  Alignment = Undefined
                  Width = 15
                }
                class TableControlColumnHeader
                {
                  Label =
                  Alignment = Undefined
                  Width = 15
                }
              ]

            Rows =
```

```
            [
              class TableControlRow
              {
                Columns =
                  [
                    Fullemail
                    Alias
                    Domain
                  ]

              }
            ]

        }
      }
    ]

}
```

Itt gyakorlatilag visszakaptam az eredeti XML fájl tartalmát, az emberi szemnek talán kicsit olvashatóbb formában.

Ezekben a formátum fájlokban lehetnek számolt információk is, hasonlóan, ahogy a `types.PS1XML` fájlokban is, de ilyenek létrehozása túlmutat jelen könyv keretein.

1.4.17 Objektumok mentése, visszatöltése

A PowerShell lehetőséget biztosít objektumok elmentésére és visszatöltésére. Ez nagyon praktikus, hiszen ha épp a konzolon a változóim jól fel vannak töltve mindenféle objektummal és nekem valami miatt be kell csuknom a PowerShell ablakot, akkor a változóim törlődnek és legközelebb újra elő kellene állítanom őket.

Az `Export-CliXML` cmdlettel ki tudom ezeket menteni egy fájlba, bezárhatom a PowerShell ablakot, majd később vissza tudom importálni a változóim értékét az `Import-CliXML` cmdlettel.

Nézzünk erre egy példát:

```
[13] PS C:\> $p = Get-Process w*
[14] PS C:\> $p

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    123       7     2836       7924    74     0,09   2464 WindowsSearch
    655     159    10748       2976    97    11,77    440 winlogon
    700      49    40684      78780   280    50,63   2316 WINWORD
    215      12     4708       8584    79     0,11   2196 WLLoginProxy
    161       7     2704       7728    39     0,19   1812 wmiprvse
     30       3     1020       3088    50     0,02    504 wscntfy


[15] PS C:\> $p | get-member


   TypeName: System.Diagnostics.Process

Name                           MemberType     Definition
----                           ----------     ----------
Handles                        AliasProperty  Handles = Handlecount
Name                           AliasProperty  Name = ProcessName
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
NPM                            AliasProperty  NPM = NonpagedSystemMemorySize
PM                             AliasProperty  PM = PagedMemorySize
VM                             AliasProperty  VM = VirtualMemorySize
WS                             AliasProperty  WS = WorkingSet
add_Disposed                   Method         System.Void add_Disposed(Event...
...

[16] PS C:\> $p | Export-Clixml c:\export.xml
[17] PS C:\> Clear-Variable p
[18] PS C:\> $p
[19] PS C:\> $p = Import-Clixml C:\export.xml
[20] PS C:\> $p

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    123       7     2836       7924    74     0,09   2464 WindowsSearch
    655     159    10748       2976    97    11,77    440 winlogon
    700      49    40684      78780   280    50,63   2316 WINWORD
    215      12     4708       8584    79     0,11   2196 WLLoginProxy
    161       7     2704       7728    39     0,19   1812 wmiprvse
     30       3     1020       3088    50     0,02    504 wscntfy

[21] PS C:\> $p | get-member


   TypeName: Deserialized.System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize
PM                         AliasProperty  PM = PagedMemorySize
VM                         AliasProperty  VM = VirtualMemorySize
WS                         AliasProperty  WS = WorkingSet
__NounName                 NoteProperty   System.String __NounName=Process
BasePriority               Property       System.Int32 {get;set;}
Container                  Property        {get;set;}
EnableRaisingEvents        Property       System.Boolean {get;set;}
...
```

A fenti példában a `$p` változó megkapta a W-vel kezdődő processz-objektumok tömbjét. A [15]-ben látszik, hogy ez tényleg jól nevelt `System.Diagnostics.Process` objektumokat tartalmaz, a rájuk jellemző tulajdonságokkal és metódusokkal együtt.

Ezután kiexportáltam `$p`-t, töröltem majd visszaimportáltam. [20]-ban megnézem, mi van `$p`-ben, szemre teljesen ugyanaz, mint korábban. Azonban a [21] `get-member`-je felfedi, hogy azért ez mégsem 100%-osan ugyanaz az objektum, hiszen elveszítette a metódusait és típusa is más lett:

`Deserialized.System.Diagnostics.Process`.

Mindenesetre, ha a statikus adataival szeretnénk csak foglalkozni ez bőven elég, éles, „real time” adatokkal meg úgysem biztos, hogy tudnánk dolgozni, hiszen az export óta lehet, hogy megszűntek már a korábban futó processzek.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5 Operátorok

Minden programnyelv alapvető elemei az operátorok. A PowerShell alkotói az operátorok esetében is arra törekedtek, hogy minél „hatásosabbak” legyenek, azaz a lehető legtömörebb kifejezésekkel lehessen a legtöbb eredményt elérni.

Ebben a fejezetben a PowerShell operátorait mutatom be.

1.5.1 Aritmetikai operátorok

A legtriviálisabb operátorok – a kifejezés eredeti jelentése alapján – valós számokon végezhető műveletekhez kötődik. A PowerShell ezen jóval túlmutat, a matematikai tanulmányainkban megszokott műveletek ki vannak terjesztve egyéb objektumokra: többek között a tömbökre, sztringekre, fájlokra.

1.5.1.1 Összeadás

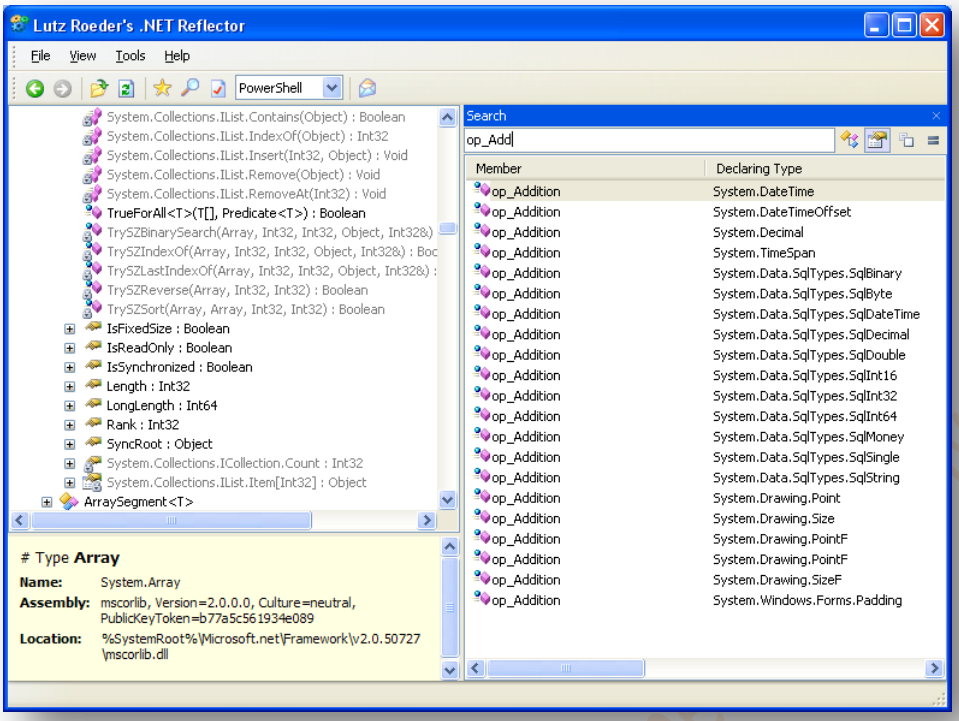
Rögtön nézzünk is példákat az összeadásra:

```
[1] PS C:\> 5+4
9
[2] PS C:\> "ab" + "lak"
ablak
[3] PS C:\> "egy", "kettő", "három" + "négy", "öt"
egy
kettő
három
négy
öt
```

Látszik, hogy akár az egészekre, sztringekre vagy a tömbökre is értelmezett az összeadás. Ebben jelentős szerepe van a .NET Framework osztálydefinícióinak, hiszen valójában az ő érdemük, az ő megfelelő összeadást végző metódusuké, hogy mi is történik ténylegesen az összeadás hatására. A PowerShell „csak” abban ludas, hogy megtalálja az adott metódust és átalakítsa a kifejezést, esetlegesen típuskonverziót végezzen az operandusok között.

A Reflector programmal ezeket meg is tudjuk nézni, a számoknál az `op_Addition` metódust kell keresni:

A könyv tanfolyami felhasználása nem engedélyezett!



29. ábra Az összeadás metódusainak felderítése a Reflectorral

A sztringeknél talán az `Append` metódust, de ezzel nekünk nem kell törődni, ezt a PowerShell helyettünk elvégzi.

Ezen kívül még dátumokat és esetlegesen hashtáblákat szoktunk még összeadni:

```
[4] PS I:\>$hash1 = @{Első = 1; Második = 2}
[5] PS I:\>$hash2 = @{Harmadik = 3; Negyedik = 4}
[6] PS I:\>$hash1 + $hash2

Name                           Value
----                           -----
Második                        2
Harmadik                       3
Negyedik                       4
Első                           1
```

Ha olyan hashtáblát próbálunk hozzáadni egy meglevőhöz, amiben újra szerepel egy korábbi elem, akkor hibát kapunk:

```
[8] PS C:\> $hash3 = @{Harmadik = 33; hatodik = 6}
[9] PS C:\> $hash2 + $hash3
Bad argument to operator '+': Item has already been added. Key in dictionary:
'Harmadik'  Key being added: 'Harmadik'.
At line:1 char:9
+ $hash2 + <<<<  $hash3
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : BadOperatorArgument
```

Dátumoknál kicsit zűrösebb a helyzet:

```
[12] PS I:\>get-date

2008. március 11. 9:53:26


[13] PS I:\>(get-date) + 5000000000000

2008. március 17. 4:47:00
```

Jó nagy számot kellett a dátumhoz adnom, hogy valami látható legyen az eredményen, mert valójában a dátum un. „tick”-ekben, „ketyegésekben” számolódik, ami 100 ns-os felbontású időtárolást tesz lehetővé.

Épp ezért a dátumoknál már eleve van mindenféle praktikusabb `Add…` metódus:

```
[18] PS I:\>get-date | Get-Member


   TypeName: System.DateTime

Name                 MemberType     Definition
----                 ----------     ----------
Add                  Method         System.DateTime Add(TimeSpan value)
AddDays              Method         System.DateTime AddDays(Double value)
AddHours             Method         System.DateTime AddHours(Double value)
AddMilliseconds      Method         System.DateTime AddMilliseconds(Double v...
AddMinutes           Method         System.DateTime AddMinutes(Double value)
AddMonths            Method         System.DateTime AddMonths(Int32 months)
AddSeconds           Method         System.DateTime AddSeconds(Double value)
AddTicks             Method         System.DateTime AddTicks(Int64 value)
AddYears             Method         System.DateTime AddYears(Int32 value)
...

[19] PS I:\>get-date

2008. március 11. 9:57:55


[20] PS I:\>(get-date).AddDays(5)

2008. március 16. 9:58:10
```

A fenti példában először kilistáztam a `get-date` által visszaadott `[DateTime]` típus metódusait, majd a [20]-as sorban 5 napot adtam az aktuális dátumhoz.

1.5.1.2 Többszörözés

Az összeadáshoz hasonlóan a szorzás is jelentős általánosításon esett át:

```
[10] PS C:\> 6*7
42
[11] PS C:\> "w"*4
wwww
[12] PS C:\> ("BO","CI")*3
BO
CI
BO
CI
BO
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
CI
[13] PS C:\> "BO","CI"*3
BO
CI
BO
CI
BO
CI
[14] PS I:\>("BO","CI"*3).length
6
```

Láthatjuk, hogy a szöveget is lehet szorozni, ez annyiszor teszi össze a szöveget, ahány a szorzásjel után áll.

Tömbnél is hasonlóan többszörözi a tagokat. A [13]-as promptnál látszik, hogy nem is kell zárójelet használni! Ez a PowerShell alkotóinak az érdeme.

Megjegyzés

Érdekes módon a [14]-es promptban látszik, hogy a kéttagú tömb háromszorozása után nem háromelemű lett (3 darab kételemű), hanem hatelemű!

Nézzük meg, hogy mi történik, ha felcseréljük a szorzás operandusit:

```
[15] PS C:\> 3*"BO","CI"
The '*' operator failed: Cannot convert the "System.Object[]" value of type "S
ystem.Object[]" to type "System.Int32"..
At line:1 char:3
+ 3* <<<< "BO","CI"
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : OperatorFailed
```

Azt láthatjuk, hogy nem mindegy, hol van a szorzó és a tényező. A szorzónak kell mindig jobb oldalon állnia, ha nem, akkor hibát kaphatunk, mint ahogy az a [15]-es és [16]-ös sor után is kaptunk, hiszen a PowerShell az automatikus típuskonverziót balról jobbra végzi, azaz az [int] típust veszi alapnak, és ehhez próbálja igazítani a szorzás következő tényezőit, ami itt – szövegek és kételemű tömb esetén - nem sikerült.

1.5.1.3 Osztás, maradékos osztás

Természetesen osztás is van a PowerShellben:

```
[32] PS I:\>15/5
3
[33] PS I:\>(15/5).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType


[34] PS I:\>15/6
2,5
[35] PS I:\>(15/6).GetType()

IsPublic IsSerial Name                                     BaseType
```

```
-------- -------- ----                                           --------
True     True     Double                                         System.ValueType


[36] PS I:\>2,4/1,2
Method invocation failed because [System.Object[]] doesn't contain a method na
med 'op_Division'.
At line:1 char:5
+ 2,4/ <<<< 1,2
    + CategoryInfo          : InvalidOperation: (op_Division:String) [], Runt
   imeException
    + FullyQualifiedErrorId : MethodNotFound

[37] PS I:\>(15/6)/(125/100)
2
```

A [32]-es prompt egyértelmű, 15-öt elosztunk 5-tel, 3-at kapunk, nem is akármilyen 3-at: `Int32` típusút! Szemben a 15/6-tal, aminek eredménye természetesen [35]-ben `Double`.

Egy kicsit időzzünk el itt! Hogyan is néz ki a [34] eredményeként kapott `2,5`? A tizedes elválasztására vesszőt használt a PowerShell! Viszont a [36]-ban nem szerette ezt, mert a `(,)` általában tömb jelzésére szolgáló operátor. Akkor vajon fog tudni dolgozni ezzel az általa kiadott `2,5`-gyel? A [37]-ben azt látjuk, hogy természetesen igen! És itt jön ki a PowerShell objektumalapúságának nagy előnye! Annak ellenére, hogy a megjelenítésnél használja az aktuális gép területi beállításait, és ennek megfelelően jeleníti meg a számokat, amikor ő számol tovább egy objektummal, akkor területi beállítás-mentesen kezeli ezt. Ha sztring alapú lenne a PowerShell, hasonlóan a Linuxos/Unixos shellekhez, akkor vagy a kimenet nem venné figyelembe a területi beállításokat, vagy egy csomó sztring-hókuszpókuszt kellene elvégezni ahhoz, hogy ilyen egyszerű műveletek működhessenek. Ha mi adunk be számokat a parancssor számára, akkor mindig ponttal jelezzük a tizedesjelet, nem a területi beállítás alapján kell beírni a számokat.

Ebbe a fejezetbe tartozik még a maradékos osztás művelete, a %:

```
[41] PS I:\>21%5
1
[42] PS I:\>1.4%1.1
0,3
```

Ezzel nincs sok trükk, hiszen ez az operátor csak számokra van értelmezve, viszont nem csak egészekre, ahogy ez a [42]-ben látszik.

1.5.2 Értékadás

Értéket már számtalanszor adtam, így az alapeset nem fog senkinek meglepetést okozni:

```
[49] PS I:\>$a=5
[50] PS I:\>$a
5
```

Azonban nem csak ilyen módon lehet értéket adni.

```
[51] PS I:\>$a+=8
[52] PS I:\>$a
13
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[53] PS I:\>$a-=4
[54] PS I:\>$a
9
[55] PS I:\> $a/=3
[56] PS I:\> $a
3
[57] PS I:\> $a*=10
[58] PS I:\> $a
30
```

Az [51]-től kezdődően látszik, hogy ha ugyanannak a változónak az aktuális értékével szeretnénk műveletet végezni és utána ezt visszatölteni a változónkba, akkor ennek tömör formáját is alkalmazhatjuk az adott műveleti jel és az egyenlőségjel kombinációjával.

Nézzük a trükkösebb formákat! Ha ugyanolyan értéket szeretnék adni több változónak is, akkor ezt tömören is megtehetem:

```
[59] PS I:\>$x=$y=$z=81
[60] PS I:\>$x
81
[61] PS I:\>$y
81
[62] PS I:\>$z
81
```

Vigyázzunk a vesszővel!

```
[63] PS I:\>$x, $y, $z, $w = 1,2,3
[64] PS I:\>$x
1
[65] PS I:\>$y
2
[66] PS I:\>$z
3
[67] PS I:\>$w
[68] PS I:\>
```

A [63]-as sorban vesszővel soroltam fel a változóimat, amelyeknek szintén vesszővel elválasztott értékeket adok. Ahelyett, hogy mind a négy változóm felvenné a háromelemű tömböt értékként, az első változóm megkapta az első tagot, a második a másodikat és így tovább. Szegény `$w`-nek nem jutott érték, így ő üres maradt.

Ha fordított a helyzet, azaz a változók vannak kevesebben, mint az értékek, akkor az utolsó megkapja a maradékot egy tömbként:

```
[69] PS I:\>$x, $y = 1,2,3
[70] PS I:\>$y
2
3
```

Ez a változók felsorolásának lehetősége jól jöhet szövegek feldolgozásakor.

```
[71] PS I:\>$sor = "Soós Tibor Budapest"
[72] PS I:\>$vezetéknév, $keresztnév, $város = $sor.Split()
[73] PS I:\>$vezetéknév
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Soós
[74] PS I:\>$keresztnév
Tibor
[75] PS I:\>$város
Budapest
```

A [72]-ben látható `Split()` metódus a sztringek gyakran felhasznált metódusa, ami a paramétereként átadott karakter mentén feldarabolja a sztringet. Ha üresen hagyjuk a paraméterét (mint most), akkor a normál szóelválasztó karakterek mentén tördel. Kimeneteként az így széttördelt sztringekből álló tömböt adja vissza, amit szépen betöltök a változóimba.

1.5.3 Összehasonlító operátorok

Az összehasonlítás műveletét végző összehasonlító operátorok jelzésére a PowerShell alkotói nem a hagyományos jeleket (=, <, >, <>, !=, stb.) választották, hanem külön jelölést vezettek be, rögtön két különböző szériát a kis-nagybetű érzéketlen és érzékeny változatra:

Operátor	Leírás
-eq	egyenlő
-ne	nem egyenlő
-gt	nagyobb
-ge	nagyobb egyenlő
-lt	kisebb
-le	kisebb egyenlő

Kis-nagybetű érzékeny operátor	Leírás
-ceq	egyenlő
-cne	nem egyenlő
-cgt	nagyobb
-cge	nagyobb egyenlő
-clt	kisebb
-cle	kisebb egyenlő

Megjegyzés

„Érzéketlen” változatként használhatjuk az operátorok „i” betűs megfelelőit is: `-ieq, -ine, -igt,` stb., de ezek az alap változattal teljesen ekvivalensen működnek.

Értelemszerűen az „érzékeny” változatot elsődlegesen szövegek összehasonlításakor használjuk, de a számoknál se ad hibát alkalmazásuk. Nézzünk akkor néhány példát mindezek alkalmazására:

```
[79] PS I:\>1 -ceq 1
True
[80] PS I:\>1 -eq 2
False
[81] PS I:\>1 -eq 1
True
[82] PS I:\>1 -ceq 1
True
[83] PS I:\>"ablak" -eq "ABLAK"
True
[84] PS I:\>"ablak" -ceq "ABLAK"
False
[85] PS I:\>5 -gt 2
True
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Sokat azt hiszem nem is kell magyarázni.

Talán azt érdemes megnézni, hogy a szövegeket hogyan tudom „nagyság” szempontjából összehasonlítani:

```
[86] PS I:\>"ablak" -gt "ajtó"
False
[87] PS I:\>"baba" -gt "ablak"
True
[88] PS I:\>"ablak" -lt "állat"
True
[89] PS I:\>"asztal" -lt "állat"
False
[90] PS I:\>"á" -lt "a"
False
[91] PS I:\>"ab" -cgt "Ab"
False
```

Látszik, hogy itt a területi beállításoknak megfelelő szótár-sorrend alapján dől el, hogy melyik sztring nagyobb a másiknál.

Vigyázat! A PowerShell összehasonlítás során is végez automatikus típuskonverziót:

```
[1] PS I:\>"1234" -eq 1234
True
[2] PS I:\>1234 -eq "1234"
True
```

Persze nem mindenkor tud okos lenni, ha nagyon akarjuk, akkor átverhetjük:

```
[3] PS I:\>"0123" -eq 123
False
```

Ugye itt a 0-val kezdődő, idézőjelek közé tett szám esetén azt tartja valószínűbbnek, hogy ez szöveg, és akkor a jobb oldalon levő részt is szöveggé konvertálja magában, márpedig a „0123” nem egyenlő „123”-mal.

Nézzük meg, hogy vajon tömbök esetében hogyan működnek ezek az összehasonlító operátorok?

```
[10] PS I:\>1,2,3,4,1,2,3,1,2 -eq 1
1
1
1
[11] PS I:\>1,2,3,4,1,2,3,1,2 -eq 2
2
2
2
[12] PS I:\>1,2,3,4,1,2,3,1,2 -lt 3
1
2
1
2
1
[13] PS I:\>1,2,3,4,1,2,3,1,2 -eq 4
4
[14] PS I:\>1,2,3,4,1,2,3,1,2 -eq 5
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Hoppá! Az első, amit láthatunk tömbök esetében, hogy nem `True` vagy `False` értéket kapunk, hanem azokat az elemeket, amelyekre igaz az adott összehasonlító művelet. Sőt! Ahogy a [14]-es promptban is látható, ha nincs egyezés, akkor sem kapunk `False`-t, hanem „semmi” a válasz.

Ha a jobb oldalon van tömb, akkor nem igazán kapunk egyenlőséget semmilyen esetben sem:

```
[19] PS I:\>(1,2),3 -eq 1,2
[20] PS I:\>1,2 -eq 1,2
[21] PS I:\>1 -eq 1,2
False
```

1.5.4 Tartalmaz (-contains, -notcontains, -ccontains, -cnotcontains)

Az előző fejezetben azt láthattuk, hogy az összehasonlító `-eq` elég furcsa módon működik tömbök esetében. Ezért, hogy ha csak arra vagyunk kíváncsiak, hogy egy tömb tartalmaz-e valamilyen elemet, akkor erre a `-contains` operátor használható, illetve ennek negáltja, a `-notcontains`, valamint ha kell, akkor ezeknek kis-nagybetű érzékeny változatai:

```
[24] PS I:\>1,2,3,4,1,2,3,1,2 -contains 3
True
[25] PS I:\>1,2,3,4,1,2,3,1,2 -notcontains 5
True
[26] PS C:\> "a", "A", "a", "c" -ccontains "a"
True
[27] PS C:\> "a", "A", "a", "c" -ccontains "C"
False
```

Sajnos itt is vigyázni kell, ha a jobb oldalon tömb áll, mert nem ad találatot:

```
[28] PS I:\>(1,2),3 -contains (1,2)
False
```

1.5.5 Dzsóker-minták (-like)

Ha valaki a számítógépekkel kezd el foglalkozni, azon belül a fájlokkal, akkor viszonylag hamar találkozik a (*) karakterrel, mint dzsoli-dzsókerrel. A PowerShellben nagyon kiterjedt lehetőségeket adnak az ilyen jellegű dzsoli-dzsókerek, akár a hagyományosnak tűnő „DOS” parancsokkal is:

```
[2] PS C:\scripts> dir


    Directory: C:\scripts

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         2008.02.22.    22:47       <DIR> EntryForm
-a---         2008.01.15.    20:21          709 alice.txt
-a---         2008.01.11.    11:36          235 coffee.txt
-a---         2008.02.12.    11:07          382 DebugMe.pl
-a---         2008.02.12.    11:07          253 DebugMe.ps1
-a---         2008.02.12.    11:06          823 DebugMe.vbs
-a---         2008.02.08.    20:47           32 lettercase.txt
-a---         2008.02.08.    20:55           22 numbers.txt
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
-a---   2008.02.11.     8:36        42496 Password_Checklist.doc
-a---   2008.01.14.     8:16       229376 pool.mdb
-a---   2008.02.09.    21:15          726 presidents.txt
-a---   2008.02.12.    11:34         1760 readme.txt
-a---   2008.02.05.    13:52         3366 skaters.txt
-a---   2007.01.03.     8:00         2139 songlist.csv
-a---   2008.02.08.    20:48           80 symbols.txt
-a---   2008.02.08.    20:46           46 vertical.txt
-a---   2007.01.03.     8:00        60358 votes.txt
-a---   2007.01.03.     8:00       328620 wordlist.txt


[3] PS C:\scripts> dir *.txt


    Directory: C:\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.01.15.    20:21          709 alice.txt
-a---   2008.01.11.    11:36          235 coffee.txt
-a---   2008.02.08.    20:47           32 lettercase.txt
-a---   2008.02.08.    20:55           22 numbers.txt
-a---   2008.02.09.    21:15          726 presidents.txt
-a---   2008.02.12.    11:34         1760 readme.txt
-a---   2008.02.05.    13:52         3366 skaters.txt
-a---   2008.02.08.    20:48           80 symbols.txt
-a---   2008.02.08.    20:46           46 vertical.txt
-a---   2007.01.03.     8:00        60358 votes.txt
-a---   2007.01.03.     8:00       328620 wordlist.txt


[4] PS C:\scripts> dir s*.txt


    Directory: C:\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.02.05.    13:52         3366 skaters.txt
-a---   2008.02.08.    20:48           80 symbols.txt


[5] PS C:\scripts> dir [ad]*.txt


    Directory: C:\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.01.15.    20:21          709 alice.txt


[6] PS C:\scripts> dir [a-r]*.txt


    Directory: C:\scripts
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          2008.01.15.     20:21            709 alice.txt
-a---          2008.01.11.     11:36            235 coffee.txt
-a---          2008.02.08.     20:47             32 lettercase.txt
-a---          2008.02.08.     20:55             22 numbers.txt
-a---          2008.02.09.     21:15            726 presidents.txt
-a---          2008.02.12.     11:34           1760 readme.txt


[7] PS C:\scripts> dir ?o*.txt


    Directory: C:\scripts


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          2008.01.11.     11:36            235 coffee.txt
-a---          2007.01.03.      8:00          60358 votes.txt
-a---          2007.01.03.      8:00         328620 wordlist.txt
```

Azt hiszem, a fenti példák magukért beszélnek. Kiemelném az [a-r] formulát, tehát ez az „a” és „r” közti összes betűt helyettesíti.

Ilyen jellegű „dzsókeres” kifejezéseket lehet használni a `-like`, `-notlike` vizsgálatokkal és a kis-nagybetű érzékeny változataikkal: `-clike`, `-cnotlike`:

```
[11] PS C:\scripts> "ablak" -like "[a-f]lak"
False
[12] PS C:\scripts> "ablak" -like "[a-f]blak"
True
[13] PS C:\scripts> "ablak" -clike "[A-F]blak"
False
[14] PS C:\scripts> "blak" -clike "[A-F]blak"
False
[15] PS C:\scripts> "blak" -like "[a-f]blak"
False
```

Hasonlóan bánik el a tömbökkel is a `-like` és a többi dzsókeres operátorunk, mint ahogyan az `-eq` is:

```
[17] PS C:\scripts> "ablak", "abrosz", "alabástrom", "baba" -like "ab[l-r]*"
ablak
abrosz
```

Természetesen dzsókeres kifejezés csak az operátor jobb oldalán állhat.

1.5.6 Regex (-match, -replace)

A dzsókerek nagyon praktikusak, de egy csomó mindenre nem jók. Például szeretnénk megvizsgálni, hogy egy szöveg e-mail cím formátumú-e? Vagy van-e a szövegben valahol egy webcím, vagy telefonszám? Mivel ilyen jellegű vizsgálatok nagyon gyakoriak a számítástechnikában, ezért erre külön „tudományág” alakult, matematikai alapját pedig a formális nyelvek képezik.

Ak... tanfo... ...lása nem

Az ilyen jellegű szövegminta vizsgálatok kifejezéseit hívják Regular Expression-nek, vagy röviden Regex-nek. Miután ez ténylegesen külön tudomány, Regular Expression kifejezésekről számtalan könyv látott napvilágot, ezért itt most nem vállalkozom arra, hogy még csak közelítő részletességgel tárgyaljam ezt a témát, de azért a főbb elveket és jelöléseket megpróbálom bemutatni.

1.5.6.1 Van-e benne vagy nincs?

Elsőként nézzünk a legegyszerűbb esetet, amikor egy szövegről el akarjuk dönteni, hogy van-e benne általunk keresett mintára hasonlító szövegrész:

```
[12] PS C:\> "Benne van a NAP szó" -match "nap"
True
[13] PS C:\> "Benne van a NAPfény szó" -match "nap"
True
```

A fenti két példa még nem igazán mutatja meg a regex erejét, akár a `-like` operátor is használható lett volna:

```
[14] PS C:\> "Benne van a NAP szó" -like "*nap*"
True
```

De vajon mit tennénk, ha a „napfény”-t nem tekintenénk a mintánkkal egyezőnek, csak az önálló „nap”-ot? Az önállóságot persze sokfajta karakter jelölheti: szóköz, pont, zárójel, stb. A regex-szel ez nagyon egyszerű, használni kell a „szóvég” jelölő „\b” karakterosztály szimbólumot a mintában:

```
[15] PS C:\> "Benne van a NAPfény szó" -match "\bnap\b"
False
[16] PS C:\> "Benne van a NAP. Fény is" -match "\bnap\b"
True
[17] PS C:\> "Benne van a (NAP)" -match "\bnap\b"
True
```

Azaz keresek egy olyan mintát, amely áll egy szóelválasztó karakterből, a „nap”-ból, és megint egy szóelválasztó karakterből.

Megjegyzés:

Mint ahogy láttuk, egy reguláris kifejezésben a „\b” szóhatárt jelöl, de ha ezt szögletes zárójelek között „[]” alkalmazzuk (lásd később a karakterosztályokat), akkor a „\b” a visszatörlés, azaz a „backspace” karaktert jelenti. A cserénél (`-replace`, lásd szintén később) a „\b” mindig visszatörlés karakter jelent.

Vagy keresem, hogy van-e a szövegben szám:

```
[26] PS C:\> "Ebben van 1 szám" -match "\d"
True
```

A \d a számjegy jelölője. Ezen jelölőknek van „negáltjuk” is, \B – a nem szóelválasztó karakter, \D – a nem számjegy.

Vagy keresem a Soós nevűeket, akik lehetnek „Sós”-ok is, de azért „Sooós” ne legyen:

```
[23] PS C:\> "Soós" -match "So?ós"
True
[24] PS C:\> "Sós" -match "So?ós"
True
[25] PS C:\> "Sooós" -match "So?ós"
False
```

A „?” 0 vagy 1 darabot jelöl az azt megelőző regex kifejezésből. A * bármennyi darabot jelent, akár 0-t is. A + legalább 1 darabot jelöl. Tetszőleges darabszámot jelölhetünk {*min,max*} formában, ahol a maximum akár el is hagyható.

Kereshetem az „s”-sel kezdődő, „s”-sel végződő háromkarakteres szavakat:

```
[26] PS C:\> "Ez egy jó találat 'sas' nekem" -match "\bs.s\b"
True
[27] PS C:\> "Ez nem jó találat 'saras' nekem" -match "\bs.s\b"
False
```

Alaphelyzetben a „.” minden karaktert helyettesít, kivéve a sortörést.

Egyszerűen vizsgálhatom, hogy van-e a szövegben ékezetes betű:

```
[28] PS C:\> "Ékezet van ebben több is" -match "[áéíóöőüű]"
True
[29] PS C:\> "Ekezetmentes" -match "[áéíóöőüű]"
False
```

A [] zárójelpár közti betűk vagylagosan kerülnek vizsgálat alá, ezt is úgy hívjuk, hogy karakterosztály (meg a \d, \w, stb. kifejezéseket is). Ezzel karakter tartományokat is meg lehet jelölni:

```
[30] PS C:\> "A" -match "[a-f]"
True
[31] PS C:\> "G" -match "[a-f]"
False
```

Természetesen ezeket az alapelemeket kombinálni is lehet. Keresem a legfeljebb négyjegyű hexadecimális számot:

```
[32] PS C:\> "Ebben van négyjegyű hexa: 12AB" -match "\b[0-9a-f]{1,4}\b"
True
[33] PS C:\> "Ebben nincs: 12kicsiindián" -match "\b[0-9a-f]{1,4}\b"
False
[34] PS C:\> "Ebben sincs: 12baba" -match "\b[0-9a-f]{1,4}\b"
False
```

A fenti karakterosztályokban használhatunk negált változatot is. Például tartalmaz olyan karaktert, ami nem szóköz jellegű karakter:

```
[35] PS C:\> "Van nem szóköz is benne" -match "[^\s]"
True
[36] PS C:\> "     " -match "[^\s]"
False
```

A „nem szóköz”-nek van egyszerűbb jelölése is:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[37] PS C:\> "Van nem szóköz is benne" -match "[\S]"
True
[38] PS C:\> "     " -match "[\S]"
False
```

Ennek analógiájára van „nem szókarakter” - \W, „nem számjegy” - \D.

A szögletes zárójelek között csak karaktereket használhatok „vagylagos” értelemben. Ha azt akarom vizsgálni, hogy a szövegben vagy „PowerShell” vagy „PS” található, azt így vizsgálhatom:

```
[39] PS C:\> "Ebben PowerShell van" -match "PowerShell|PS"
True
[40] PS C:\> "Ebben PS van" -match "PowerShell|PS"
True
[41] PS C:\> "Ebben egyik sem" -match "PowerShell|PS"
False
```

Hogy egy kicsit trükkösebb legyen, nézzük az alábbi két példát:

```
[42] PS C:\> "ab" -match "[^ab]"
False
[43] PS C:\> "ab" -match "[^a]|[^b]"
True
```

A [42]-es sorban azért kaptunk hamis eredményt, mert kerestünk egy nem „a”, de nem is „b” karaktert, de az „ab”-ben csak ilyen van, így hamis eredményt kaptam. A [43]-as sorban kerestem egy vagy nem „a”, vagy nem „b” karaktert, márpedig az „ab” első „a”-jára igaz, hogy az nem „b”, tehát van találat!

1.5.6.2 További karakterosztályok

Láthattuk, hogy szögletes zárójelek között vizsgálhatunk többfajta karakter létét vagy hiányát a mintánkban. Ennek van még továbbfejlesztett módja is. Például tagadást másként is ki lehet fejezni:

```
[15] PS C:\> "hijk" -match "[a-z-[h-i]]"
True
```

Itt azt vizsgáltam, hogy a mintában van-e olyan karakter, ami „a” és „z” közötti, kivéve „h” és „i”. Ez itt igaz volt. Ellenben:

```
[16] PS C:\> "hijk" -match "[a-z-[h-k]]"
False
```

Ez már hamisra értékelődött. Ezt még mélyebb beágyazással is lehetne bonyolítani:

```
[23] PS C:\> "h" -match "[a-z-[h-k-[i-j]]]"
False
[24] PS C:\> "i" -match "[a-z-[h-k-[i-j]]]"
True
```

A fenti bonyolult kifejezést így kell értelmezni: vesszük a teljes angol abc-t, ebből kiszedem a „h” és „k” közötti karaktereket, majd „visszarakom” az „i” és „j” közöttieket.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

További lehetőségeink vannak Unicode karakterek esetében, márpedig a PowerShell alaphelyzetben a szövegeket unicode-ként keteli. Ez pedig a `\p{tulajdonság}` formátumú hivatkozás:

Tulajdonság	Jelentés
Lu	Betű, nagybetű
Ll	Betű, kisbetű
Nd	Szám, decimális
No	Szám, egyéb: ² ³ ¹ ¼ ½ ¾
Pd	Elválasztójel: -
Ps	Elválasztójel, nyitó: ([{
Pe	Elválasztójel, záró:)] }
Pi	Elválasztójel, nyitó angol idézőjel: «
Pf	Elválasztójel, záró angol idézőjel: »
Po	Elválasztójel, egyéb: ! " # % & ' * , . / : ; ? @ \ ¡ · ¿
Sm	Szimbólum, matematikai: + < = > | ~ ¬ ± × ÷
Sc	Szimbólum, valuta: $ ¢ £ ¤ ¥
Sk	Szimbólum, módosító: ^ ` ¨ ¯ ´
So	Szimbólum, egyéb: ¦ § © ® ° ¶
Zs	Elválasztó, space
Zl	Elválasztó, sor
Zp	Elválasztó, paragrafus

Ezek vizsgálatához az alábbi kifejezést használtam:

```
[83] PS C:\> $ofs = ""; $minta = [string] (0..255 | ForEach-Object {[char]$_})
[84] PS C:\> $minta
 ☺☻♥♦
♂♀♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼ !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~□□ □ ‚ƒ„…†‡ˆ‰Š‹Œ□ □ □ □‘’“”•–—˜™š›œ□ □ ¡¢£¤¥¦§
¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõ
ö÷øùúûüýþÿ
[85] PS C:\> $m="P"; $reg = [regex] "\p{$m}"; [string] ($reg.Matches($minta) |
Select-Object -Property value | %{[char] $_.value})
!"#%&'()*,-./:;?@[\]_{}¡«-·»¿
[86] PS C:\> $m="Ll"; $reg = [regex] "\p{$m}"; [string] ($reg.Matches($minta) |
 Select-Object -Property value | %{[char] $_.value})
abcdefghijklmnopqrstuvwxyzªµº ßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþÿ
```

A [83]-as sorban létrehoztam egy olyan sztringet, amelyik tartalmazza az összes alap karaktert, majd a [85]-ös sorban készítettem egy olyan regex kifejezést, ami használta ezt a tulajdonság-alapú mintát, és amely visszaadta az összes karaktert egy egyesített sztring formában, amely karakterek kielégítették a regex kifejezést. Látható, hogy a „`\p{P}`”-re az összes elválasztó karaktert kaptam vissza, a [86]-os sorban a „`\p{Ll}`” az összes kis betűt.

A könyv tanfolyami felhasználása nem engedélyezett!

1.5.6.3 Van benne, de mi?

Az eddigiekben a `-match` operátorral arra kaptunk választ egy `$true` vagy `$false` formájában, hogy a keresett minta megtalálható-e a szövegünkben vagy sem. De egy bonyolultabb mintánál, ahol „vagy” feltételek vagy karakterosztályok is vannak, egy `$true` válasz esetében nem is tudjuk, hogy most mit is találtunk meg. Nézzük meg ezt a kettővel előző fejezet [43]-as sorának példája kapcsán:

```
[43] PS C:\> "ab" -match "[^a]|[^b]"
True
[44] PS C:\> $matches

Name                           Value
----                           -----
0                              a
```

Mint látható, a PowerShell automatikusan generál egy `$matches` változót, amely tartalmazza, hogy mit is találtunk a `-match` operátor alkalmazása során. Jelen esetben megtalálhattuk volna a „b”-t is, hiszen arra is igaz, hogy „nem a”, de a `-match` operátor szigorúan balról jobbra halad, így először az „a” karakterre vizsgálja a mintát. Ha ott sikert ér el, akkor nem is halad tovább, megelégszik az eredménnyel és abbahagyja a keresgélést.

A `$matches` igazából nem is egy tömböt, hanem egy hashtáblát tartalmaz, amelynek 0-s kulccsal hivatkozható eleme a megtalált sztring. Nézzünk erre néhány példát:

```
[22] PS C:\> "Ez itt egy példa: szöveg" -match ":.*"
True
[23] PS C:\> $matches[0]
: szöveg
[24] PS C:\> $matches.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Hashtable                                System.Object
```

Kerestük a kettőspontot és utána tetszőleges számú karaktert, illetve látható, hogy a `$matches` tényleg hashtábla.

1.5.6.4 A mohó regex

A `-match` mögött meghúzódó regex motor „mohó”, azaz ha rá bízzuk a mennyiségeket, akkor ő alapesetben annyit vesz belőle, amennyit csak tud a `-match` kielégítése mellett. Ez nem biztos, hogy nekünk mindig jó:

```
[26] PS C:\> "Keresem ezt: xTalálatx, az x-ek közti szöveget" -match "x.*x"
True
[27] PS C:\> $matches[0]
xTalálatx, az x
```

Mi történt itt? Ugye `xTalálatx`-re számítottunk, de „mohóság” miatt az első x megtalálása után a „.*” minta a szövegem végéig kielégíti a feltételt. Csak az utolsó karakter elérése után döbben rá a regex motor, hogy lemaradt az utolsó feltétel, a második x teljesítése, és ekkor kezd visszafele lépkedni, hogy hátha még ezt

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

is tudja teljesíteni valahol. A visszafele lépkedés során eljut a harmadik x-ig, és ott áll meg. Mit lehetett volna tenni, hogy a kívánt találatot kapjuk meg? A regex „mohóságát” ki is lehet kapcsolni, ehhez egy „?”-et kell tenni az ilyen meghatározatlan számú többszörözők után. Ekkor csak a minimális darabszámot veszi a mintából, és ha az nem elég, akkor növeli azt:

```
[32] PS C:\> "Keresem ezt: xTalálatx, az x-ek közti szöveget" -match "x.*?x"
True
[33] PS C:\> $matches[0]
xTalálatx
```

Vagy pontosíthatjuk a „.*” mintát, hiszen ha például a két *x* között szókarakterek lehetnek csak, akkor pontosabb ez a minta:

```
[41] PS C:\> "Keresem ezt: xTalálatx, az x-ek közti szöveget" -match "x\w*x"
True
[42] PS C:\> $matches[0]
xTalálatx
```

Vagy ez is lehet megoldás:

```
[43] PS C:\> "Keresem: xTalálatx, az x-ek közti szöveget" -match "x[^x]+x"
True
[44] PS C:\> $matches[0]
xTalálatx
```

Itt azzal tettem a regexet szerényebbé, hogy az *x* utáni „nem x” karaktereket keresem x-ig. Ez azért is jó, mert jóval hatékonyabb ennek az ellenőrzése, nincs felesleges ide-oda járkálás a mintában. Nagytömegű feldolgozásnál sokkal gyorsabban eredményre jutunk.

1.5.6.5 Escape a Regex-ben

Ezen utóbbi példáknál gondot jelenthet, ha az „x”-nél valami ésszerűbb szeparátor szerepel a keresett szövegrészünk határaként. Ilyen elválasztó karakter szokott lenni mindenfajta zárójel, perjel, stb. Ezek a regexben is legtöbbször funkcióval bírnak, így ha rájuk keresünk, akkor „escape” karakterrel kell megelőzni ezeket. A regexben az „escape” karakter a visszafele-perjel (\), nem pedig a PowerShellben megszokott visszafele aposztróf (`)!

A fenti „x”-es példát cseréljük () zárójel-párra:

```
[45] PS C:\> "A zárójelek közti (szöveget) keresem" -match "\([^)]*\)"
True
[46] PS C:\> $matches[0]
(szöveget)
```

Megjegyzés:

Ne csak a `$matches` változó tartalmára hagyatkozzunk egy `-match` vizsgálat során, mert ha nincs találatunk, attól még a `$matches` tartalmazza egy korábbi `-match` eredményét:

```
[62] PS C:\> "baba" -match "b"
True
[63] PS C:\> $matches[0]
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
b
[64] PS C:\> "baba" -match "c"
False
[65] PS C:\> $matches[0]
b
```

1.5.6.6 Tudjuk, hogy mi, de hol van?

Most már tudjuk, hogy van-e a mintánknak megfelelő szövegrészünk, azt is tudjuk, hogy mi az, csak azt nem tudjuk, hogy hol találta a `-match`. Ez főleg akkor érdekes, ha több találatunk is lehet.

Az alapműködést már megbeszéltük, a regex balról jobbra elemez:

```
[3] PS C:\> "Minta az elején, végén is minta" -match "minta"
True
[4] PS C:\> $matches[0]
Minta
```

Erről meg is győződhetünk, hiszen a fenti találat nagy „M”-mel kezdődik, az pedig a szövegünk legeleje.

A regex lehetőséget biztosít, hogy a szövegünk végén keressük a mintát:

```
[7] PS C:\> "Minta az elején, végén is minta" -match "minta$"
True
[8] PS C:\> $matches[0]
minta
```

Itt a találat kis „m” betűs lett, így ez tényleg a szövegem vége. Tehát a szöveg végét egy „$” jel szimbolizálja a mintában. Ez azonban nem jelent fordított működést, azaz itt is balról jobbra történik a kiértékelés. A „$” jelet úgy lehet felfogni, mintha az az eredeti szövegemben is ott lenne rejtett módon, és így a minta tényleg csak a sor végén illeszkedik a szövegre. Például a szöveg utolsó szava:

```
[10] PS C:\> "Utolsó szót keresem" -match "\b\w+$"
True
[11] PS C:\> $matches[0]
keresem
```

A szöveg elejére is lehet hivatkozni a „kalap” (^) jellel. Tehát az a „kalap” jel más funkciójú a szögletes [] zárójelek között (negálás), mint a „sima” regex mintában, ahol szöveg elejét jelenti:

```
[15] PS C:\> "1234 Elején van-e szám?" -match "^\d+"
True
[16] PS C:\> $matches[0]
1234
[17] PS C:\> "Elején van-e 4321 szám?" -match "^\d+"
False
```

Az előző példában csak a szöveg elején található számot tekintjük a mintával egyezőnek.

Most már tudjuk, hogy az elején vagy a végén találtuk-e a mintának megfelelő szöveget, de vajon egy köztes szöveget, például szóközök közti szöveget hogyan tudunk úgy megtalálni, hogy magát a szóközöket ne tartalmazza a találat? Erre - egyik megoldásként - alkalmazhatjuk a csoportosítást a mintában:

```
[25] PS C:\> "Ez szóközök közötti" -match "\s(.+?)\s"
True
```

A könyv tanfolyami felhasználása nem engedélyezett!

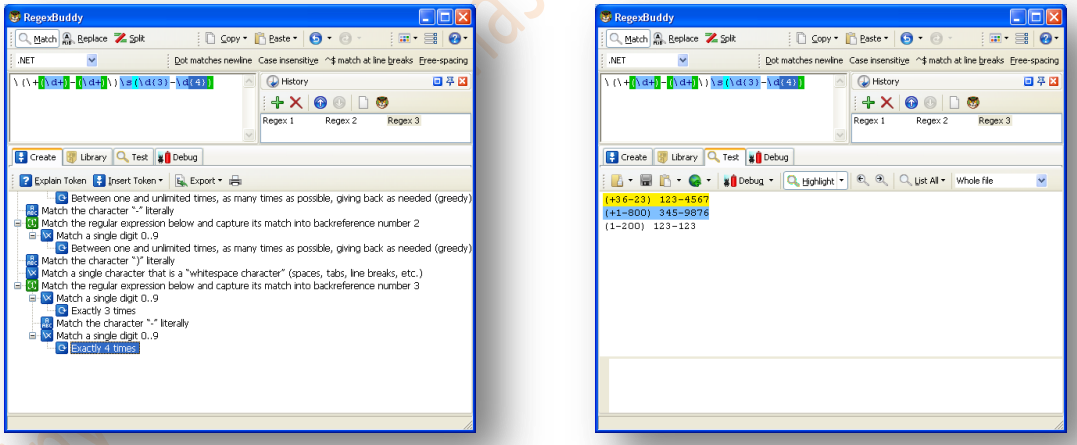
© Soós Tibor – soost@ IQJB.hu

```
[26] PS C:\> $matches

Name                           Value
----                           -----
1                              szóközök
0                               szóközök
```

Itt a mintában található „(\.+?)” rész a gömbölyű zárójelek között egy „lusta” többszörözött (a ? lustítja el a + hatását) szókarakter al-minta a teljes minta részeként, más szóval csoport. A `$matches[0]` továbbra is a teljes találatot tartalmazza, de most már van egy `$matches[1]` is, ami az első találati csoport eredményét tartalmazza. Természetesen több csoportot is elkülöníthetünk, például számjegyek csoportjait:

```
[28] PS C:\> "T.: (+36-1) 123-4567" -match "\(\+(\d+)-(\d+)\)\s(\d{3}-\d{4})"
True
[29] PS C:\> $matches

Name                           Value
----                           -----
3                              123-4567
2                              1
1                              36
0                              (+36-1) 123-4567
```

A fenti példában egy telefonszámból választom ki külön az országkódot, a körzetszámot és a helyi számot. Természetesen egy ilyen mintát összerakni nem egyszerű segédeszköz nélkül. Erre használhatunk olyan eszközöket, mint például a *2.1.12 RegexBuddy* fejezetben említett eszköz:



30. ábra Barátunk megint segít: értelmez és tesztel

Ez a segédprogram a mintánk különböző részeit színkódokkal jelzi, és a „*Create*” fülön részletes magyarázatot is kapunk az általunk beírt mintáról, észre tudjuk venni, hogy esetleg kifelejtettünk egy „escape” karaktert. A „*Test*” fülön meg ellenőrizni tudjuk, hogy megkapjuk-e találatként a helyes szöveget, illetve azt is, hogy a nem jó szöveg tényleg nem találat-e.

Ezeket a csoportokat nevesíthetjük is a jobb azonosíthatóság érdekében:

```
[47] PS C:\> "Telefon: (+36-1) 123-4567" -match "\(\+(?<ország>\d+)-(?<körze
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
t>\d+)\)\s(?<helyi>\d{3}-\d{4})"
True
[31] PS C:\> $matches

Name                           Value
----                           -----
körzet                         1
helyi                          123-4567
ország                         36
0                              (+36-1) 123-4567
```

Tehát a csoportot jelző nyitó zárójel mögé tett „?” és „<>” kacsacsőrök közötti címkével tudjuk nevesíteni a csoportot. Itt nyer értelmet, hogy a `$matches` miért hashtábla, és miért nem egyszerű tömb.

Mi van akkor, ha egy csoportra nincs szükségünk? Erre használhatjuk a „(?:)”, un. „nem rögzülő” csoportot:

```
[36] PS C:\> "123-4567" -match "(?<eleje>\d+)(?:-)(?<vége>\d+)"
True
[37] PS C:\> $matches

Name                           Value
----                           -----
eleje                          123
vége                           4567
0                              123-4567
```

A fenti példában a kötőjelet nem rögzülő csoportba helyeztem, így az nem is szerepel külön találatként, de természetesen a teljes találatban (`$matches[0]`) benne van.

Természetesen a csoportokra is alkalmazhatjuk a „?” többszörözés-jelölőt, ami jelenti ugye a 0 vagy 1 darabot, azaz kezelhetjük azokat a telefonszámokat is, ahol nincs országkód és körzetszám:

```
[42] PS C:\> "Telefon: (+36-1) 123-4567" -match "(\(\+(?<ország>\d+)-
(?<körzet>\d+)\)\s)?(?<helyi>\d{3}-\d{4})"
True
[43] PS C:\> $matches

Name                           Value
----                           -----
körzet                         1
helyi                          123-4567
ország                         36
1                              (+36-1)
0                              (+36-1) 123-4567
```

Ilyenkor – a [42]-es példában látható módon – az egymásba ágyazott csoportok külön találatot adnak a listában, jelen esetben 1-es számmal. Nézzük mi van akkor, ha nincs országkód és körzetszám:

```
[44] PS C:\> "Telefon: 123-4567" -match "(\(\+(?<ország>\d+)-
(?<körzet>\d+)\)\s)?(?<helyi>\d{3}-\d{4})"
True
[45] PS C:\> $matches

Name                           Value
----                           -----
helyi                          123-4567
```

```
0                               123-4567
```

Most térjünk át a `-match` egy nagy korlátjához: csak az első találatig keres. Bár az előző példában használtuk a „?”-et, más többszörözések nem úgy működnek, ahogy szeretnénk. Vegyünk egy olyan mintát, amellyel el szeretnénk különíteni a szavakat, kezdjünk egyszerűség kedvéért három szóval:

```
[76] PS C:\> "egy kettő három" -match "(\w+)\s(\w+)\s(\w+)"
True
[77] PS C:\> $matches

Name                           Value
----                           -----
3                              három
2                              kettő
1                              egy
0                              egy kettő három
```

Meg is kaptuk a várt csoportokat. Vajon egyszerűsíthetjük-e ezt a kifejezést, egyszersmind felkészíthetjük-e a mintánkat változó számú szóra?

```
[84] PS C:\> "egy kettő három " -match "((?<szó>\w+)\W){3}"
True
[85] PS C:\> $matches

Name                           Value
----                           -----
szó                            három
1                              három
0                              egy kettő három
```

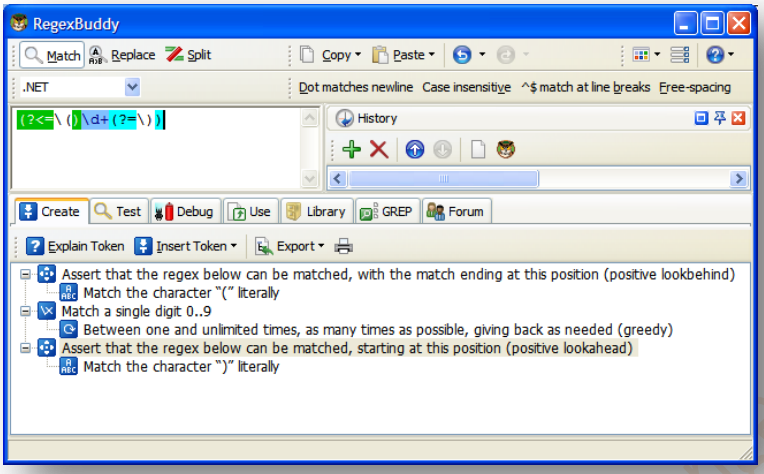
Már a fix számú vizsgálatot se tudtam elérni, a `$matches` nem jegyzi meg egy adott pozíciójú csoport korábbi értékeit. Itt csak a legutolsó szó, a „három” került bele a hashtáblába. Azaz hiába igaz az, hogy csak három szó egyidejű jelenléte teljesíti a feltételt, a végeredménybe csak az utolsó szó kerül bele. Ezt a problémát kicsit később oldom meg.

1.5.6.7 Tekintsünk előre és hátra a mintában

Egyelőre kanyarodjunk oda vissza, hogy a minta helyét valamilyen elválasztó karakter szabja meg. Ha magát a mintát nem akarjuk a találatban viszontlátni, akkor egyik megoldásként alkalmazhatjuk a csoportokat. Egy másik lehetőség a mintában az előretekintés és visszatekintés lehetősége. Például nézzük csak azokat a számokat, amelyek zárójelek között vannak:

```
[86] PS C:\> "sima: 12, nem jó: 11), ez sem: (22, ez jó: (46)" -match
>> "(?<=\()\d+(?=\))"
>>
True
[87] PS I:\>$matches

Name                           Value
----                           -----
0                              46
```

Na, ez megint kezd kicsit hasonlítani az egyiptomi hieroglifákhoz. Nézzük meg, hogy hogyan vizualizálja ezt a RegexBuddy:



31. ábra RegexBuddy értelmezi a mintát

A „(?<=xxx)” szerkezet „visszatekint”, azaz megvizsgálja, hogy az aktuális vizsgálati pozíció előtt az „xxx” minta megtalálható-e. Ha nem, akkor már meg is bukott a vizsgálat. A minta végén található „(?=xxx)” előretekint, azaz megvizsgálja, hogy az aktuális vizsgálati pozíció után az „xxx” minta megtalálható-e. Bár mindkét kitekintő minta zárójeles, de ahogy [87]-ben láthatjuk, nem adnak vissza találatot.

Ezen kitekintő mintáknak is van negatív változatuk, azaz pont azt keressük, hogy a mintánk előtt vagy után ne legyen valamilyen minta:

```
[22] PS I:\>"Árak: Banán:207 Alma:88" -match "(?<!Banán:)\d+"
True
[23] PS I:\>$matches[0]
07
```

Hoppá! Ez nem is jó. Mit is szerettem volna? A „nem banán” gyümölcs árát szerettem volna kiszedni. Azonban a mintám teljesül már a banán áránál is a 2-es után, hiszen a „2” az nem „Banán:”. Azaz ebből az a tanulság, hogy igen gyakran, a negatív kitekintés mellé pozitívot is kell rakni, mert a nem egyezés az nagyon sokféleképpen teljesülhet. A mi esetünkben tehát a jó megoldás:

```
[24] PS I:\>"Árak: Banán:207 Alma:88" -match "(?<!Banán:)(?<=:)\d+"
True
[25] PS I:\>$matches[0]
88
```

Azaz itt most a „pozitív” rész a „:” kettőspont megléte. Ez a két feltétel együttesen már kielégíti a keresési feladatot.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés:

Felmerülhet a tisztelt olvasóban, hogy van-e értelme egykarakteres esetben a negatív körültekintésnek, hiszen használhatnánk negatív karakterosztályt is. Nézzük, hogy mi a különbség a kettő között. Keresem a nem számjeggyel folytatódó „a” betűt:

```
[26] PS C:\> "CAS" -match "a[^\d]"
True
[27] PS C:\> $matches[0]
AS
[28] PS C:\> "CAS" -match "a(?!\d)"
True
[29] PS C:\> $matches[0]
A
[30] PS C:\> "CA" -match "a[^\d]"
False
[31] PS C:\> "CA" -match "a(?!\d)"
True
[32] PS C:\> $matches[0]
A
```

A [27]-es és [29]-es sorokban látható a különbség az eredményben, de ennél súlyosabb a helyzet akkor, ha nincs az „a” betű után „nem számjegy”, hiszen ekkor a negatív karakterosztály fals eredményt ad, mert ő a minta azon helyére mindenképpen akar valamit passzítani, míg a negatív előretekintés esetében a mintának nincs hiányérzete, ha nincs ott semmi már.

1.5.6.8 A mintám visszaköszön

Térjünk vissza a csoportokhoz! Hogyan lehetne felismerni azokat a szavakat, amelyek ugyanolyan elválasztó karakterek között vannak:

```
[16] PS C:\> "-valami-" -match "(\W)\w+\1"
True
[17] PS C:\> "/valami/" -match "(\W)\w+\1"
True
```

Azaz egy csoport eredményét már magában a mintában is felhasználhatom. A csoportra történő hivatkozás formátuma „\1”, „\2”, stb.

Fontos!

Ha egy csoportot ilyen visszahivatkozásban akarjuk szerepeltetni, akkor az a csoport nem lehet „nem rögzülő” csoport, azaz a „?:” jelölést nem használhatjuk.

Természetesen nevesített csoportokra is lehet hivatkozni, ennek formája: „\k<név>”:

```
[28] PS C:\> "A kétszer kétszer rossz" -match "(?<szó>\b\w+\b).+\k<szó>"
True
[29] PS C:\> $matches.szó
kétszer
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5.6.9 Változatok a keresésre

Az eddigi példákban azt láthattuk, hogy a `-match` operátor, hasonlóan a többi szöveges operátorhoz – nem kis-nagybetű érzékeny. De – mint ahogy a többinél is – ennek is van kis-nagybetű érzékeny változata is, a `-cmatch`:

```
[1] PS I:\>"Ebben Kis Nagy van" -cmatch "kis"
False
```

Ha pont a nem egyezőséget akarjuk vizsgálni, arra a `-notmatch` vagy a kis-nagybetű érzéketlen `-cnotmatch` használható:

```
[2] PS I:\>"Nem szeretem a kacsamajmot!" -notmatch "kacsamajom"
True
[3] PS I:\>$matches
[4] PS I:\>
```

Természetesen a `$true` válasz ellenére itt a `$matches` nem ad találatot.

A `-match` még ezt is lehetővé teszi, hogy egy keresési mintán belül váltogassuk a különböző opciókat. Erre a speciális üzemmód jelölő ad lehetőséget:

(?imnsx-imnsx) **(?imnsx-imnsx:)**	**Jelentése** **(a mintára),** **(vagy az adott csoportra vonatkozóan)**
i	kis-nagybetű érzéketlen
m	többsoros üzemmód, a ^ és $ metakarakterek a sorvégekre is illeszkednek
n	„explicit capture”, azaz csak a nevesített csoportok rögzülnek, a névnélküli csoportok nem
s	egysoros üzemmód, a „.” metakarakter illeszkedik a sorvégkarakterre
x	szóközt figyelmen kívül hagyja a mintában, ilyenkor csak a \s-sel hivatkozhatunk a szóközre

A fenti táblázat fejlécében található jelzés egy kis magyarázatra szorul. A `(?imnsx-imnsx)` jelzés, a végén kettőspont nélkül jelzi, hogy ez az egész mintára vonatkozik. Természetesen nem használjuk egyszerre az össze opciót bekapcsolt és kikapcsolt módon. Például:

```
"szöveg" -match "(?im-sx)minta"
```

Ebben a példában az „i” és „m” opció be van kapcsolva, az „s” és az „x” meg ki van kapcsolva.

Ha egy mintán belül többfajta üzemmódot szeretnénk, akkor jön a kettőpontos változata ezen kapcsolók használatának:

```
"szöveg" -match "minta(?-i:M)"
```

A fenti példában csak az „M” betűre, mint részmintára vonatkozik a kis-nagybetű érzéketlenség kikapcsolása.

A könyv tanfolyami felhasználása nem engedélyezett!

Nézzünk ezek használatára gyakorlati példákat!

```
Tegyük kis-nagybetű érzékennyé a vizsgálatot:
[1] PS C:\> "a NAGY betűst keresem" -match "(?-i)NAGY"
True
[2] PS C:\> "a NAGY betűst keresem" -match "(?-i)nagy"
False
```

Kikapcsoltam az érzéketlenséget, azaz kis-nagybetű érzékennyé tettem a vizsgálatot, és csak a nagybetűs minta esetében kaptam találatot.

Nézzük a többsoros szöveg vizsgálatának opcióit:

```
[5] PS C:\> $szöveg="első sor1
>> második sor2"
>>
[6] PS C:\> $szöveg -match "sor\d$"
True
[7] PS C:\> $matches[0]
sor2
[8] PS C:\> $szöveg -match "(?m)sor\d$"
True
[9] PS C:\> $matches[0]
sor1
```

Létrehoztam egy kétsoros szöveget, keresem a sorvégi „sor” szót és egy számjegyet. A két sorvégnél a számmal teszek különbséget, hogy a találatból kiderüljön, hogy melyiket is találtuk meg. A [6]-os sorban nem szóltam a `-match`-nek, hogy lelkileg készüljön fel a többsoros szövegek vizsgálatára, így a sztringvég karakter csak a szöveg végére illeszkedik, így a találatunk a sor2 lett. Ha szólok neki, hogy a szövegem több soros, akkor az első sor vége is találatot ad. A „szólás” a `(?m)` többsoros üzemmód-kapcsolóval történt.

Nézzünk egy példát a nem nevesített csoportok eldobására:

```
[10] PS I:\>"Eleje vége" -match "(?n)(\w+)\s(?<ketto>\w+)"
True
[11] PS I:\>$matches

Name                           Value
----                           -----
ketto                          vége
0                              Eleje vége


[12] PS I:\>"Eleje vége" -match "(?-n)(\w+)\s(?<ketto>\w+)"
True
[13] PS I:\>$matches

Name                           Value
----                           -----
ketto                          vége
1                              Eleje
0                              Eleje vége
```

A fenti példában látható, hogy amikor a [10]-es sorban bekapcsoltam az „n” kapcsolót, akkor külön csoportként nem szerepel az első, név nélküli csoport (de természetesen a teljes találat részét képezi), míg a kikapcsolt, alaphelyzet szerinti működés mellett (12]-es sor) az első, nevesítetlen csoport is létrejött.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nézzük az egysoros kapcsolót. Kicsit zavaró a „többsoros”, „egysoros” elnevezés, hiszen ezek egymás mellett is megférnek, mert más metakarakter működését szabályozzák. Szóval nézzünk egy példát az egysoros üzemmódra:

```
[14] PS C:\> $szöveg="Eleje közepe
>> újsor vége"
>>
[15] PS C:\> $szöveg -match "eleje.+vége"
False
[16] PS C:\> $szöveg -match "(?s)eleje.+vége"
True
```

A [14]-es sorban vizsgálom, hogy illeszkedik-e a kétsoros szövegem olyan mintára, amely egy „eleje” karaktersorozattal kezdődik, és egy „vége” karaktersorozattal végződik. Alapesetben nem kaptam találatot, hiszen a „.+” felakad a sor végén. Míg ha „kierőltetem” az egysoros értelmezést, ahogy az a [16]-os sorban tettem, akkor a „.+” keresztülgázol a sorvégkarakteren is és megkapom a találatomat.

Szóköz mintaként való alkalmazása:

```
[17] PS C:\> "Valami más" -match "(?x)i m"; $matches[0]
False

[18] PS C:\> "Valami más" -match "(?-x)i m"; $matches[0]
True
i m
```

Ebben a példában azt láthatjuk, hogy ha lusták vagyunk a szóköz `\s`-ként való mintában való hivatkozásához, akkor használhatjuk az igazi szóköz karaktert is, ha ezt az üzemmódot bekapcsoljuk. Lehetőség szerint ennek használatát azért kerüljük, mert sok regex változat nem ismeri ezt a lehetőséget, meg a minták olvashatósága sem biztos, hogy a legjobb lesz.

1.5.6.10 Tudjuk, hogy mi, de hányszor?

Az eddigi példákban azt tapasztaltuk, hogy a mintánkat az első találatig keresi a `-match` és változatai. Még a csoportok használatával is csak akkor tudjuk megkeresni a többszörös találatot, ha mi azt a mintát erre eleve felkészítjük, méghozzá meghatározott darabszámmal.

A PowerShell saját `-match` operátorával nem is tudunk többszörös találatot előcsiholni, de szerencsére van erre a cmdlet (lásd következő fejezetet), és – mint ahogy már többször láttuk – a .NET Framework ebben is segítségünkre van:

```
[41] PS C:\> $minta = [regex] "\w+"
```

Itt most a `[regex]` típusjelölővel hozok létre mintát. Nézzük meg ennek tagjellemzőit:

```
[42] PS C:\> $minta | gm


   TypeName: System.Text.RegularExpressions.Regex

Name                 MemberType Definition
----                 ---------- ----------
Equals               Method     System.Boolean Equals(Object obj)
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
GetGroupNames        Method     System.String[] GetGroupNames()
GetGroupNumbers      Method     System.Int32[] GetGroupNumbers()
GetHashCode          Method     System.Int32 GetHashCode()
GetType              Method     System.Type GetType()
get_Options          Method     System.Text.RegularExpressions.RegexOptio...
get_RightToLeft      Method     System.Boolean get_RightToLeft()
GroupNameFromNumber  Method     System.String GroupNameFromNumber(Int32 i)
GroupNumberFromName  Method     System.Int32 GroupNumberFromName(String n...
IsMatch              Method     System.Boolean IsMatch(String input), Sys...
Match                Method     System.Text.RegularExpressions.Match Matc...
Matches              Method     System.Text.RegularExpressions.MatchColle...
Replace              Method     System.String Replace(String input, Strin...
Split                Method     System.String[] Split(String input), Syst...
ToString             Method     System.String ToString()
Options              Property   System.Text.RegularExpressions.RegexOptio...
RightToLeft          Property   System.Boolean RightToLeft {get;}
```

A metódusokat nézve mindent tud, amit eddig láttunk, de van egy sokat ígérő `Matches()` metódusunk is, ami többes számot sejtet (reméljük nem egyes szám harmadik személyt! ☺):

```
[43] PS C:\> $eredmény = $minta.matches("Ez a szöveg több szóból áll")
[44] PS C:\> $eredmény


Groups   : {Ez}
Success  : True
Captures : {Ez}
Index    : 0
Length   : 2
Value    : Ez

Groups   : {a}
Success  : True
Captures : {a}
Index    : 3
Length   : 1
Value    : a

Groups   : {szöveg}
Success  : True
Captures : {szöveg}
Index    : 5
Length   : 6
Value    : szöveg

Groups   : {több}
Success  : True
Captures : {több}
Index    : 12
Length   : 4
Value    : több

Groups   : {szóból}
Success  : True
Captures : {szóból}
Index    : 17
Length   : 6
Value    : szóból

Groups   : {áll}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Success  : True
Captures : {áll}
Index    : 24
Length   : 3
Value    : áll
```

Egészen érdekes és sokat sejtető eredményt kaptunk. Nézzük meg a `get-member` cmdlettel, hogy ez mi:

```
[45] PS C:\> Get-Member -InputObject $eredmény


   TypeName: System.Text.RegularExpressions.MatchCollection

Name                MemberType            Definition
----                ----------            ----------
CopyTo              Method                System.Void CopyTo(Array array,...
Equals              Method                System.Boolean Equals(Object obj)
GetEnumerator       Method                System.Collections.IEnumerator ...
GetHashCode         Method                System.Int32 GetHashCode()
GetType             Method                System.Type GetType()
get_Count           Method                System.Int32 get_Count()
get_IsReadOnly      Method                System.Boolean get_IsReadOnly()
get_IsSynchronized  Method                System.Boolean get_IsSynchroniz...
get_Item            Method                System.Text.RegularExpressions....
get_SyncRoot        Method                System.Object get_SyncRoot()
ToString            Method                System.String ToString()
Item                ParameterizedProperty System.Text.RegularExpressions....
Count               Property              System.Int32 Count {get;}
IsReadOnly          Property              System.Boolean IsReadOnly {get;}
IsSynchronized      Property              System.Boolean IsSynchronized {...
SyncRoot            Property              System.Object SyncRoot {get;}
```

Ez pont az, amit szerettünk volna: egy tömb, ami tartalmazza az összes találatot! Hogyan lehet ezt az eredményt kezelni? Például egy ciklussal kiírathatom az egyes találati elemeket:

```
[46] PS C:\> foreach($elem in $eredmény){$elem.value}
Ez
a
szöveg
több
szóból
áll
```

Megjegyzés:

Vigyázzunk, hogy a [regex] típusnak van egy `match()` metódusa is, de az szintén csak egy találatod ad vissza, hasonlóan a `-match` operátorhoz.

Ahogy a [43]-as sor kimenetében láttuk, a `[regex]` objektumok rendelkeznek `replace()` metódussal is, de van `split()` is, amellyel a minta találati helyeinél lehet feltördelni a szöveget, amelyre alkalmazzuk. Nézzünk erre egy gondolatébresztő példát:

```
[49] PS C:\> $minta = [regex] '\W+'
[50] PS C:\> $minta.split("Ebben {van}, minden-féle (elválasztó) [jel]")
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Ebben
van
minden
féle
elválasztó
jel
```

Ebben a példában feldaraboltam a szövegemet mindenféle szóelválasztó karakternél. Ugye ugyanezt az eredményt nem tudtam volna elérni a sztringek `split()` metódusával:

```
[51] PS C:\> ("Ebben {van}, minden-féle (elválasztó) [jel]").split()
Ebben
{van},
minden-féle
(elválasztó)
[jel]
```

1.5.6.11 Regexre cmdlettel (Select-String)

A PowerShell 2.0-ban a több találatot is megjelenítő regex kifejezéseket már a `Select-String` cmdlet is tudja! Nézzük, hogy hogyan:

```
[1] PS C:\> "ablak", "ajtó", "ház" | Select-String -Pattern "a"

ablak
ajtó
```

Első ránézésre úgy néz ki, mintha egyszerűen csak kiválogatta volna azokat a sztringeket, amelyekben szerepel az „a” minta. De nézzük kicsit részletesebben az eredményt:

```
[2] PS C:\> "ablak", "ajtó", "ház" | Select-String -Pattern "a" | Format-List


IgnoreCase : True
LineNumber : 1
Line       : ablak
Filename   : InputStream
Path       : InputStream
Pattern    : a
Context    :
Matches    : {a}

IgnoreCase : True
LineNumber : 2
Line       : ajtó
Filename   : InputStream
Path       : InputStream
Pattern    : a
Context    :
Matches    : {a}
```

Látható, hogy a `Matches` tulajdonságban megtalálható egyelőre az első találat. Ha az összesre kíváncsiak vagyunk, akkor használjuk az `-AllMatches` kapcsolót:

```
[8] PS C:\> "ablak", "ajtó", "ház" | Select-String -Pattern "a" -AllMatches | F
```

```
ormat-List


IgnoreCase : True
LineNumber : 1
Line       : ablak
Filename   : InputStream
Path       : InputStream
Pattern    : a
Context    :
Matches    : {a, a}

IgnoreCase : True
LineNumber : 2
Line       : ajtó
Filename   : InputStream
Path       : InputStream
Pattern    : a
Context    :
Matches    : {a}
```

Látható, hogy így már az első illeszkedést kielégítő sztringnél a `Matches` tulajdonságban minden egyező találat megtalálható. A `Select-String` közvetlenül tud szövegfájlokban is keresni, ezt a képességét majd a gyakorlati rész *2.5.4 Szövegfájlok feldolgozása (Get-Content, Select-String)* fejezetben lesz szó.

1.5.6.12 Csere

Sok esetben azért keresünk egy mintát, hogy az illeszkedő részeket kicseréljük valami másra. Ez a valami más akár lehet egy találati csoport is. Például cseréljük ellenkező sorrendűre egy IP címben szereplő számokat:

```
[64] PS C:\> "Ezt kellene megfordítani: 192.168.1.2 reverse zónához" -replac
e "(\d{1,3})\.(\d{1,3})\.(\d{1,3}).(\d{1,3})", '$4.$3.$2.$1'
Ezt kellene megfordítani: 2.1.168.192 reverse zónához
```

Fontos!

A `-replace` operátornál a minta és a csereszabály között vessző van és a csereszabály nem macskakörmök, hanem aposztrófok között kell hogy álljon, egyébként a csoporthivatkozásokat `($1, $2,...)` a PowerShell változóként értékelné, holott ezek nem változók, hanem a `[regex]` formai elemei!

Látjuk, hogy itt a mintacsoportok résztalálataira `$1, $2,...` formában tudunk hivatkozni, az indexek nem nullától indulnak, hanem 1-től. A `-replace`-en kívül nem lehet hozzáférni a `$1` és a többi változóhoz, és a `-replace` által végrehajtott illeszkedésvizsgálat eredménye sem olvasható ki, nem keletkezik `$matches` változó sem.

A `-replace` szerencsére már alaphelyzetben a mintának megfelelő összes szövegrészt kicseréli:

```
[65] PS C:\> "Szóköz kicsrélése aláhúzásra" -replace "\s", '_'
Szóköz_kicsrélése_aláhúzásra
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Azonban ez nem rekurzív, azaz ha a csere után is marad cserélni való, akkor arra nekünk kell újra meghívni a `-replace`-t:

```
[66] PS C:\> "Felesleges     szóközök    eltávolítása" -replace "\s\s", ' '
Felesleges   szóközök  eltávolítása
```

A fenti példában látszik, hogy a sok egymás melletti szóközt szeretném egyre cserélni azáltal, hogy kettőből csinálok egyet. A fenti módszerrel ez nem működik, hiszen csak egyszer szalad végig, a megoldásban még bőven marad több egymás melletti szóköz. Erre egy jobb minta alkalmazása a megoldás:

```
[67] PS C:\> "Felesleges    szóközök    eltávolítása" -replace "\s{2,}", ' '
Felesleges szóközök eltávolítása
```

Mint ahogy a `-match`-nek is, a `-replace`-nek is van kis-nagybetű érzékeny változata, a `-creplace`:

```
[68] PS C:\> "Csak a nagy 'A'-t cserélem" -creplace "A", 'a'
Csak a nagy 'a'-t cserélem
```

A `-replace` még tud néhány érdekességet:

```
[56] PS C:\> "egy kettő három" -replace "kettő", '$`'
egy egy  három
```

A `$`` szimbólum jelképezi a találatig terjedő szövegrészt.

```
[58] PS C:\> "egy kettő három" -replace "kettő", "$'"
egy  három három
```

A `$'` szimbólum jelképezi a találat utáni szövegrészt. Vigyázni kell, hogy itt a PowerShell idézőjel-kezelési szabályai miatt vagy macskakörmöt (”) használunk a jel körül, vagy duplázzuk az aposztrófot: `'$'''`.

```
[61] PS C:\> "egy kettő három" -replace "kettő", '$_'
egy egy kettő három három
```

A `$_` szimbólum a teljes mintát szimbolizálja. Itt megint ügyelni kell arra, hogy a PowerShellben a `$_` változót szimbolizál, így a regex számára szó szerint ezt úgy tudjuk átadni, ha aposztrófot használunk.

```
[63] PS C:\> "egy kettő három" -replace "kettő", '$+$+'
egy kettőkettő három
```

A `$+` szimbólummal a találatot szimbolizálhatjuk, így a fenti példában duplikálni tudtam a „kettő”-t.

1.5.7 Logikai és bitszintű operátorok

Logikai operátorok segítségével (`-and`, `-or`, `-xor`) össze tudunk fűzni több logikai kifejezést, és a szokásos „igazság táblák” alapján kapjuk meg az összetett kifejezésünk igaz vagy hamis értékét:

```
[24] PS I:\>$false -eq $false -and $true -eq $true
True
[25] PS I:\>"ablak" -ne "ajtó" -or 5 -eq 6
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
True
[26] PS I:\>1 -eq 1 -xor 2 -gt 1
False
[27] PS I:\>-not $true
False
[28] PS C:\> !$true
False
```

A [28]-ban látható, hogy az egytagú `-not` operátor negálja az eredményt. A `-not` helyett használhatjuk a `!` (felkiáltójel) jelölést is.

Kényelmi szolgáltatásként nem kell `$true` vagy `$false`-ra redukálni a logikai operátorok operandusait, a triviális leképezéseket elvégzi helyettünk a PowerShell. Általában minden 0-tól, `$null`-tól, vagy üres kifejezéstől eltérő értékeket `$true`-nak kezel:

```
[29] PS I:\>7 -and $true
True
[30] PS I:\>"valami" -and $true
True
[31] PS I:\>$null -and $true
False
[32] PS I:\>"" -and $true
False
[33] PS I:\>0 -and $true
False
```

Megjegyzés

Az `-and`, `-or` használatakor vigyázni kell, mert nem értékelődik ki a jobboldaluk, ha már a baloldal alapján el tudja dönteni a végeredményt:

```
[35] PS I:\>$false -and ($a=25)
False
[36] PS I:\>$a
[37] PS I:\>($a=25) -and $false
False
[38] PS I:\>$a
25
[39] PS I:\>$true -or ($b="valami")
True
[40] PS I:\>$b
[41] PS I:\>($b="valami") -or $true
True
[42] PS I:\>$b
valami
```

Tehát nem érdemes egyéb műveleteket végeztetni a logikai operátorok operandusaiban, mert ha nem kellő figyelemmel választjuk ki az oldalakat, akkor esetleg nem úgy működik a programunk, ahogy vártuk.

A fenti logikai operátorok tehát vagy `$false`, vagy `$true` értéket adnak vissza. Ha „igazi” bit szintű logikai műveleteket akarunk elvégezni, akkor ehhez külön logikai operátorok vannak: `-band`, `-bor`, `-bxor`, `-bnot`:

```
[44] PS I:\>126 -bor 1
127
[45] PS I:\>(128+8+4+1) -bor 132
141
[46] PS I:\>(128+8+4+1) -band (128+16+8)
136
[47] PS I:\>(128+8+4+1) -bxor (128+16+8)
21
[48] PS I:\>-bnot 255
-256
[49] PS I:\>-bnot 1
-2
```

Itt természetesen mindig kiértékelődik a művelet mindkét oldala.

1.5.8 Típusvizsgálat, típuskonverzió (-is, -isnot, -as)

Az 1.4.8 Típuskonverzió fejezetben variáltam a típusokkal. Ebbe a témakörbe tartozik a típusok vizsgálata is. Ha nem lekérni akarjuk egy objektum típusát, hanem rákérdezni, hogy valami valamilyen típusú-e, akkor az `-is` operátort lehet használni, ha pont arra vagyunk kíváncsiak, hogy egy változó nem valamilyen-e, akkor az `-isnot` operátor használható:

```
[52] PS I:\>"szöveg" -is [string]
True
[53] PS I:\>"szöveg" -is [object]
True
[54] PS I:\>"szöveg" -is [array]
False
[55] PS I:\>1 -is [double]
False
[56] PS I:\>1.1 -is [double]
True
[57] PS I:\>1.1 -is [float]
False
[58] PS I:\>$true -is [int]
False
[59] PS I:\>$true -is [bool]
True
[60] PS I:\> (get-date) -isnot [int]
True
```

Minden „dolog” a PowerShellben objektum, így akár a szöveg, vagy a szám is, de akár a típusjelölő is objektum:

```
[64] PS I:\>[string] -is [object]
True
```

És egy majdnem[8] örök igazság:

```
[70] PS I:\>$a -is $a.GetType()
True
```

[8] kivéve az `$a=$null` esetet

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A már korábban látott típuskonverziós szintaxis mellett használhatunk direkt típuskonvertáló operátort is, ami az `-as`:

```
[73] PS I:\>"01234" -as [int]
1234
[74] PS I:\>1.1234 -as [int]
1
[76] PS I:\>1.1234 -as [double]
1,1234
[77] PS I:\>112345678 -as [float]
1,123457E+08
[78] PS I:\>112345678 -as [double]
112345678
[79] PS I:\>"szöveg" -as [int]
[80] PS I:\>
```

A [79]-es sornál látszik, hogy mi a különbség a korábbi típuskonverziós szintaxis és az `-as` operátor használata között: ez utóbbinál, ha nem tudja végrehajtani a műveletet, akkor nem ad hibajelzést, hanem `$null` értéket ad vissza.

1.5.9 Egytagú operátorok (+, -, ++, --, [típus])

A + jel, ha csak mögötte van objektum, egyszerűen számmá konvertáló operátorként is felfogható, a – jel ugyancsak számmá konvertál, de még negál is:

```
[81] PS I:\>-"01234"
-1234
[82] PS I:\>+"0123"
123
```

A duplázott ++ és –– jel meg inkrementálást és dekrementálást végez, csakhogy két lehetőségünk is van ezek elhelyezésére:

```
[88] PS I:\>$a = 10
[89] PS I:\>$b = $a++
[90] PS I:\>$b
10
[91] PS I:\>$a
11
[92] PS I:\>$c = 10
[93] PS I:\>$d = --$c
[94] PS I:\>$d
9
[95] PS I:\>$c
9
```

A [89]-es sorban előbb kapja meg `$b` az `$a` értékét ahhoz képest, hogy `$a`-t megnöveltem, a [93]-ban pedig előbb csökkentettem `$c`-t, és már ezt a csökkentett értéket vette át a `$d`.

Az 1.4.8 Típuskonverzió fejezetben már láthattuk, hogy a `[típusnév]` is egy egytagú operátor, ez a típuskonverziós operátor.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5.10 Csoportosító operátorok

Sokfajta zárójelet használ a PowerShell, szedjük ezeket össze most ide egy helyre.

1.5.10.1 Gömbölyű zárójel: ()

Elsőként nézzük a gömbölyű zárójelet:

Zárójel	Használat, magyarázat
()	Egyszerű kifejezések kifejtése, beleértve az egyszerű kifejezéseket tartalmazó csővezetékeket „Elfojtott” visszatérési értékű kifejezések visszatérési értékének megjeleníttetése

Példák:

```
[1] PS C:\>(1+2)*4
12
[2] PS C:\>(Get-Location).Drive

Name        Provider      Root                                    CurrentLocatio
                                                                              n
----        --------      ----                                    --------------
C           FileSystem    C:\


[3] PS C:\>(get-childitem).count
42
[4] PS C:\>(1,2,3,-5,4,-6,0,-11 | where-object {$_ -lt 0}).count
3
```

Az [1]-es sor egyértelmű, meghatározom, hogy mely műveletet végezze előbb. A [2]-es sorban előbb kell elvégeztetni a `Get-Location` parancsot, hogy az ő kimenetének megnézhessem a `Drive` tulajdonságát. A [3]-as sorban a `Get-ChildItem` cmdlettel előbb létre kell hozni azt a gyűjteményt, aminek aztán megnézhetjük a számosságát. A [4]-es sorban látható, hogy a zárójelek között a csővezetékkel mindenféle műveletet is végezhetünk, például beirányítani egy következő cmdletbe.

Van még egy érdekes jelenség a PowerShellben, nézzünk egy egyszerű értékadást:

```
[21] PS I:\>$a = 1111
[22] PS I:\>
```

Nem ad semmilyen visszatérési értéket. A PowerShell alkotói úgy vélték, hogy ez a gyakoribb igény, azaz nem akarjuk viszontlátni az éppen most átadott értéket. Ezek az un. „voidable” kifejezések, amelyek esetében el lehet rejteni a visszatérési értéket. Ettől függetlenül, ha akarjuk, akkor meg lehet jeleníttetni a visszatérési értéket:

```
[28] PS I:\>($a = 1111)
1111
```

Hasonló módon, a növelés és csökkentés operátora sem ad alaphelyzetben kimenetet, de kicsikarhatjuk ezt is:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[63] PS C:\> $i=1
[64] PS C:\> $i++
[65] PS C:\> ($i++)
2
[66] PS C:\> $i
3
```

1.5.10.2 Dolláros gömbölyű zárójel: $()

Nézzük a következő csoportosítási lehetőséget!

Zárójel	Használat, magyarázat
$()	Több kifejezés szeparálása. Idézőjelek közti kifejezés kifejtése.

Példák:

```
[3] PS C:\> ($a ="ab"; $b="lak"; $a+$b).length
Missing closing ')' in expression.
At line:1 char:10
+ ($a ="ab" <<<< ; $b="lak"; $a+$b).length
    + CategoryInfo          : ParserError: (CloseParenToken:TokenId) [], Pare
   ntContainsErrorRecordException
    + FullyQualifiedErrorId : MissingEndParenthesisInExpression
```

Ha a sima zárójelbe több kifejezést akarnánk beírni, akkor hibát kapunk, mint ahogy a [3]-as sor után látható. Ha ezt nem szeretnénk, akkor a `$()` változatot kell használni:

```
[4] PS I:\>$($a ="ab"; $b="lak"; $a+$b).length
5
```

Megjegyzés:

Itt, a [4]-es sorban egyébként pont jól jön nekünk az, hogy az értékadás nem ad alaphelyzetben kimenetet, hiszen így csak az `$a+$b`-re vonatkozóan kapjuk meg a hosszt, ami pont kell nekünk. Kényszerítsük például az első értékadást arra, hogy legyen kimenete:

```
[30] PS I:\>$(($a ="ab"); $b="lak"; $a+$b).length
2
```

Itt 2-t kaptunk kimenet gyanánt, merthogy ez már egy kételemű tömb, nem pedig egy valamekkora hosszúságú sztring.

Már korábban is tapasztaltuk ennek a speciális zárójelezésnek a jótékony hatását az idézőjeleknél:

```
[6] PS I:\>$g = "szöveg"
[7] PS I:\>"Ez a `$g hossza: ($g.length)"
Ez a $g hossza: (szöveg.length)
[8] PS I:\>"Ez a `$g hossza: $($g.length)"
Ez a $g hossza: 6
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ugye itt a [7]-ben nem tettem `$` jelet, csak sima zárójelpárt, az idézőjel miatt csak a `$g` fejtődött ki, ezzel szemben a [8]-ban az egész kifejezés kiértékelődött.

1.5.10.3 Kukacos gömbölyű zárójel: @()

Zárójel	Használat, magyarázat
@()	Tömbkimenetet garantál.

A harmadik zárójelezés típus azt biztosítja, hogy mindenképpen tömb legyen a zárójelek közti művelet eredménye, ha csak egy elem amúgy a kimenet, akkor ezzel egy egyelemű tömböt fogunk kapni. Miért jó ez? Nézzünk erre egy példát:

```
[32] PS I:\>$tömb = "egy", "kettő", "három"
[33] PS I:\>$tömb.length
3
[34] PS I:\>$tömb = "egy", "kettő"
[35] PS I:\>$tömb.length
2
[36] PS I:\>$tömb = "egy"
[37] PS I:\>$tömb.length
3
```

Van egy `$tömb` változónk, amely változó számú elemet tartalmaz. Szükségünk van az elemszámra valamilyen művelet elvégzéséhez, ezért lekérjük a `Length` tulajdonságát. A [33]-ban és a [35]-ben jó eredményt kapunk, a [37]-ben, amikor már csak egyelemű a tömböm, akkor váratlanul nem 1-et, hanem 3-at kapok! Vajon miért? Mert ekkor „hirtelen” a `$tömb`-öm már nem is tömb, hanem már egy egyszerű sztring. Mivel ennek is van `Length` tulajdonsága, ezért nem kapunk hibát, viszont már más az egésznek a jelentése. Ha nem akarunk ilyen jellegű „hibát”, akkor rákényszeríthetjük a kifejezésünket arra, hogy mindenképpen tömböt adjon vissza:

```
[38] PS I:\>$tömb = "egy", "kettő", "három"
[39] PS I:\>@($tömb).length
3
[40] PS I:\>$tömb = "egy", "kettő"
[41] PS I:\>@($tömb).length
2
[42] PS I:\>$tömb = "egy"
[43] PS I:\>@($tömb).length
1
```

Így már teljesen konzisztens a megoldásunk.

1.5.10.4 Kapcsos zárójel: {} (bajusz)

A következő zárójeltípus a kapcsos zárójel (vagy bajusz):

Zárójel	Használat, magyarázat
{}	Szkriptblokk

Ezzel kapcsolatosan lesz egy külön fejezet, de elöljáróban annyit, hogy kódrészleteket tudunk ezzel csoportosítani, illetve ténylegesen kódként kezelni:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[46] PS C:\scripts> $script = {$dátum = get-date; $dátum}
[47] PS C:\scripts> $script
$dátum = get-date; $dátum
[48] PS C:\scripts> $script | Get-Member


   TypeName: System.Management.Automation.ScriptBlock

Name             MemberType Definition
----             ---------- ----------
Equals           Method     System.Boolean Equals(Object obj)
GetHashCode      Method     System.Int32 GetHashCode()
GetType          Method     System.Type GetType()
get_IsFilter     Method     System.Boolean get_IsFilter()
Invoke           Method     System.Collections.ObjectModel.Collection`1[...
InvokeReturnAsIs Method     System.Object InvokeReturnAsIs(Params Object...
set_IsFilter     Method     System.Void set_IsFilter(Boolean value)
ToString         Method     System.String ToString()
IsFilter         Property   System.Boolean IsFilter {get;set;}


[49] PS C:\scripts> $script.invoke()

2008. április 22. 23:33:35
```

A [46]-ban úgy néz ki, mintha a `$()`-t használtuk volna, de a [47]-ben kiderült, hogy mégsem ugyanaz a helyzet, hiszen a `$script`-változónk nem a zárójeles kifejezés értékét kapta meg, hanem magát a kifejezést. De mégsem sztringgel van dolgunk, hiszen a `Get-Member`-rel nem a sztringekre jellemző tagjellemzőket kapjuk meg, hanem a típusnál is látjuk, hogy ez egy `ScriptBlock`. Ennek fő jellemzője, hogy van neki egy `invoke` metódusa, amellyel végre is lehet hajtatni, mint ahogy a [49]-ben látjuk.

1.5.10.5 Dolláros kapcsos zárójel: ${}

Zárójel	Használat, magyarázat
${}	Változó nevének operátora Psmeghajtó elemének kiolvasása

Ezt már mutattam, de azért ismétlésképpen szerepeljen itt újra. Ezzel a jelöléssel lehet bonyolult nevű változókat is használni:

```
[10] PS C:\> ${ilyen név nem lehetne} = 1
[11] PS C:\> ${ilyen név nem lehetne}
1
```

Ugyanezt a jelölésrendszert tudjuk használni különböző PSmeghajtók tartalmának kiolvasásához is:

```
[19] PS C:\> ${function:cd..}
Set-Location ..
```

Ez a kifejezés is hibát adna a kapcsos zárójelek nélkül.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5.10.6 Szögletes zárójel: []

Az utolsó zárójeltípus ugyan nem csoportosít, de ha már itt vannak a zárójelek, akkor tegyük ezt is ide:

Zárójel	Használat, magyarázat
[]	Tömbindex
	Típusoperátor

Példák:

```
[5] PS I:\>(1,2,3,4,5)[2]
3
[6] PS I:\>(1,2,3,4,5)[0]
1
[7] PS I:\>[int] "00011"
11
[8] PS I:\>[datetime]::now

2008. március 14. 10:28:39
```

Az [5]-ös és [6]-os sorokban tömbindex szerepben látjuk a szögletes zárójelet, a [7]-es és [8]-as sorban típusjelöléshez használtam.

1.5.11 Tömbelem-operátor: „,”

Az előző részben volt szó a `@()` csoportosító operátorról, amit különösen akkor tudunk kihasználni, ha az egyelemű tömböket is tömbként akarjuk kezelni. Erre a célra egy másik operátort, a tömboperátort (`,`) is felhasználhatjuk:

```
[11] PS I:\>(,"szöveg").GetType().Fullname
System.Object[]
[12] PS I:\>(,1).GetType().Fullname
System.Object[]
[13] PS I:\>(@(1)).GetType().Fullname
System.Object[]
[14] PS I:\>(,(1,2,3,4)).count
1
[15] PS I:\>(@(1,2,3,4)).count
4
```

A [12]-es és [13]-as sorokban látszik, hogy egyelemű tömbök esetében teljesen ugyanúgy működik a kétfajta operátor. Azonban, ha már többelemű tömböt kezelünk, akkor nagy különbséget láthatunk. A tömboperátor (,) használatával egy egyelemű tömböt kapunk a [14]-es sorban, ahol ez az egy elem egy tömb. A [15]-ös sorban pedig visszakapjuk az eredeti tömbünket.

1.5.12 Tartomány-operátor: „..”

Viszonylag gyakran kell számsorozatokkal dolgoznunk. Ennek megkönnyítésére van egy nagyon praktikus operátor:

```
[17] PS I:\>1..5
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
1
2
3
4
5
[18] PS I:\>$a=3
[19] PS I:\>1..$a
1
2
3
```

A [19]-es sorban látható, hogy nem csak statikus lehet egy ilyen tartomány vége, hanem változók is lehetnek benne, így még szélesebb körű a felhasználhatósága:

```
[20] PS I:\>$s = "hétfő", "kedd", "szerda", "csütörtök", "péntek", "szombat",
"vasárnap"
[21] PS I:\>$n1 = 2
[22] PS I:\>$n2 = 4
[23] PS I:\>$s[$n1..$n2]
szerda
csütörtök
péntek
```

A fenti példában például tömbök indexelésére használtam.

Vagy lehet fordított sorrendet is kérni, illetve negatív tartományba is lehet menni:

```
[26] PS I:\>3..-2
3
2
1
0
-1
-2
```

Ez az operátor csak egész számokkal működik. Ha valami egyebet (tört szám, sztring formátumban szám) adunk be, akkor a PowerShell automatikusan egésszé konvertálja.

1.5.13 Tulajdonság, metódus és statikus metódus operátora: „.”, „::”

Ilyet is már mutattam sokszor, a pontról (`.`)-ról és a kettőspontról (`::`)-ról van szó:

```
[42] PS I:\>[string]::compareordinal("ac", "ab")
1
[43] PS I:\>$a = new-object random
[44] PS I:\>$a.Next()
1418203324
[45] PS I:\>(get-date).Year
2008
```

A [42]-ben statikus metódusra hivatkoztam a (`::`)-tal, [44]-ben metódusra, [45]-ben tulajdonságra hivatkoztam a (`.`) segítségével.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5.14 Végrehajtás

Úgy látszik, szűkösen állunk írásjelekkel, mert a (.)-nak van egy másik jelentése. Amikor egytagú operátorként használjuk, akkor végrehajtási operátor a funkciója, azaz az operandusát végrehajtandó kódként tekinti és végrehajtja azt:

```
[55] PS I:\>$a= {get-date}
[56] PS I:\>.$a

2008. március 14. 14:11:14

[57] PS C:\> $a= "get-date"
[58] PS C:\> .$a

2008. március 14. 14:12:21
```

A (.)-hoz hasonlóan az (&) jel is végrehajt:

```
[59] PS C:\scripts> &$a

2008. március 14. 14:13:51
```

Természetesen a (.) és a (&), mint végrehajtási operátor paraméterként valamilyen futtatható objektumot várnak: cmdletet, függvényt, szkriptet vagy szkriptblokkot. Akármilyen, számunkra futtathatónak tűnő sztring nem jó nekik:

```
[60] PS I:\>$a = "(1+2)"
[61] PS I:\>.$a
The term '(1+2)' is not recognized as a cmdlet, function, operable program, or
script file. Verify the term and try again.
At line:1 char:2
+ .$ <<<< a
```

Ha mégis ilyen sztringet akarunk végrehajtatni, akkor van szerencsére erre egy PowerShell cmdletünk, az `Invoke-Expression`:

```
[62] PS I:\>Invoke-Expression "1+2"
3
```

Van egy másik végrehajtó cmdlet, `invoke-item`:

```
[63] PS C:\> Invoke-Item C:\munka\szöveg.txt
```

Ennek paramétereként egy olyan fájlt kell megadni, amelyikhez tartozik hozzárendelt alkalmazás, és a futtatásának hatására elindítja az alkalmazást és betölteti vele a fájlt. Gyakorlatilag ugyanazt csinálja, mintha egérrel duplán kattintanánk a fájlon. Az `invoke-expression` is megteszi ezt, viszont a [62]-ben látott példa is mutatja, hogy ennél ez többet is tud, így bizonyos esetekben biztonságosabb az `Invoke-Item` használata.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.5.15 Formázó operátor

Láttuk az *1.3.15 Kimenet (Output)* fejezetben, hogy a `write-host` cmdlet alkalmas arra, hogy színesen írjunk a képernyőre, de hogyan lehet mondjuk egy szép táblázatot kiíratni?

Az alábbi kis szkriptben – felhasználva a korábban már bemutatott tartomány-operátort – azt szeretném, hogy a számokat írja ki a PowerShell a nevükkel együtt szép, táblázatszerű módon. A [24]-es promptban készítek egy tömböt, ami tartalmazza a számokat egytől tízig, a [26]-os promptban pedig a korábban bemutatott tartomány-operátor segítségével kiírom a számokat és a nevüket.

```
[24] PS C:\> $tomb= "egy", "kettő", "három", "négy", "öt", "hat", "hét",
"nyolc", "kilenc", "tíz"
[25] PS C:\> $tomb
egy
kettő
három
négy
öt
hat
hét
nyolc
kilenc
tíz
[26] PS I:\>0..9 | foreach-object {($_+1), $tomb[$_]}
1
egy
2
kettő
3
három
4
négy
5
öt
6
hat
7
hét
8
nyolc
9
kilenc
10
tíz
```

Látjuk, hogy ez nem túl szép kimenet, hiszen egymás alatt vannak a számok és neveik, jó lenne, ha ezek egymás mellett lennének. Ugyan használhatnánk a `write-host` cmdletet a `-nonewline` kapcsolóval, vagy az `$OFS` változót, de ennél van egyszerűbb megoldás: a `-f` formázó operátor:

```
[27] PS I:\>0..9 | foreach-object {'{0} {1}' -f ($_+1), $tomb[$_]}
1 egy
2 kettő
3 három
4 négy
5 öt
6 hat
7 hét
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
8 nyolc
9 kilenc
10 tíz
```

Ennek a működése a következő:

A `-f` előtt kell a formátumot leíró sztringet beírni, a `-f` után meg a formázandó kifejezést. Mivel ez utóbbiból több is lehet (a példámban a `($_+1)` és a `$tomb[$_]`), ezért a formázó sztringben is mindkét elemre hivatkozni kell. A hivatkozás formája pedig {0} az első formázandó kifejezést szimbolizálja, a {1} a másodikat és így tovább.

De nem csak egyszerűen hivatkozhatunk a formázandó kifejezésekre, hanem sokkal bonyolultabb műveleteket is végezhetünk velük:

```
[1] PS C:\> "Szöveg elöl {0,-15} és nem hátul" -f "itt van"
Szöveg elöl itt van         és nem hátul
[2] PS C:\> "Szöveg nem elöl {0,15} hanem hátul" -f "van"
Szöveg nem elöl             van hanem hátul
[3] PS C:\> "Ez most az óra: {0:hh}" -f (get-date)
Ez most az óra: 10
[4] PS C:\> "Ez most a forint: {0:c}" -f 11
Ez most a forint: 11,00 Ft
[5] PS C:\> "Ez most a szép szám: {0:n}" -f 1234568.9
Ez most a szép szám: 1 234 568,90
[6] PS C:\> "Ez most a százalék: {0:p}" -f 0.561
Ez most a százalék: 56,10 %
[7] PS C:\> "Ez most a hexa szám: {0:x}" -f 3000
Ez most a hexa szám: bb8
[8] PS C:\> "Ez most a 8 számjegyű hexa szám: {0:x8}" -f 3000
Ez most a 8 számjegyű hexa szám: 00000bb8
[9] PS C:\> "Ez most a forint, fillér nélkül: {0:c0}" -f 11
Ez most a forint, fillér nélkül: 11 Ft
[10] PS C:\> "Ez most rövid dátum: {0:yyyy. M. d}" -f (get-date)
Ez most rövid dátum: 2008. 4. 18
[11] PS C:\> "Ez most hosszú dátum: {0:f}" -f (get-date)
Ez most hosszú dátum: 2008. április 18. 23:10
```

A szélesség és megjelenítés szabályozóit vegyíthetjük is. A következő példában véletlen számokat írok ki szépen két tizedessel, jobbra igazítva[9]:

```
[12] PS C:\> 1..5 | foreach-object {"Szép szám: {0,8:n2}!" -f (get-random -Maxi
mum 120.0001)}
Szép szám:    83,50!
Szép szám:   119,30!
Szép szám:    85,46!
Szép szám:    67,46!
Szép szám:   100,88!
```

Aztán számokkal lehet akármilyen mintázatot is kirakni:

[9] A felfedezésért köszönet Oláh Istvánnak!

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[22] PS C:\> "Ez most a telefonszám: {0:(##) ###-####}" -f 303116867
Ez most a telefonszám: (30) 311-6867
```

Megjegyzés:

A legtöbb formázási művelet elvégezhető a `ToString` metódussal (ami nagyon sok típusnál megtalálható):

```
[23] PS C:\> (303116867).ToString("(##) ###-####")
(30) 311-6867
```

Fontos hangsúlyozni, hogy ezeknek a formázási műveleteknek a kimenete mindig sztring, azaz a formázás után elveszítik az eredeti tulajdonságaikat. Azaz csak egy műveletsor legvégén érdemes formázni, amikor már csak a megjelenítés van hátra, egyéb feldolgozási műveleteken már túljutottunk. Amúgy még sokkal több formázási lehetőség van, javaslom az MSDN honlapján utánanézni.

Visszatérve a számok és neveik kiírásához, még szebben rendezve:

```
[24] PS C:\> 0..9 | foreach-object {'{1,-6} : {0,2}' -f ($_+1), $tomb[$_]}
egy    :  1
ketto  :  2
harom  :  3
negy   :  4
ot     :  5
hat    :  6
hét    :  7
nyolc  :  8
kilenc :  9
tíz    : 10
```

Ugye ez már ezek után érthető, előre vettem a tömbelemet, hat karakter szélességben, balra igazítva, utána a tömbindexet, két karakter szélességben, jobbra igazítva.

Megjegyzés

Végezetül egy meglepő kifejezés:

```
[26] PS I:\>Get-Date -f MMddyyHHmmss
052708131411
```

Vigyázat! Ez nem formátum operátor, hanem a `get-date`-nek a `format` paramétere rövidítve! Ennek használatáról bővebben a http://msdn.microsoft.com/en-us/library/system.globalization.datetimeformatinfo.aspx oldalon lehet olvasni.

1.5.16 Átirányítás: „>”, „>>”

A kimenetet átirányíthatjuk a (>) és a (>>) jelekkel. Nézzünk erre példát az előző fejezet tízelemű tömbjének felhasználásával:

```
[41] PS C:\> 0..3 | foreach-object {'{1,-6} : {0,2}' -f ($_+1), $tomb[$_]} >
c:\szamok.txt
```

```
[42] PS C:\> Get-Content C:\szamok.txt
egy    :  1
ketto  :  2
harom  :  3
negy   :  4
[43] PS C:\> "még egy sor" >> C:\szamok.txt
[44] PS C:\> Get-Content C:\szamok.txt
egy    :  1
ketto  :  2
harom  :  3
negy   :  4
még egy sor
```

A [41]-es sorban átirányítottam a kimenetet egy szöveges fájlba. A szimpla (>) jel új fájlt kezd. A fájl tartalmának kiírásához a `get-content` cmdletet használtam.

A [43]-as sor dupla (>>) jelével hozzáfűztem egy újabb sort a fájlhoz.

```
[45] PS C:\> "új sor" > C:\szamok.txt
[46] PS C:\> Get-Content C:\szamok.txt
új sor
```

A [45]-ös sorban, mivel megint szimpla (>) jelet használtam, ezért elveszettem a fájl addigi tartalmát.

1.5.17 Az összefűzés és szétdarabolás operátora (-join, -split, -csplit)

A PowerShell 2.0 két új operátort hoz be a repertoárjába, mindkettő tulajdonképpen korábban is használatos funkciókat hajt végre talán egy picit egyszerűbben. Az egyik a sztringek összefűzését végrehajtó –`join` operátor:

```
PS C:\> "egy", "kettő", "három" -join "-"
egy-kettő-három
```

Az operátor bal oldalán áll az összefűzendő szövegek tömbje, jobb oldalán meg az összefűzéskor beillesztett karakter vagy sztring.

Érdekes módon, egyoperandusú operátorként is használhatjuk, de ehhez szerintem egy picit PowerShell-idegen formátumot kell alkalmazni:

```
PS C:\> -join ("egy", "kettő", "három")
egykettőhárom
```

Ilyenkor az összefűző karakter egy üres-sztring. Én személy szerint nem igazán javaslom a használatát.

Megjegyzés

Ráadásul itt azért kell a zárójelet használni, mert a PowerShell parancsfeldolgozója az egytagú operátorokat nagyobb precedenciájúnak tekinti, mint a többparaméteres operátorokat (jelen esetben a vessző tömboperátort), így a zárójel nélkül nem igazán jó eredményt kapunk:

```
[66] PS C:\> -join "egy", "kettő", "három"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
egy
kettő
három
```

Hiszen itt valójában a `(-join "egy"), "kettő", "három"` kerül végrehajtásra.

A művelet fordítottját, azaz a szétdarabolást a `-split` operátor végzi:

```
PS C:\> "egy-kettő-három" -split "-"
egy
kettő
három
```

Ráadásul ez regex szemlélettel működik alaphelyzetben, azaz számjegyek mentén való darabolást az alábbi kifejezéssel lehet végrehajtani:

```
PS C:\> "egy1kettő2három3" -split "\d"
egy
kettő
három
```

Ha még ez sem lenne elég, akkor `$true` vagy `$false` értékre kiértékelődő bármilyen PowerShell kifejezés is írható:

```
PS C:\> "egy-kettő+három.négy!öt" -split {$_ -eq "-" -or $_ -eq "+"}
egy
kettő
három.négy!öt
```

Ezt úgy kell értelmezni, mint egy csőfeldolgozást: érkeznek a karakterek a sztringből, és amelyik kielégíti a feltételt, amentén tördel.

A tördelés eredményét be is lehet korlátozni. Ha én csak maximum 3 darabot szeretnék kapni a széttördelés után, akkor a `-split`-nek még egy szám paramétert adva ezt megtehetem:

```
PS C:\> '1.2.3.4.5.6.7.8' -split "\.", 3
1
2
3.4.5.6.7.8
```

(Ugye itt sem felejtkezünk meg a regex szintaxisról, azaz a „`.`” mentén való tördeléshez a „`\.`” kifejezést kell használni.)

A tördelés operátorának van néhány beállítható opciója is. Ha az a kifejezés, ami mentén tördelni akarunk egyszerű kifejezés, és esetleg olyan karakterek is vannak benne, amelyek a regexben vezérlő karakterek, akkor használhatjuk a „`SimpleMatch`” opciót:

```
[1] PS C:\munka> 'Egy$kettő$három' -split '$',0
Egy$kettő$három

[2] PS C:\munka> 'Egy$kettő$három' -split '$',0, "simplematch"
Egy
kettő
```

```
három
```

A fenti első nem működött, hiszen a „$” jel speciális jelentésű a regexben, a második eset már a kívánt eredményt adta. Ha ilyen opciókat használunk, akkor kötelező használni a maximális darabszámot meghatározó paramétert is (0 jelenti az összes darabot), hiszen csak így kerül helyére a „`simplematch`” opció.

A regex típusú tördelésnél a regex összes opcióját használhatjuk:

```
[RegexMatch] [,IgnoreCase] [,CultureInvariant] [,IgnorePatternWhitespace]
[,ExplicitCapture] [,Singleline | ,Multiline]
```

Ráadásul a `-split` operátornak van egy `-csplit` kis-nagybetű érzékeny változata is:

```
[67] PS C:\> "axb" -csplit "x"
a
b
[68] PS C:\> "aXb" -csplit "x"
aXb
```

Látható, hogy a fenti példákban a `-csplit` csak a kisbetűs „x” mentén tördelt.

A könyv tanfolyami felhasználása nem engedélyezett!

1.6 Vezérlő utasítások

Bár az eddigi példákban és a csővezeték jellegű feldolgozás miatt sokszor bonyolultabb esetben sincs szükség a feldolgozási sorrend megváltoztatására, természetesen egy komoly programnyelv nem nélkülözheti a vezérlő utasításokat sem.

1.6.1 IF/ELSEIF/ELSE

Az `IF-ELSEIF-ELSE` elágazás az egyik legegyszerűbb lehetőség. Be lehet írni akár egysoros kifejezésekbe is, de akkor a kicsit nehezen értelmezhető. Viszont szkriptekben praktikusan használható:

```
[47] PS C:\> $a = 25
[48] PS C:\> if($a -gt 10)
>> { "nagyobb, mint 10" }
>> elseif($a -lt 10)
>> { "kisebb, mint 10" }
>> else
>> { "pont 10" }
>>
nagyobb, mint 10
```

Természetesen nem kötelező az IF mellett ELSEIF és ELSE használata, bármelyik, akár mindkettő is elhagyható.

1.6.2 WHILE, DO-WHILE

A `WHILE` segítségével ciklust szervezhetünk. Kétfajta változata is van, a `WHILE` elöl tesztelő és a `DO-WHILE` a hátul tesztelő. A hátul tesztelő mindenképpen lefut egyszer, az elöl tesztelő akár a belsejének lefuttatása nélkül is továbbléphet, ha az alkalmazott feltétel rögtön hamis. Nézzünk pár példát:

```
[13] PS C:\> $i = 3
[14] PS C:\> while($i -lt 6)
>> {
>>     "*" * $i
>>     $i++
>> }
>>
***
****
*****
```

Amíg az `$i` kisebb, mint 6, addig `$i` darabszámú csillagot írok ki, és növelem az `$i`-t. A ciklus 5 csillagnál lép ki, mivel a 6.-nál már nem igaz az, hogy kisebb, mint 5.

És egy hátul tesztelő:

```
[16] PS C:\> $a=4
[17] PS C:\> do
>> {
>>     $a++
>>     $a
>> }
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
>> while ($a -lt 6)
>>
5
6
```

Itt meg a ciklus 6-nál lépett ki, hiszen a feltétel nem teljesülését csak akkor vette észre, amikor már a ciklus magja lefutott.

1.6.3 FOR

Miután az előző ciklusokban is láttuk, hogy viszonylag gyakori szerkezet az, hogy a ciklus elején van egy értékadás (inicializálás), aztán van a feltétel, aztán van egy értékváltoztatás része, így erre van egy tömörebb forma is, a `FOR` ciklus:

```
[18] PS C:\> for($a=0; $a -lt 3; $a++)
>> {
>>     [char] (65+$a)
>> }
>>
A
B
C
```

Az *1.5.10 Csoportosító operátorok* fejezetben ismertetett módon a `$()` felhasználásával egyszerre több kifejezést is tehetünk a `FOR` ciklus különböző tevékenységet végző pozícióiba:

```
[24] PS C:\> for($($i=1; $j=10); $i -lt 4; $($i++; $j--)){"{0,2} {1,2}" -f $i,
$j}
 1 10
 2  9
 3  8
```

1.6.4 FOREACH

Mivel a PowerShellben nagyon sokszor gyűjteményekkel (collection) dolgozunk, így az alkotók praktikusnak találták ezek elemein történő műveletvégzést megkönnyítő ciklust is készíteni. Köszönet ezért nekik! A `FOREACH` kulcsszó segítségével olyan ciklust tudunk létrehozni, ahol nem nekünk kell nyilvántartani, számlálni és léptetni a ciklusváltozót, hanem a PowerShell ezt megteszi helyettünk:

```
[25] PS C:\> $egyveleg = 1,"szöveg",(get-date),@{egy=2}
[26] PS C:\> foreach($elem in $egyveleg){$elem.gettype()}

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType
True     True     String                                   System.Object
True     True     DateTime                                 System.ValueType
True     True     Hashtable                                System.Object
```

Látszik a ciklus működési elve: az `$elem` változóba a PowerShell mindig betölti az aktuális tömbelemet az `$egyveleg` tömbből egészen addig, amíg van elem, és minden elem mellett a sor végén látható

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

szkriptblokkot végrehajtja. Ugyanezt `FOR` ciklussal is meg tudnánk csinálni, de mennyivel többet kell gépelni, és eggyel több változóra is szükségünk van, plusz még az olvashatósága is sokkal nehézkesebb:

```
[27] PS C:\> $egyveleg = 1,"szöveg",(get-date),@{egy=2}
[28] PS C:\> for($i=0;$i -lt $egyveleg.length;$i++){$egyveleg[$i].GetType()}
...
```

Láthattunk a *1.4.3.1 Egyszerű tömbök* fejezetben, hogy a PowerShell képes automatikusan „kifejteni” a tömböket (gyűjteményeket) például a `get-member` cmdletbe való becsövezéskor. A `FOREACH` ennek ellenkezőjét végzi, azaz egy skaláris (nem tömb) paraméterből képes automatikusan egyelemű tömböt készíteni, ha erre van szükség:

```
[29] PS I:\>$skalár = 1
[30] PS I:\>foreach($szám in $skalár){"Ez egy szám: $szám"}
Ez egy szám: 1
```

A fenti példában nem okozott a PowerShellnek problémát `foreach` ciklust futtatni egy nem tömb típusú változóra, automatikusan egy elemű tömbként kezelte.

1.6.4.1 $foreach változó

A `foreach` ciklus belsejében a PowerShell automatikusan létrehoz egy `$foreach` változót, aminek a `$foreach.current` tulajdonsága az éppen aktuális elemet tartalmazza, így akár a [26]-os sorban levő példát másként is írhatnánk:

```
[53] PS C:\> foreach($elem in $egyveleg) {$foreach.current.gettype()}

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType
True     True     String                                   System.Object
True     True     DateTime                                 System.ValueType
True     True     Hashtable                                System.Object
```

Nézzük meg, hogy ennek a `$foreach` változónak milyen tagjellemzői vannak:

```
 [50] PS C:\> foreach($elem in "a"){,$foreach | get-member}


   TypeName: System.Array+SZArrayEnumerator

Name        MemberType Definition
----        ---------- ----------
Clone       Method     System.Object Clone()
Equals      Method     System.Boolean Equals(Object obj)
GetHashCode Method     System.Int32 GetHashCode()
GetType     Method     System.Type GetType()
get_Current Method     System.Object get_Current()
MoveNext    Method     System.Boolean MoveNext()
Reset       Method     System.Void Reset()
ToString    Method     System.String ToString()
Current     Property   System.Object Current {get;}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Az igazán izgalmas számunkra a `MoveNext()` metódus, amellyel arra késztethetjük a `foreach` ciklust, hogy egy elemet kihagyjon:

```
[54] PS C:\> $tömb = 1,2,3,4,5,6,7
[55] PS C:\> foreach($elem in $tömb){$elem; $foreach.MoveNext()}
1
True
3
True
5
True
7
False
```

A fenti példában miután minden elemnél rögtön még egy elemet léptettünk, ezért csak minden második számot írtam ki. Ugyan a `MoveNext()` ad visszatérési `$true` vagy `$false` értéket attól függően, hogy van-e még elem vagy nincs, de ezt elnyomhatjuk a `[void]` típuskonverzióval:

```
[56] PS C:\> foreach($elem in $tömb){$elem; [void] $foreach.MoveNext()}
1
3
5
7
```

A `$foreach` változó másik fontos metódusa a `Reset()`, ezzel vissza lehet ugrani a ciklus első elemére:

```
[60] PS C:\> $első = $true
[61] PS C:\> foreach($elem in $tömb){$elem; if($első -and !$foreach.MoveNext())
{$foreach.Reset(); $első = $false}}
1
3
5
7
1
2
3
4
5
6
7
```

A fenti példában egy ciklussal kétszer is végigjárom a `$tömb` tagjait, első menetben minden másodikat írom ki, a második menetben pedig az összes tagot. Itt is jól jött, hogy az `-or` feltétel második operandusa csak akkor kerül kiértékelésre, ha az első tag `$false`, ezzel értem el, hogy a második menetben már ne legyen végrehajtva a `MoveNext()`.

1.6.5 ForEach-Object cmdlet

Az előzőekben bemutatott `FOREACH` egy PowerShell kulcsszó ciklus létrehozására. Azonban létezik egy másik `FOREACH` is, ami a `Foreach-Object` cmdlet álneve:

```
[25] PS C:\> get-alias foreach
```

```
CommandType     Name                                Definition
-----------     ----                                ----------
Alias           foreach                             ForEach-Object
```

Honnan tudja a PowerShell hogy egy parancssorban most melyiket is akarom használni: a kulcsszót vagy az cmdletet? Nagyon egyszerűen: ha a `foreach` után egy zárójel „`(`” van, akkor kulcsszóval van dolga, ha „`{`” bajusz, akkor alias. Másrészt a kulcsszavak mindig a kifejezések elején kell álljanak. A kifejezések eleje vagy a parancssor eleje, vagy pontosvessző (`;`), vagy nyitó zárójel „(”. A csőjel (`|`) nem számít kifejezés elejének.

Mire lehet használni ezt a `foreach-object` cmdletet? Míg a `foreach` kulcsszó tömbje, amin végigmegy, rendelkezésre kell álljon teljes egészében készen, mire a vezérlés ráadódik, addig a `ForEach-Object` cmdlet csővezetékből érkező elemeket dolgoz fel, azaz ha már egy bemeneti elem elkészült, akkor azt már át is veszi és elvégzi a kijelölt műveletet:

```
[1] PS C:\> 1,2,3 | ForEach-Object {$_ * 3}
3
6
9
```

A fenti példában nem a `$foreach` változón keresztül értem el az aktuális elemet, hanem a szokásos `$_` változón.

Megjegyzés:

Van egy érdekessége ennek a cmdletnek. Figyeljük meg a help-ben a szintaxisát:

```
[2] PS C:\> get-help foreach-object

NAME
    ForEach-Object

SYNOPSIS
    Performs an operation against each of a set of input objects.


SYNTAX
    ForEach-Object [-process] <ScriptBlock[]> [-inputObject <psobject>] [-b
    egin <scriptblock>] [-end <scriptblock>] [<CommonParameters>]
...
```

Hoppá! Nem is egy, hanem három szkriptblokkot lehet átadni neki paraméterként, kicsit hasonlóan, mint ahogy a FOR ciklusnál is három rész vezérli a ciklust. Itt a három rész szerepe: mit tegyen az első elem érkezése előtt, mit tegyen az éppen aktuális elemek érkezésekor, mit tegyen az utolsó elem érkezése után.

Nézzünk erre egy példát:

```
[13] PS C:\> "egy","kettő","három" | foreach-object {"----eleje----"} {$_} {"--
--vége----"}
----eleje----
egy
kettő
három
----vége----
```

Ebben a példában nem is írtam ki a paraméterek neveit, azaz a pozíció alapján rendelte hozzá az egyes szkriptblokkokat a PowerShell. Ha csak egy szkriptblokkot adunk meg az a `Process` szekcióhoz sorolódik.

1.6.6 Where-Object cmdlet

Gyakran fordul az elő, hogy nem minden gyűjtemény-elemmel szeretnénk foglalkozni, hanem csak bizonyos tulajdonsággal rendelkezőkkel, és ezeket szeretnénk továbbküldeni a csővezetékünkben feldolgozásra, a szűrőfeltételünknek nem megfelelő elemeket egyszerűen el szeretnénk dobni. Ugyan ezt meg lehet tenni a `ForEach-Object` és az `IF` kombinációjával, de van egy direkt erre kitalált cmdletünk is, a `Where-Object`:

```
[1] PS C:\> $tömb = 1,2,"szöveg", 3
[2] PS C:\> $tömb | Where-Object {$_ -is [int]} | ForEach-Object {$_*2}
2
4
6
```

Ugyan a `Where-Object` nem vezérlő utasítás, a hagyományos programozási értelemben, de mivel a csővezetékek kezelése annyira tipikus a PowerShellben, ezért minden csővezetéket „vezérlő” utasítást is nyugodtan sorolhatunk ebbe a kategóriába.

1.6.7 Címkék, törés (Break), folytatás (Continue)

Előfordulhat, hogy a ciklusunkat nem szeretnénk minden körülmények között végigfuttatni minden elemre. Ha teljesen ki akarunk lépni, akkor a `Break` kulcsszót használjuk, ha csak az adott elemen akarunk továbblépni, akkor a `Continue` kulcsszó használható. Nézzünk erre példákat:

```
[4] PS C:\> $tömb= 1,2,"szöveg",3
[5] PS C:\> for($i=0; $i -lt $tömb.length;$i++){if($tömb[$i] -is
[int]){$tömb[$i]*2}else{continue};" ez most a $i. elem duplája volt"}
2
 ez most a 0. elem duplája volt
4
 ez most a 1. elem duplája volt
6
 ez most a 3. elem duplája volt
```

A fenti példában van egy vegyes elemű `$tömb`-öm. Én az egész számokat tartalmazó elemekre szeretnék a szám dupláját kiírni és kikerülni a nem egész számokat. Látszik, hogy az `IF`-nek az `ELSE` ágán a `continue` után már nem hajtódik végre a szöveg kiírása.

Nézzünk egy másik példát, ahol nem átugrunk cikluselemet, hanem teljesen kiugrunk a ciklusból mihelyst oda nem illő elemet találunk:

```
[25] PS C:\> $tömb= 1,2,"szöveg",3
[34] PS C:\> foreach($elem in $tömb){if($elem -isnot [int]){"Hiba!!!!: $elem";
break}; $elem}
1
2
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Hiba!!!!: szöveg
```

Mi van akkor, ha nem egy, hanem két ciklusunk van egymásba ágyazva, és ekkor akarok kiugrani a ciklusokból? Honnan tudja a PowerShell, hogy melyik ciklusból akartam kiugorni, csak a belsőből, vagy mindkét rétegből? Ennek tisztázására használhatunk címkéket, amelyek megmondják a `break` utasításnak, hogy hova ugorjon ki:

```
[38] PS C:\> for($i=0; $i -lt 3;$i++)
>> {
>>     for($j=5; $j -lt 8;$j++)
>>         {
>>             "i:{0} j:{1}" -f $i, $j
>>             if($j -eq 6){break}
>>         }
>> }
>>
i:0 j:5
i:0 j:6
i:1 j:5
i:1 j:6
i:2 j:5
i:2 j:6
```

A fenti példában látszik, hogy amikor a `$j` eléri a 6-os értéket és lefut a `break`, akkor csak a belső ciklusból léptem ki, megjelenik az `$i=1` és `$i=2` érték is, azaz a külső ciklusból nem lépett ki. Hogyan tudok mindkét ciklusból egyszerre kilépni? Használjunk címkét:

```
[40] PS C:\> :külső for($i=0; $i -lt 3;$i++)
>> {
>>     for($j=5; $j -lt 8;$j++)
>>         {
>>             "i:{0} j:{1}" -f $i, $j
>>             if($j -eq 6){break külső}
>>         }
>> }
>>
i:0 j:5
i:0 j:6
```

Látszik, hogy a `$j=6` első előfordulásánál kilépett mindkét ciklusból.

A címkéket tehát mindig a ciklust képző kifejezés elejére kell tenni, ahonnan ki akarok lépni. A címkéket a PowerShell bentről kifelé keresi, azaz ha ugyanolyan címkéből több is van, akkor a legmélyebben található címkéből lép ki csak:

```
[24] PS C:\> :címke do
>> {
>>     :címke do
>>     {
>>         break címke
>>     }while($false)
>>     "belső"
>> }while($false)
>> "külső"
>>
belső
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
külső
```

A fenti példából látható, hogy a belső `DO` ciklusból léptem csak ki a `BREAK` kifejezéssel, hiszen a kimeneten megjelent a „belső” kimenet is. Ha a külsőből lépett volna ki, akkor csak a „külső” feliratot láthattuk volna.

A címkék globálisak, azaz ha én egy szkriptet hívok egy ciklusban, és a szkriptben van hivatkozás egy külső címkére, akkor is kilép. Nézzük a szkriptet:

```
while($true)
{
    "Bent vagyok"
    Break messze
}
```

És az ezt meghívó interaktív kifejezést, valamint a kimenetet:

```
PS C:\Users\tibi> :messze while($true){C:\_munka\powershell2\v2\munka\globalbre
ak.ps1}
Bent vagyok
```

Látható, hogy mindkét ciklus végtelen lenne, de ott van a `BREAK`, ráadásul kívülre mutat, így mindkét ciklusból kilépett.

Megjegyzés:

Még trükkösebb lehetőség, hogy a `break` attribútuma nem csak statikus szöveg lehet, hanem ez a címke egy változóból is jöhet. Azaz bizonyos helyzetekben, mondjuk, az egyik címkéhez ugrunk vissza, más esetekben egy másik címkéhez, és ehhez csak a változónk tartalmát kell változtatni.

1.6.8 SWITCH

A `SWITCH` kulcsszóval nagyon bonyolult `IF/ELSEIF/ELSE` feltételrendszerek tudunk nagyon átláthatóan, egyszerűen megvalósítani. Sőt! Még akár bonyolultabb `ForEach-Object` vagy `Where-Object` cmdleteket is tudunk helyettesíteni ezzel úgy, hogy a szkriptünk sokkal átláthatóbb és egyszerűbb lesz. Nézzünk erre egy példát, először egy `IF` szerkezet kiváltására:

```
[46] PS C:\> $a = 1
[47] PS C:\> if($a -gt 10){"nagyobb mint 10"}elseif($a -gt 5) {"kisebbegyenl
ő 10, nagyobb 5"} else{"kisebbegyenlő 5"}
kisebbegyenlő 5
```

Ennek `switch`-et alkalmazó megfelelője:

```
[48] PS C:\> switch($a)
>> { {$a -gt 10} {"nagyobb mint 10"}
>>     {$a -gt 5} {"kisebbegyenlő 10, nagyobb 5"}
>>     default {"kisebbegyenlő 5"}
>> }
>>
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
kisebbegyenlő 5
```

Vajon tényleg ekvivalens egymással? Nézzük meg, hogy mi van akkor, ha `$a=15`:

```
[49] PS C:\> $a = 15
[50] PS C:\> if($a -gt 10){"nagyobb mint 10"}elseif($a -gt 5){"kisebbegyenlő
10, nagyobb 5"}else{"kisebbegyenlő 5"}
nagyobb mint 10
```

`Switch`-csel:

```
[52] PS C:\> switch($a)
>> { {$a -gt 10} {"nagyobb mint 10"}
>>     {$a -gt 5} {"kisebbegyenlő 10, nagyobb 5"}
>>     default {"kisebbegyenlő 5"}
>> }
>>
nagyobb mint 10
kisebbegyenlő 10, nagyobb 5
```

Azt láthatjuk itt, hogy a `switch` különböző sorai nem igazi `ELSEIF` módon működnek, azaz nem csak akkor adódik rájuk a vezérlés, ha a megelőző sorok nem adtak eredményt, hanem minden esetben. Ha ezt nem szeretnénk, akkor használjuk a `BREAK` kulcsszót:

```
[53] PS C:\> switch($a)
>> { {$a -gt 10} {"nagyobb mint 10"; break}
>>     {$a -gt 5} {"kisebbegyenlő 10, nagyobb 5"; break}
>>     default {"kisebbegyenlő 5"}
>> }
>>
nagyobb mint 10
```

Ez már ugyanazt a kimenetet adta, mint az `IF`-es megoldás. Tulajdonképpen a `default` ág előtti `break` felesleges, mert oda a vezérlés csak akkor kerül, ha egyik megelőző feltétel sem teljesül, de ha kiírjuk, akkor sem okoz ez semmilyen problémát.

A fenti példák talán még nem érzékeltetik eléggé a `switch` előnyét, hiszen itt csak egy háromtagú `IF` láncolatot helyettesítettem. Többtagú kifejezés esetén jobban látszódna a `switch` átláthatósága, bár így is, a legbeágyazottabb `ELSE` ág helyett sokkal szebben néz ki egy `default` címke.

Még jobban kiviláglik a `switch` egyszerűsége, ha egyenlőségre vizsgálunk, hiszen ekkor nem kell külön az egyenlőség operátort sem használni:

```
[1] PS C:\> $a = 10
[2] PS C:\> switch($a){ 1 {"egy"} 2 {"kettő"} 3 {"három"} 10 {"tíz"} default
{"egyéb"}}
tíz
```

De itt nem is ér véget a `switch` előnye, mert – ellentétben az `IF`-fel – a `switch` még gyűjteményeket is képes kezelni, így nagyon jó alternatívája lehet a `ForEach-Object` és a `Where-Object` cmdleteknek is. Például egy „ékezetlenítő” szkript csak ennyiből áll:

```
[42] PS C:\> $a = "öt új űrírás az ütvefúrógépen"
```

```
[43] PS C:\> $ofs="";"$(switch([char[]] $a){'á' {'a'} 'é' {'e'} 'í' {'i'} 'ó'
{'o'} 'ö' {'o'} 'ő' {'o'} 'ú' {'u'} 'ü' {'u'} 'ű' {'u'} default {$_}})"
ot uj uriras az utvefurogepen
```

Itt a `switch` megkapja az `$a` sztringből képzett karaktertömböt (`[char[]]`), majd a sok karaktercserét megvalósító `switch` tag után visszakonvertálom a karaktersorozatot adó kimenetet (az egész kifejezés `$()`-ben) sztringgé (`""`), de úgy, hogy ne legyen elválasztó szóköz karakter a betűk között (`$ofs=""`).

Megjegyzés

Bár nem használtam a fenti példában explicit csövet, azaz a „|” csőjelet, de mégis hivatkozható a `switch`-ben a $_ változó.

1.6.8.1 –wildcard

És még itt sem ér véget a `switch` lehetőség-tára. Gyakran foglakozunk szövegek vizsgálatával, így például ilyen jellegű kifejezések is gyakran szükségessé válnak:

```
[44] PS C:\> $a = "szöveg"
[45] PS C:\> switch ($a) {{$a -like "s*"} {"s-sel kezdődik"} {$a -like "z*"}
{"z-vel kezdődik"}}
s-sel kezdődik
```

Ha ezt próbálnánk egyszerűbben írni, és kihagyjuk a „`-like`” operátort, akkor nem kapunk jó eredményt:

```
[46] PS C:\> switch($a){s* {"s-sel kezdődik"} z* {"z-vel kezdődik"}}
[47] PS C:\>
```

Hiszen ekkor a `switch` egyenlőséget vizsgált. Ha `-like` feltétel van mindenütt, akkor a `switch -wildcard` kapcsolójával tudjuk az alaphelyzet szerinti egyenlőségvizsgálatot `-like` vizsgálattá alakítani:

```
[48] PS C:\> switch -wildcard ($a) {s* {"s-sel kezdődik"} z* {"z-vel
kezdődik"}}

s-sel kezdődik
```

1.6.8.2 –regex

Szintén gyakori eset a szövegminták vizsgálata, mint ahogy erre már láttunk példát a *1.5.6 Regex (-match, -replace)* fejezetben. A `switch` erre is fel van készítve:

```
[3] PS C:\> $vizsgálandó="telefon: (30) 311-6867"
[4] PS C:\> switch -regex ($vizsgálandó)
>> {'\((\d{1,2})\)(-|\s)' {"Körzetszám: $($matches[1])"; break}
>> default {"Nincs körzetszám"}}
>>
Körzetszám: 30
[5] PS C:\> $vizsgálandó = "Telefon: 123-4567"
[6] PS C:\> switch -regex ($vizsgálandó)
>> {'\((\d{1,2})\)(-|\s)' {"Körzetszám: $($matches[1])"; break}
>> default {"Nincs körzetszám"}}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
>>
Nincs körzetszám
```

A fenti példában zárójelek közti 1 vagy 2 számjegyből álló mintát keresek a vizsgálandó szövegben, amely mintát vagy szóköz, vagy kötőjel követ. Ha van ilyen, akkor a regex belső csoportképzésével kiadódó zárójelek közti számot adom vissza körzetszám gyanánt, egyébként kiíratom, hogy nincs körzetszám.

1.6.8.3 A $switch változó

Hasonlóan a `ForEach` kulcsszóhoz, a `switch`-nek is van belső változója, melynek `$switch` a neve, és amelyet felhasználhatunk ciklus-jellegű működés megvalósításra:

```
[8] PS C:\> $Hosszú = "Vezetéknév: Soós
>> Keresztnév: Tibor
>> e-mail: soostibor@citromail.hu
>> Telefon: 30-3116867
>> Város: Budapest"
>>
[9] PS C:\> $darabok = $Hosszú.Split()
[10] PS C:\> $darabok
Vezetéknév:
Soós
Keresztnév:
Tibor
e-mail:
soostibor@citromail.hu
Telefon:
30-3116867
Város:
Budapest
[12] PS C:\> switch($darabok) {
>> "Vezetéknév:" {[void] $switch.MoveNext(); $vez = $switch.Current}
>> "Keresztnév:" {[void] $switch.MoveNext(); $ker = $switch.Current}
>> "e-mail:" {[void] $switch.MoveNext(); $ema = $switch.Current}
>> "Telefon:" {[void] $switch.MoveNext(); $tel = $switch.Current}
>> "Város:" {[void] $switch.MoveNext(); $var = $switch.Current}
>> }
>>
[13] PS C:\> $ker
Tibor
[14] PS C:\> $vez
Soós
```

Tehát ez a változó akkor használható igazából, ha egy gyűjteményt adunk át a `switch`-nek. Ekkor a gyűjtemény egyes tagjait a `$switch` változó az ő `MoveNext()` metódusával ciklus-szerűen tudja kezelni. Maga a `switch` pedig meg tudja vizsgálni az egyes elemeket és a vizsgálat eredményének függvényében különböző kódrészletek futhatnak le.

A fenti példában van egy hosszú sztringem, amiből ki szeretném szedni a nem címke jellegű adatokat, és ezeket be akarom tölteni a megfelelő változókba. Elsőként feldarabolom a sztringet a `Split()` metódussal. Az így megszülető szavakból álló `$darabok` tömböt átadom a `switch`-nek. Ha a `switch` címkét talál (Vezetéknév:, Keresztnév:, stb.), akkor továbblép a következő darabra és azt betölti a megfelelő változóba. Így minden adat pont a helyére került.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.7 Függvények

A PowerShell 2.0 236 darab cmdlettel rendelkezik, ami elég soknak tűnik elsőre, de amint elkezdünk dolgozni ebben a környezetben rájöhetünk, hogy bizonyos műveletek elvégzéséhez újra és újra ugyanazt az utasítássort használjuk. Ilyenkor érdemes lehet ezeket a műveleteket elmenteni, elmentés előtt esetleg – a szélesebb körű felhasználhatóság érdekében – általánosítani. Az elmentésnek két lehetősége van: függvény és szkript készítése. Persze ez a kétfajta mentési típus nem különül el általában egymástól, hiszen függvénydefiníciót tehetünk szkriptbe, és szkriptet is meghívhatunk függvényből. A közös bennük tehát, hogy tartalmaznak egy olyan PowerShell utasítássort, amely már korábban definiált utasításokból – kulcsszavakból, cmdletekből, függvényekből, szkriptekből, stb. – áll.

A fő különbség a kettő között, hogy a függvényeknek minden esetben van saját nevük és futás időben jönnek létre, míg a szkripteknek nem feltétlenül van saját nevük, viszont fájlokban tároljuk őket, ezeknek a fájloknak viszont biztos van nevük, és a szkriptek a fájljuk betöltésével hajtódnak végre.

Elsőként nézzük a függvényeket!

1.7.1 Az első függvényem

Az első függvényem nem sok mindent fog csinálni, kiírja üzembiztosan, hogy „első”:

```
[1] PS C:\> function függvény1 {"első"}
```

Ki is próbálom:

```
[2] PS C:\> függvény1
első
```

Amikor a függvényt meghívom, a PowerShell valójában az általam beírt függvénynév helyett a háttérben végrehajtja, amit én a függvénydefinícióban a két kapcsos zárójel {} közé írtam.

1.7.2 Paraméterek

Természetesen a `függvény1`-nek nem sok értelme van, de például készítsünk egy olyan függvényt, ami kiszámítja egy kör területét. Nem egy konkrét körére gondolok, hanem olyan függvényt szeretnék, amibe paraméterként átadhatom az aktuális kör sugarát, és ebből számítja a területet:

```
[3] PS C:\> function körterület ($sugár)
>> {
>>     $sugár*$sugár*[math]::Pi
>> }
>>
[4] PS C:\> körterület 12
452,38934211693
[5] PS C:\> körterület 1
3,14159265358979
```

A könyv tanfolyami felhasználása nem engedélyezett!

Ez a formátum hasonlatos a hagyományos programnyelvek függvénydefinícióihoz, de a függvényem meghívásának módja kicsit más. Nem kell zárójelet használni, és ha több paraméterem lenne, azok közé nem kell vessző, mert ugye a PowerShellben a vessző a tömb elemeinek szétválasztására való!

1.7.2.1 Paraméterinicializálás

Nézzünk egy újabb függvényt, egy négyszög területét akarom kiszámoltatni:

```
[12] PS C:\> function négyszög ($x, $y)
>> { $x*$y }
>>
[13] PS C:\> négyszög 5 6
30
```

Mi van akkor, ha egy négyzetnek a területét akarom kiszámoltatni, aminek nincs külön x és y oldala, csak x:

```
[14] PS C:\> négyszög 7
0
```

Hát ez nem sikerült, hiszen a függvényem két paramétert várt, az `$y` nem kapott értéket, így a szorzat értéke 0 lett. Lehetne persze a függvényen belül egy IF feltétellel megvizsgálni, hogy vajon a `$y` kapott-e értéket, és ha nem, akkor x^2-et számolni. Ennél egyszerűbben is lehet ezt elérni a paraméterek inicializálásával:

```
[15] PS C:\> function négyszög ($x, $y = $x)
>> { $x*$y }
>>
[16] PS C:\> négyszög 7
49
```

Itt a [15]-ös sorban az `$y` változónak átadom az `$x` értékét. Ez az értékadás azonban csak akkor jut érvényre, ha a függvény meghívásakor én nem adok neki külön értéket, azaz továbbra is működik a téglalap területének kiszámítása is:

```
[17] PS C:\> négyszög 8 9
72
```

Megjegyzés

Mi lenne, ha a négyszög függvény meghívására a hagyományos, zárójeles formátumot használnám most:

```
[22] PS C:\> négyszög(2,3)
Cannot convert "System.Object[]" to "System.Int32".
At line:2 char:6
+ { $x*$ <<<< y }
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Meg is kaptam a hibát, hiszen a függvényem nincs felkészülve egy kételemű tömb területének kiszámítására.

A paramétereknek nem csak másik változó adhat alapértéket, hanem tehetünk oda egy konstanst is:

```
[18] PS C:\> function üdvözlet ($név = "Tibi")
>> { "Szia $név!" }
>>
[19] PS C:\> üdvözlet Zsófi
Szia Zsófi!
[20] PS C:\> üdvözlet
Szia Tibi!
```

1.7.2.2 Típusos paraméterek

Hasonlóan, ahogy a változóknál is definiálhatunk típust, a függvények paramétereinél is jelezhetem, hogy milyen típusú adatot várok. Ez a függvény hibás működésének lehetőségét csökkentheti.

Nézzük a körterület-számoló függvényemet, mit tesz, ha szövegként adok be neki sugarat:

```
[33] PS C:\> körterület "2"
222222
```

Elég érdekesen alakul ez a geometria a PowerShellben! De eddigi ismereteink alapján rájöhetünk, hogy mi történt. Ugye a függvényünk belsejében ez található:

```
$sugár*$sugár*[math]::Pi
```

Az első `$sugár` a PowerShell szabályai szerint lehet „2” (szövegként). A második már nem, viszont működik az automatikus típuskonverzió, tehát csinál belőle 2-t (szám). Idáig kapunk „22”-t, azaz az első sztringemet („2”) kétszer egymás mellett. Majd jön a PI, ugyan az nem egész, de itt is jön a típuskonverzió, lesz belőle 3, azaz immár a „22”-t rakja egymás mellé háromszor, így lesz belőle „222222”.

Na, ezt nem szeretnénk, ezért tegyük típusossá a sugarat:

```
[42] PS C:\> function körterület ([double] $sugár)
>> {
>>     $sugár*$sugár*[math]::Pi
>> }
>>
[43] PS C:\> körterület "2"
12,5663706143592
```

Itt már helyes irányba tereltük a PowerShell típuskonverziós automatizmusát, rögtön a paraméterátadásnál konvertálja a szöveges 2-t számmá, innentől már nincs félreértés.

Megjegyzés

Az is helyes eredményt adott volna, ha felcseréltük volna a tényezők sorrendjét:

```
[math]::Pi *$sugár*$sugár
```

De elegánsabb és „hibatűrőbb”, ha inkább típusjelzőkkel látjuk el a paramétereket.

1.7.2.3 Hibajelzés

Az előző körterület függvényemnél, ha a paraméter nem konvertálható `[double]` típussá, akkor hibát kapunk, anélkül, hogy mi ezt leprogramoztuk volna:

```
[19] PS C:\>körterület "nagy"
körterület : Cannot convert value "nagy" to type "System.Double". Error: "Input
string was not in a correct format."
At line:1 char:11
+ körterület  <<<< "nagy"
```

Viszont elképzelhető olyan paraméterérték is, ami bár nem okoz problémát közvetlenül se a típusos paraméternek, se a függvény belsejének, de mi mégis szeretnénk hibaként kezelni. A körterület példámnál tekintsük hibának, ha valaki negatív számot ad meg sugárnak. A fentihez hasonló formátumú (piros betűk) hibaüzenetet írattathatunk ki a `throw` kulcsszó segítségével:

```
[22] PS C:\>function körterület ([double] $sugár)
>> {
>>     if ($sugár -lt 0)
>>         { throw "Pozitív számot kérek!" }
>>      $sugár*$sugár*[math]::Pi
>> }
>>
[23] PS C:\>körterület -2
Pozitív számot kérek!
At line:4 char:16
+         { throw  <<<< "Pozitív számot kérek!" }
```

De nem csak ilyen esetben lehet szükség hibakezelésre, hanem akkor is, ha valaki nem ad meg paramétert. Egy paraméter kötelező vagy opcionális voltát majd a *2.4 Fejlett függvények – script cmdletek* fejezetben, a fejlett függvények függvénydefiníciójában tudjuk majd igazán profi módon beállítani, de most itt egyszerűen a `throw` segítségével végezzük ezt el:

```
[29] PS C:\>function körterület ([double] $sugár = $(throw "kötelező paramét
er"))
>> {
>>     if ($sugár -lt 0)
>>         { throw "Pozitív számot kérek!" }
>>      $sugár*$sugár*[math]::Pi
>> }
>>
[30] PS C:\>körterület
kötelező paraméter
At line:1 char:47
+ function körterület ([double] $sugár = $(throw  <<<< "kötelező paraméter"
))
[31] PS C:\>körterület 5
78,5398163397448
```

Látszik, hogy már az értékadásnál használhatjuk a hibajelzést, mint alapértéket. Ha nem ad a felhasználó paraméterként értéket, akkor végrehajtódik a hibakezelés, viszont ha ad értéket, akkor természetesen erre nem kerül sor.

1.7.2.4 Változó számú paraméter

Előfordulhat olyan is, hogy nem tudjuk előre, hogy hány paraméterünk lesz egy függvényben. Például szeretnénk kiszámolni tetszőleges darabszámú sztring hosszának az átlagát. Ilyenkor dilemmába kerülhetünk, hiszen hány paramétert soroljunk fel? Hogyan inicializáljuk ezeket?

Szerencsére nem kell ezen töprengenünk, mert a PowerShell készít automatikusan egy `$args` nevű változót, amely tartalmazza a függvénynek átadott valamennyi olyan paramétert, amelyet úgy adunk át, hogy nem névvel hivatkozunk rájuk:

```
[25] PS C:\> function átlaghossz
>> {
>>     $hossz = 0
>>     if($args)
>>         {
>>             foreach($arg in $args)
>>                 {$hossz += $arg.length}
>>             $hossz = $hossz/$args.count
>>         }
>>     $hossz
>> }
>>
[26] PS C:\> átlaghossz "Egy" "kettő" "három" "négy"
4,25
```

Az `átlaghossz` függvényemnek ezek alapján nincs explicit paramétere, azaz névvel nem is lehet hivatkozni a paramétereire, viszont implicit módon az `$args` tömbön keresztül természetesen megkapja mindazt a paramétert, amelyet átadunk neki. Ebben az esetben azonban nehezebben tudjuk kézben tartani a paraméterek típusát, ezt is majd a fejlett függvényekkel tudjuk elvégezni (*2.4 Fejlett függvények – script cmdletek* fejezet).

Mi van az `$args` változóval, ha vannak explicit paramétereink is? Nézzünk erre egy példát:

```
[46] PS C:\> function mondatgenerátor ($eleje, $vége)
>> {
>>     write-host $eleje, $args, $vége
>> }
>>
[47] PS C:\> mondatgenerátor "Egy" "kettő" "három" "négy"
Egy három négy kettő
```

Láthatjuk, hogy úgy működik a PowerShell, ahogy vártuk, a nevesített paraméterek nem kerülnek bele az `$args` tömbbe, így a fenti függvényemben egyetlen argumentum sem lett duplán kiírva.

1.7.2.5 Hivatkozás paraméterekre

Eddig az összes függvénypéldámban a függvények meghívásakor a paraméterátadás pozíció alapján történt. Ez egyszerűen azt jelenti, hogy az első paraméter átadódik a függvénydefiníció paraméterei között felsorolt első változónak, a második paraméter a második változónak és így tovább. Ha több paramétert

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

adunk át, mint amennyit a függvény definíciójában felsoroltunk, akkor ezek az extra paraméterek az `$args` változóba kerülnek bele.

Mi van akkor, ha egy függvénynek opcionális paramétere is van, és nem akarok neki értéket adni? Ezt eggyel több szóköz leütésével jelezzem? Alakítsuk át ennek szemléltetésére az előző „mondatgenerátor" függvényemet:

```
[48] PS C:\> function mondatgenerátor ($eleje, $közepe, $vége)
>> {
>>     write-host "Eleje: $eleje közepe: $közepe vége: $vége"
>> }
>>
[49] PS C:\> mondatgenerátor egy kettő három
Eleje: egy közepe: kettő vége: három
[50] PS C:\> mondatgenerátor egy  három
Eleje: egy közepe: három vége:
```

Itt három explicit paramétert fogad a függvény. Ha tényleg három paraméterrel hívom meg, akkor minden paraméter a helye alapján kerül a megfelelő helyre. Ha a második paramétert ki szeretném hagyni, akkor ezt hiába akarom jelezni eggyel több szóközzel ([50]-es sor), ez természetesen a PowerShell parancsértelmezőjét nem hatja meg, így más módszerhez kell folyamodnom, ez pedig a név szerinti paraméterátadás:

```
[55] PS C:\> mondatgenerátor -eleje egy -vége három
Eleje: egy közepe:  vége: három
```

Így már tényleg minden a helyére került. A név szerinti paraméterátadásnál is számos gépelést segítő lehetőséget biztosít számunkra a PowerShell:

```
[56] PS C:\> mondatgenerátor -e egy  -v három
Eleje: egy közepe:  vége: három
[57] PS C:\> mondatgenerátor -e:egy  -v:három
Eleje: egy közepe:  vége: három
```

Az [56]-os sorban látható, hogy a nevekre olyan röviden elég hivatkozni, ami még egyértelművé teszi, hogy melyik paraméterre is gondolunk. Az [57]-es sorban meg az látható, hogy a PowerShell elfogadja a sok egyéb parancssori környezetben megszokott paraméterátadási szintaxist is, azaz hogy kettőspontot teszünk a paraméternév után.

A hely szerinti és pozíció szerinti paraméterhivatkozást vegyíthetjük is:

```
[58] PS C:\> mondatgenerátor -v: három egy -k: kettő
Eleje: egy közepe: kettő vége: három
```

Itt a szabály az, hogy a PowerShell először kiveszi a függvényhívásból a nevesített paramétereket, majd hely szerint feltölti a többi explicit paramétert, és ha még mindig marad paraméter, azt berakja az `$args` változóba.

1.7.2.6 Paraméterátadás változó szétpasszírozásával (splat operátor)

Nézzük, hogy hogyan lehet még paramétereket átadni. Vegyük példa gyanánt az előző mondatgenerátor függvényt. Egy változóban vannak tömbként az egyes paraméterek, vajon hogyan lehet a tömb elemeit átadni rendre az egyes paraméterek gyanánt?

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[2] PS C:\> $v = "a", "b", "c"
[3] PS C:\> mondatgenerátor $v
Eleje: a b c közepe:  vége:
```

A [2]-es sorban látható a paramétereke tartalmazó tömböm, de ha ezt adom át a mondatgenerátoromnak, akkor nem azt kapom, amit szerettem volna. Az átadott változó mint egy egység adódott át az első, „eleje” paraméterként, és a többi paraméternek nem jutott érték.

A PowerShell 2.0-ban bevezettek egy új operátort, a „szétpasszírozás” operátorát (angolul *splatting* – szétfröccsenés), amit pont arra találtak ki, hogy egy tömbváltozó elemeit adja át rendre a függvény vagy cmdlet paramétereiként:

```
[4] PS C:\> mondatgenerátor @v
Eleje: a közepe: b vége: c
```

Ez pont azt az eredményt adta, amit szerettem volna. Enélkül elég bonyolult lett volna ezt a fajta paraméterátadást megoldani.

Nézzük, hogy hogyan lehet név alapján átadni a paramétereket a szétpasszírozás operátorával. Ilyenkor a változóba hashtáblát kell tenni:

```
[7] PS C:\> $v = @{ közepe = "b"; vége = "c"; eleje = "a"}
[8] PS C:\> $v

Name                           Value
----                           -----
eleje                          a
közepe                         b
vége                           c


[9] PS C:\> mondatgenerátor @v
Eleje: a közepe: b vége: c
```

A fenti példában láthatjuk, hogy a paraméterek nem a sorrendjük, hanem a nevük alapján lettek átadva.

1.7.2.7 Kapcsoló paraméter ([switch])

Sok függvénynél előfordul olyan paraméter, amelynek csak két állapota van: szerepeltetjük a paramétert vagy sem. Ez nagyon hasonlít a `[bool]` adattípushoz, de feleslegesen sokat kellene gépelni, ha tényleg `[bool]`-t használnánk:

```
függvény -bekapcsolva $true
```

Sokkal egyszerűbb lenne csak ennyit írni:

```
függvény -bekapcsolva
```

Erre találták ki a `[switch]` adattípust, amelyet elsődlegesen pont ilyen paraméterek esetében használunk.

```
[60] PS C:\> function lámpa ([switch] $bekapcsolva)
>> {
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
>>      if($bekapcsolva) {"Világos van!"}
>>      else {"Sötét van!"}
>> }
>>
[61] PS C:\> lámpa
Sötét van!
[62] PS C:\> lámpa -b
Világos van!
```

A fenti „lámpa” függvényemet, ha kapcsoló nélkül használom, akkor „sötétet” jelez (az IF ELSE ága), ha van kapcsoló, akkor „világos”. Azaz, ha szerepel a paraméterek között a kapcsoló, akkor változójának értéke `$true` lesz, ha nem szerepel a kapcsoló, akkor változójának értéke `$false` lesz.

Megjegyzés

Korábban volt arról szó, hogy a paraméterek név szerinti hivatkozása mellett akár lehet szóközzel, akár kettősponttal értéket adni. Ez a `[switch]` paramétertípusnál nem mindegy:

```
[63] PS C:\old> lámpa -b $false
Világos van!
[64] PS C:\old> lámpa -b:$false
Sötét van!
```

A [63]-as sorban szóközzel választottam el a paramétertől az értéket, ezt a függvényem figyelmen kívül hagyta és a kapcsolónak `$true` értéket adott. A [64]-es sorban kettősponttal adtam a paraméternek értéket, itt már az elvárt módon `$false` értéket vett fel a kapcsolóm.

1.7.2.8 Paraméter-definíció a függvénytörzsben (param)

Ha valaki olyan programozói háttérrel rendelkezik, ahol megszokta, hogy a függvénydefinícióknál van egy külön deklarációs rész, akkor használhatják a `param` kulcsszót a függvény paramétereinek deklarálásához:

```
[2] PS I:\>function get-random
>> {
>>      param ([double] $max = 1)
>>      $rnd = new-object random
>>      $rnd.NextDouble()*$max
>> }
>>
[3] PS I:\>get-random 1000
315,589330306085
```

Itt a `get-random` függvényem neve után nem szerepel semmilyen paraméter, hanem beköltöztettem a függvénytörzsbe, itt viszont kötelezően egy `param` kulcsszóval kell jelezni a paramétereket. Ez akkor is hasznos lehet, ha bonyolultabb kifejezésekkel adok kezdőértéket a paramétereimnek, mert ezzel a külön deklarációs résszel sokkal olvashatóbb lesz a függvényem.

Ennek paraméterezési lehetőségnek egyébként nem is itt, hanem majd a szrkipteknél lesz még inkább jelentősége.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.7.2.9 Paraméterek, változók ellenőrzése (validálás)

Természetesen a paraméterek ellenőrzése akkor a legegyszerűbb, ha eleve csak a kívánalmainknak megfelelő értékeket lehetne átadni. A típus megfelelőségének vizsgálatához már láthattuk, egyszerűen csak egy típusjelölőt kell a paraméter neve előtt szerepeltetni. De vajon milyen lehetőségünk van, ha még további megszorításokat szeretnénk tenni?

Ennek egyik lehetősége, hogy a függvényünk törzsében saját programkóddal ellenőrizzük az érték helyességét. Például nézzünk a korábban már látott körterület kiszámító függvényt:

```
[22] PS C:\>function körterület ([double] $sugár)
>> {
>>     if ($sugár -lt 0)
>>         { throw "Pozitív számot kérek!" }
>>      $sugár*$sugár*[math]::Pi
>> }
>>
```

Itt én ellenőrzöm, hogy csak pozitív számot lehessen megadni sugárnak. A .NET keretrendszerben azonban vannak kifejezetten ilyen jellegű, paraméterellenőrzésre szolgáló osztályok is. Ilyen szabályokat hozzá is rendelhetünk változókhoz, erre szolgál a változók tulajdonságai között az `Attributes` tulajdonság:

```
[1] PS C:\> $a = 1
[2] PS C:\> Get-Variable a | fl


Name        : a
Description :
Value       : 1
Options     : None
Attributes  : {}
```

Alaphelyzetben ez a tulajdonság üres, vajon hogyan és mivel lehet feltölteni? Nézzük meg, hogy milyen metódusai vannak ennek az attribútumnak:

```
[8] PS C:\> ,(Get-Variable a).Attributes | gm


   TypeName: System.Management.Automation.PSVariableAttributeCollection

Name           MemberType            Definition
----           ----------            ----------
Add            Method                System.Void Add(Attribute item)
Clear          Method                System.Void Clear()
Contains       Method                System.Boolean Contains(Attribute item)
CopyTo         Method                System.Void CopyTo(Attribute[] array...
Equals         Method                System.Boolean Equals(Object obj)
GetEnumerator  Method                System.Collections.Generic.IEnumerat...
GetHashCode    Method                System.Int32 GetHashCode()
GetType        Method                System.Type GetType()
get_Count      Method                System.Int32 get_Count()
get_Item       Method                System.Attribute get_Item(Int32 index)
IndexOf        Method                System.Int32 IndexOf(Attribute item)
Insert         Method                System.Void Insert(Int32 index, Attr...
Remove         Method                System.Boolean Remove(Attribute item)
RemoveAt       Method                System.Void RemoveAt(Int32 index)
```

```
set_Item      Method                 System.Void set_Item(Int32 index, At...
ToString      Method                 System.String ToString()
Item          ParameterizedProperty  System.Attribute Item(Int32 index) {...
Count         Property               System.Int32 Count {get;}
```

Tehát van neki `Add` metódusa, ezzel lehet ellenőrzési objektumokat hozzáadni ehhez az attribútumhoz. A PowerShell MSDN weboldalán[10] és a Reflector programmal a `System.Management.Automation` névtérbe beásva ezeket az ellenőrzési osztályokat lehet megtalálni:

Osztály neve	Felhasználási terület
System.Management.Automation.ValidateArgumentsAttribute	Szülő osztály
System.Management.Automation.ValidateCountAttribute	Attribútumok számának ellenőrzése
System.Management.Automation.ValidateEnumeratedArgumentsAttribute	Szülő osztály
System.Management.Automation.ValidateLengthAttribute	Hossz ellenőrzése (pl.: sztring, tömb)
System.Management.Automation.ValidatePatternAttribute	Regex minta ellenőrzése (sztring)
System.Management.Automation.ValidateRangeAttribute	Értéktartomány ellenőrzése (pl.: int, double)
System.Management.Automation.ValidateSetAttribute	Értéklista ellenőrzése (sztring)
System.Management.Automation.ValidateNotNullAttribute	Nem null érték ellenőrzése
System.Management.Automation.ValidateNotNullOrEmptyAttribute	Nem null és nem üresség ellenőrzése

Nézzünk ezek használatára néhány példát. Az elsőben szeretnék olyan változót használni, amely csak 1 és 5 darab karakter közötti hosszúságú szöveget fogad el:

```
[87] PS C:\old>$vl = New-Object System.Management.Automation.ValidateLengthA
ttribute 1,5
[88] PS C:\old>$szó = "rövid"
[89] PS C:\old>(get-variable szo).attributes.add($vl)
[90] PS C:\old>$szó="nagyonhosszú"
Cannot validate because of invalid value (nagyonhosszú) for variable szo.
At line:1 char:5
+ $szo= <<<< "nagyonhosszú"
```

A [87]-es sorban definiálom a `VaildateLengthAttribute` osztály egy objektumát. Majd létrehozok egy `$szó` nevű változót „rövid” tartalommal. Ennek a változónak az `Attributes` tulajdonságához hozzáadom az előbb létrehozott ellenőrzési objektumot. A [90]-es sorban a változómnak egy 5 karakternél hosszabb értéket szeretnék adni, de erre hibát kaptam.

Nézzünk egy hasonló példát az üresség vizsgálatára:

```
[94] PS C:\old>$s2="valami"
[95] PS C:\old>$vnne = New-Object System.Management.Automation.ValidateNotNu
llOrEmptyAttribute
[96] PS C:\old>(get-variable s2).Attributes.add($vnne)
[97] PS C:\old>$s2=""
Cannot validate because of invalid value () for variable s2.
At line:1 char:4
+ $s2= <<<< ""
```

Ha egy ilyen ellenőrzési attribútummal látunk el egy változót, akkor az megmutatkozik a változó tulajdonságai között is:

[10] http://msdn.microsoft.com/en-us/library/system.management.automation.validateargumentsattribute(VS.85).aspx

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[98] PS C:\old>Get-Variable s2 | fl *


Name        : s2
Description :
Value       : valami
Options     : None
Attributes  : {System.Management.Automation.ValidateNotNullOrEmptyAttribute
              }
```

Ezekből az ellenőrzési objektumokból egyszerre többet is lehet egy változóhoz rendelni.

Sajnos ezen ellenőrzési objektumok felhasználásának egyik fő nehézsége az, hogy ahhoz, hogy ilyen szabályt egy változóhoz rendelhessük, ahhoz addigra már a változónak olyan értékkel kell rendelkeznie, amelyre teljesül az ellenőrzési feltétel. Azaz, ha például egy *NotNull* típusú ellenőrzést szeretnénk hozzárendelni egy változóhoz, addigra már valamilyen értékkel kell rendelkeznie a változónak, különben nem engedi a szabályt hozzárendelni. Így a függvények paramétereinél történő felhasználásuk kicsit körülményes:

```
[9] PS C:\> function nemnull ($p)
>> {
>>     $pteszt = 1
>>     $vnn = New-Object System.Management.Automation.ValidateNotNullAttribu
te
>>     (Get-Variable pteszt).Attributes.Add($vnn)
>>
>>     $pteszt = $p
>> }
>>
[10] PS C:\> nemnull 100
[11] PS C:\> nemnull
Cannot validate because of invalid value () for variable pteszt.
At line:6 char:12
+     $pteszt  <<<< = $p
```

Azaz a függvény törzsében létrehozok egy tesztelési célokat szolgáló `$pteszt` változót, ami kielégíti azt a feltételt, hogy nem üres az értéke, és ez a változó hordozza az ellenőrzési objektumot is. A függvény paraméterének ellenőrzésére a `$p` változó értékét átadom ennek a tesztelési célra létrehozott változónak. Amikor a függvényemet paraméter nélkül hívtam meg, akkor hibát kaptam. Itt igazából csak annyit spóroltam, hogy nem nekem kell lekezelni és kiírni a hibát, ezt a PowerShell helyettem elvégezte.

Az igazi, profi paraméterellenőrzést a PowerShell 2.0 a fejlett függvényeknél már lehetővé teszi, erről a *2.4 Fejlett függvények – script cmdletek* fejezetben lesz szó.

1.7.3 Függvény a parancsfeldolgozás előnyeinek kihasználására

A parancs-üzemmódot felhasználhatjuk például arra, hogy egyszerűen hozzunk létre sztringtömböket. Merthogy ilyet létrehozni nem túl egyszerű:

```
[22] PS C:\munka> $stringtömb = egy, kettő, három, négy, öt
The term 'egy' is not recognized as the name of a cmdlet, function, script fil
e, or operable program. Check the spelling of the name, or if a path was inclu
ded, verify that the path is correct and try again.
At line:1 char:18
+ $stringtömb = egy <<<< , kettő, három, négy, öt
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    + CategoryInfo          : ObjectNotFound: (egy:String) [], CommandNotFoun
   dException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Láthatóan ebben a formában nem tudunk sztringtömböt létrehozni, hiszen most a sor eleji $ jel miatt kifejezésmódban voltam, így az egyenlőségjel utáni szöveget megpróbálja kifejezésként értelmezni, ami nem megy.

A megoldás ez lenne:

```
[23] PS C:\munka> $stringtömb = "egy", "kettő", "három", "négy", "öt"
```

Ezzel sajnos nagyon sok idézőjelet kellett begépelni, ami elég fárasztó. Hogyan lehetne „átverni” a PowerShellt, hogy automatikusan kezelje ezeket a szövegeket sztringként? Készítsünk egy olyan függvényt, ami paraméterként tudja fogadni a fenti szövegeket és ekkor a parancsfeldolgozás miatt életbe lép az automatikus sztringkonverzió:

```
[24] PS C:\munka> function str {$args}
[25] PS C:\munka> $stringtömb = str egy, kettő, három, négy, öt
[26] PS C:\munka> $stringtömb
egy
kettő
három
négy
öt
```

A fenti példában készítettem egy nagyon egyszerű függvényt, ami nem csinál semmit, mit hogy visszaadja átalakítás nélkül a neki adott paramétereket. Mivel a függvény meghívásakor már a PowerShell parancsmódba vált, így nem a függvénynek kell konvertálni, hanem maga a PowerShell ezt elvégzi helyettünk. Sőt! Még a vesszők is elhagyhatók a fenti példából:

```
[27] PS C:\munka> $stringtömb = str egy kettő három négy öt
[28] PS C:\munka> $stringtömb
egy
kettő
három
négy
öt
```

1.7.4 Változók láthatósága (scope)

A PowerShellben a változóknak van szabályozható láthatóságuk (scope). Az alapszabály, hogy egy adott környezetben létrehozott változók az alkörnyezetekben is láthatók. Környezet maga a PowerShell konzol, egy függvény, egy script. Ahogy ezeket egymásból meghívjuk generálódnak az al-, al-, … környezetek. A PowerShell konzol az un. „global” környezet, az összes többi, innen hívott függvény vagy szkript ennek alkörnyezete.

Nézzük az egyszerű láthatóság alapesetét:

```
[1] PS I:\>$a = "valami"
[2] PS I:\>function fv{$a}
[3] PS I:\>fv
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
valami
```

[1]-ben létrehoztam egy $a változót, [2]-ben egy nagyon egyszerű függvényt, ami csak annyit csinál, hogy visszaadja $a-t, majd [3]-ban meghívtam ezt az fv nevű függvényt. Az eredmény látható: a függvény „látta” $a-t, ki tudta írni a tartalmát.

Nézzünk egy kicsit bonyolultabb esetet, mi van akkor, ha a függvényemnek is van egy $a változója:

```
[6] PS I:\>$a = "valami"
[7] PS I:\>function fv1{$a="másvalami";$a}
[8] PS I:\>fv1
másvalami
[9] PS I:\>$a
valami
```

[7]-ben az fv1 függvényem definiál egy $a-t, majd ki is írja. [8]-ban meg is kaptam a „belső” $a eredményét, de ettől függetlenül [9]-ben a „global” környezetből nézve az $a továbbra is az, ami volt. Azaz a gyerek környezet nem hat vissza a szülőre.

Ha szükségünk van külön a szülő változójára is a függvényen belül, akkor ezt meg tudjuk tenni a get-variable cmdlet -scope paraméterének beállításával. A „-scope 0” jelenti a saját környezetet, „-scope 1” jelenti a szülőkörnyezetet, „-scope 2” jelentené a nagyszülőt és így tovább:

```
[11] PS I:\>function fv2{$a="másvalami";$a; get-variable a -scope 0}
[12] PS I:\>fv2
másvalami

Name                           Value
----                           -----
a                              másvalami
```

[11]-ben a saját $a-t írja ki a get-variable, a következő példában pedig a szülő $a változóját:

```
[14] PS I:\>function fv2{$a="másvalami";$a; get-variable a -scope 1}
[15] PS I:\>fv2
másvalami

Name                           Value
----                           -----
a                              valami
```

A gyerek környezet függvénye meg is tudja változtatni a szülő környezet változóját, ha a set-variable cmdlet ugyanilyen -scope paraméterét megadjuk:

```
[17] PS I:\>function fv3{$a="másvalami";$a; (get-variable a -scope 1).value;
set-variable a -scope 1 -value "változás"}
[18] PS I:\>fv3
másvalami
valami
[19] PS I:\>$a
változás
```

Itt a fv3 függvényben vége felé megváltoztatom a szülő $a változóját „változás”-ra.

A 0 és a legfelső környezetre némiképp egyszerűbb szintaxissal is hivatkozhatunk:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[25] PS I:\>$c=1
[26] PS I:\>function kie{$c=2; "Én c-m:$local:c";"Te c-d:$global:c"}
[27] PS I:\>kie
Én c-m:2
Te c-d:1
[28] PS I:\>function ront{$c=2; $local:c=3;$global:c=4;"Én c-m:$local:c";"Te c-
d:$global:c"}
[29] PS I:\>ront
Én c-m:3
Te c-d:4
```

Egyébként vigyázni kell nagyon a függvényekkel! Az alábbi kis példában, ha a programozó abból a feltételezésből indul ki, hogy a `$b` az úgyis 0 kezdetben, így a `$b=$b+5` értéke 5 lesz, akkor nagyot fog csalódni:

```
[21] PS I:\>$b=10
[22] PS I:\>function elront{$b=$b+5; $b}
[23] PS I:\>elront
15
[24] PS I:\>$b
10
```

Mi történt itt? A láthatóság miatt az `elront` függvényben az értékadás jobb oldalán szereplő `$b` az a szülő `$b`-je, a bal oldali `$b` már a függvény belső `$b`-je. Tehát – bár a nevük ugyanaz – tárhely szempontjából különbözőek.

Tipp

A függvények belső változóit mindig értékadás után kezdjük el használni, nehogy tévedésből a szülő környezet által definiált ugyanolyan nevű változó értékével kezdjünk el dolgozni.

Ha nem szeretnénk, hogy egy változónkhoz valamelyik gyerek környezet hozzáférhessen, akkor nyilvánítsuk privátnak ezt a változót:

```
[32] PS I:\>$private:priv=1
[33] PS I:\>function latom{"Privát: $priv"}
[34] PS I:\>latom
Privát:
[35] PS I:\>$priv
1
```

Ha egy függvénynek mégis ennek értékével kell dolgoznia, akkor természetesen paraméterátadással ez megtehető minden további nélkül, lényeg az, hogy a privát változókat a gyerek környezet nem fogja tudni a szülő tudta nélkül megváltoztatni.

Azonban azért így sem lehetünk teljes biztonságban, hiszen a `get-variable` és `set-variable` cmdletekkel még a privát változókat is elérhetjük:

```
[36] PS C:\> $private:priv=1
[37] PS C:\> function latom{"Privát: $((get-variable priv -scope 1).value)"}
[38] PS C:\> latom
Privát: 1
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[39] PS C:\> function ront{set-variable priv -scope 1 -value 99}
[40] PS C:\> ront
[41] PS C:\> $priv
99
```

A fenti példában láthatjuk, hogy annak ellenére, hogy privátnak definiáltam a `$priv` változómat, ennek ellenére a `latom` függvényem is el tudta érni a `get-variable` segítségével, sőt, a `ront` függvényem még meg is tudta változtatni az értékét.

Nézzünk további lehetőségeket! Definiáltam egy `scope` nevű függvényt, amellyel az „aa” betűkkel kezdődő nevű, helyi változónak látszó változókat listázom ki:

```
[44] PS C:\> function scope {get-variable aa* -Scope local}
[45] PS C:\> scope
```

Ez alaphelyzetben nem ad találatot. Most definiálok egy `$aa_felső` változót, nézzük sajátjának tekinti-e a függvényem:

```
[47] PS C:\> $aa_felső = 1
[48] PS C:\> scope
```

Ez sem adott találatot. Most szintén a PowerShell legfelsőbb szintjén definiálok egy újabb változót, de egy speciális opció, az `AllScope` megadásával:

```
[50] PS C:\> New-Variable -Name aa_mindenütt -Value 2 -Option AllScope
[51] PS C:\> scope

Name                           Value
----                           -----
aa_mindenütt                   2
```

Ebből az látható, hogy a függvény is sajátjának érzi az `$aa_mindenütt` változót! Olyannyira, hogy meg is lehet változtatni az értékét függvényből is:

```
[53] PS C:\> function scope-allscope {$aa_mindenütt++}
[54] PS C:\> scope-allscope
[55] PS C:\> $aa_mindenütt
3
```

És természetesen ha egy függvény definiál egy ugyanilyen nevű változót, akkor az nem újként jön létre, hanem a korábban létrehozott kap új értéket:

```
[56] PS C:\> function scope-újmindenütt {$aa_mindenütt = 10}
[57] PS C:\> scope-újmindenütt
[58] PS C:\> $aa_mindenütt
10
```

1.7.4.1 Privát, privát, privát

Egy `Private` felhasználást láttunk már, de nézzük ezt egy picit részletesebben. Amit korábban mutattam, a `$Private:változó` formátumban azt a `New-Variable` cmdlettel is természetesen el lehet érni:

```
[26] PS C:\> New-Variable -name titkos -Value "titok" -Option private
[27] PS C:\> $titkos
titok
[28] PS C:\> function miez {$titkos}
[29] PS C:\> miez
```

A fenti példában a `$titok` változó alapban nem látható a gyerekkörnyezetként létrejövő függvényemben. Miután a TAB-kiegészítés is függvényként fut, ezért az ilyen változók neveit nem lehet kiegészíttetni sem. Viszont a `Get-Variable` látja a saját scope-jában:

```
[30] PS C:\> Get-Variable ti*

Name                           Value
----                           -----
titkos                         titok
```

Viszont van egy másik `Private` beállítás is, ez nem `-option`, hanem `-visibility`:

```
[31] PS C:\> New-Variable -name titkos2 -Value "titok2" -Visibility private
[32] PS C:\> function miez {$titkos2}
[33] PS C:\> miez
titok2
[34] PS C:\> Get-Variable ti*

Name                           Value
----                           -----
titkos                         titok
```

Látható, hogy az így felvett változó már láthatóság szempontjából elérhető egy al-scope-ból, de a `Get-Variable` nem látja. És most jön a meglepetés:

```
[36] PS C:\> $titkos2
Cannot access the variable '$titkos2' because it is a private variable
At line:1 char:9
+ $titkos2 <<<<
    + CategoryInfo          : PermissionDenied: (titkos2:String) [], SessionS
   tateException
    + FullyQualifiedErrorId : VariableIsPrivate
```

A saját scope-ból nem szólítható meg. Ennek értelme nem a globális scope-ban van, hanem majd a következő fejezet végén nyer értelmet, és majd a modulok tárgyalásánál.

1.7.5 Függvények láthatósága, „dotsourcing”

Hasonlóan a változókhoz, a függvényeknek is van láthatóságuk, „scope”-juk. Nézzünk erre is egy példát:

```
[12] PS I:\>function külső ($p)
>> {
>>     function belső ($b)
>>     {
>>         Write-Host "Belső paramétere: $b"
>>     }
>>     write-host "Külső paramétere: $p"
>>     belső 2
```

```
>>     $k = "külső változója"
>> }
>>
[13] PS I:\>külső 1
Külső paramétere: 1
Belső paramétere: 2
[14] PS I:\>belső 3
The term 'belső' is not recognized as a cmdlet, function, operable program,
 or script file. Verify the term and try again.
At line:1 char:6
+ belső  <<<< 3
[15] PS I:\>. külső 4
Külső paramétere: 4
Belső paramétere: 2
[16] PS I:\>belső 5
Belső paramétere: 5
```

A [12]-es sorban definiált `külső` függvényemben definiálok egy `belső` függvényt. Amúgy a külső csak annyit csinál, hogy kiírja a paraméterét, meghívja a belsőt és még definiál magának egy `$k` változót. A belső csak annyit csinál, hogy kiírja a paraméterét.

Amikor a külsőt meghívom a [13]-as sorban, az szépen le is fut: a külső és a belső paramétere is kiíródik. Amikor a [14]-es sorban közvetlenül a belsőt hívom meg, akkor a PowerShell csodálkozik, hiszen a global scope számára a belső függvény nem látható.

Viszont van lehetőség arra, hogy a külső függvényemet „felemelem” globális szintre és így hívom meg, ezt tettem meg a [15]-ös sorban. Ennek formátuma nagyon egyszerű: egy darab pont, egy szóköz (fontos!) és a függvény neve. Ezt hívjuk dotsourcing-nak, azaz a meghívás szintjére emelem a függvény (vagy szkript, ld. később) futtatását.

Ez természetesen nem csak a belső függvénydefiníciót emelte fel a globális scope-ba, hanem a külső saját változóját is:

```
[17] PS I:\>$k
külső változója
```

Ha nem akarunk dotsourcing-gal bíbelődni, akkor mi magunk is definiálhatjuk eleve globálisnak a függvényben definiált függvényünket:

```
[23] PS I:\>function függvénytár
>> {
>>     function global:első
>>         { "első" }
>>     function global:második
>>         { "második" }
>> }
>>
[24] PS I:\>függvénytár
[25] PS I:\>első
első
[26] PS I:\>második
második
```

Ha ezek után meghívom a „függvénytár” függvényemet, definiálódik benne az „első” és a „második” függvény is, de a globális scope-ban, így közvetlenül meghívhatók.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés:

Ez a „dotsourcing” nagyon erős, azaz ha így hívok meg egy függvényt, akkor a benne tárolt minden „titok” meghívható lesz közvetlenül. Még akkor is, ha `private`-nak definiálom a belső függvényemet:

```
[28] PS I:\>function titkos
>> {
>>     function private:egymegegy
>>         { "1+1=2" }
>> }
>>
[29] PS I:\>. titkos
[30] PS I:\>egymegegy
1+1=2
```

Ámde ha egy változónál a `–Visibility` lehetőségnél állítjuk be a `Private`-ot, akkor már más a helyzet:

```
[114] PS C:\> function priv {New-Variable -Name pv -Visibility private -Value 5
; $pv}
[115] PS C:\> . priv
5
[116] PS C:\> $pv
Cannot access the variable '$pv' because it is a private variable
At line:1 char:4
+ $pv <<<<
    + CategoryInfo          : PermissionDenied: (pv:String) [], SessionStateE
   xception
    + FullyQualifiedErrorId : VariableIsPrivate
```

Itt nyer értelmez igazából ez a lehetőség, azaz hiába hívom dotsource-szal a függvényt, a privát láthatóságú változó még akkor sem látható kívülről.
Összefoglalóul: a privát opció a gyerek-scope-ok elől rejti el a változókat, a privát láthatóság a szülő-scope elől rejti el.

1.7.6 Referenciális hivatkozás paraméterekre ([ref])

Előfordulhat olyan igény, hogy a függvényemmel nem új értékeket akarok létrehozni, hanem jól bevált, meglevő változóm értékét akarom megváltoztattatni. Erre is lehetőséget ad a PowerShell, hiszen létezik referencia szerinti hivatkozás is:

```
[10] PS I:\>function bánt ([ref] $v)
>> {
>>     $v.value ++
>> }
>>
[11] PS I:\>$a = 1
[12] PS I:\>bánt ([ref] $a)
[13] PS I:\>$a
2
[14] PS I:\>bánt ([ref] $a)
[15] PS I:\>$a
3
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A `bánt` függvényem közvetlenül bántja a paraméterként átadott változóm értékét. Ezt úgy tudja megtenni, hogy neki nem érték alapján, hanem referencia (azaz memóriacím) alapján adjuk át a változót, amin aztán ő tud dolgozni. Mivel ez a memóriacím egy referenciát tartalmazó változóba töltődik át (`$v`), ezért értelemszerűen nem ennek a változónak az értékét kell manipulálni, hanem ezen referencia mögötti értéket kell módosítani (`$v.value`).

A függvény meghívásánál is oda kell figyelni, hiszen ha csak simán ennyit írnánk, az nem vezetne eredményre:

```
bánt $a
```

Ilyenkor a PowerShell nem végez automatikus típuskonverziót, azaz nem automatikusan az `$a` változó referenciáját adja át, segíteni kell neki a `[ref]` típusjelzővel. És mivel a szóközzel elválasztott két részt (`[ref]` és `$a`) alaphelyzetben két paraméterként értelmezné, ezért kell zárójelezni, és akkor már jó lesz, mint ahogy ez a [14]-es sorban látható volt:

```
[14] PS I:\>bánt ([ref] $a)
```

1.7.7 Kilépés a függvényből (return)

Elképzelhető, hogy egy függvényből nem a legvégén akarunk kilépni, mert időközben egy elágazás alapján már nem kell további műveletet végezni. Ennek egyik oka lehet valamilyen hiba, amit a *1.7.2.3 Hibajelzés* fejezetben látott `throw` kulcsszóval jelezni tudunk, és itt a függvény végrehajtása meg is szakad.

Ha nem hiba miatt, hanem „normális” módon akarunk egy függvényből kilépni, akkor erre használhatjuk a `return` kulcsszót.

```
[11] PS C:\>function típus ($i)
>> {
>>     switch ($i)
>>     {
>>         {$i -is [int32]} {"Egész" ; return}
>>         {$i -is [double]} {return "Akár tört"}
>>         {$i -is [string]} {return "Szöveg"}
>>     }
>>     "Nem igazán tudom, hogy mi az, hogy $i"
>> }
>>
[12] PS C:\>típus 32
Egész
[13] PS C:\>típus "bla"
Szöveg
[14] PS C:\>típus (get-date)
Nem igazán tudom, hogy mi az, hogy 04/15/2008 09:47:04
```

A példafüggvényemben típust vizsgálok a `switch` kulcsszó segítségével. Először vizsgálom, hogy az `$i` paraméter egész-e? Ha igen, akkor kiírom, hogy „Egész” és kilépek a függvényből. Használhattam volna a `break` kulcsszót is, de az csak a `switch`-ből lépett volna ki és végrehajtotta volna a függvény utolsó utasítását is.

A `return` másik szintaxisát használtam az „Akár tört” és a „Szöveg” ágon, azaz a függvény outputja lesz a `return` paramétere.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Látszik, hogy csak nagyon ritkán elkerülhetetlen a `return` alkalmazása, hiszen az én példámban is tehettük volna azt is, hogy a „Nem igazán tudom...” kiírást berakjuk a `switch default` ágába is, illetve a `return`-ök helyett alkalmazhattam volna `break`-et is (ha nem akartam volna kikerülni a függvény végén található kifejezéseket). Mindenesetre bizonyos helyzetekben jól jöhet a `return`, tudjunk róla!

1.7.8 Pipe kezelése, filter

Készítsünk olyan függvényt, ami kiszámolja a neki átadott fájlok hosszainak összegét!

```
[19] PS C:\old>function fájlhossz ([System.IO.FileInfo] $f)
>> {
>>     $hossz = 0
>>     $hossz += $f.length
>>     $hossz
>> }
>>
[20] PS C:\old>fájlhossz C:\old\alice.txt
709
```

Ez tökéletes, de ehhez nem nagyon kellett volna függvény, hiszen a fájlok `Length` tulajdonsága pont ezt az értéket adja vissza. Én azt szeretném, hogy egy fájlgyűjteményt is átadhassak a függvénynek, így több fájlnak az együttes hosszát is megkaphassam. Ehhez alakítsuk át a függvényt:

```
[33] PS C:\old>function fájlhossz ([System.IO.FileInfo[]] $f)
>> {
>>     $hossz = 0
>>     foreach ($file in $f)
>>     {
>>         $hossz += $file.length
>>     }
>>     $hossz
>> }
>>
[34] PS C:\old>fájlhossz (get-childitem)
395962
```

A függvény paraméterdefiníciós részében felkészülök fájltömb fogadására, majd a függvény törzsében egy `foreach` ciklussal végigszaladok az elemeken és összeadogatom a hosszokat. Ez már jó, csak nagyon nem PowerShell-szerű a függvény meghívása. Sokkal elegánsabb lenne, ha a `get-childitem` kimenetét lehetne becsövezni a függvényembe. Nézzük meg, hogy alkalmas-e erre a függvényem átalakítás nélkül?

```
[35] PS C:\old>get-childitem | fájlhossz
0
```

Nem igazán... Merthogy ilyen csövezős esetben a PowerShell nem ad át értéket a „normál” paramétereknek, hanem egy automatikusan generálódó `$input` változónak adja ezt át:

```
[36] PS C:\old>function fájlhossz
>> {
>>     $hossz = 0
>>     foreach ($file in $input)
>>     {
```

```
>>          $hossz += $file.length
>>      }
>>      $hossz
>> }
>>
[37] PS C:\old>get-childitem | fájlhossz
395962
```

Így már majdnem tökéletes a függvényem, de lehet ezt még szebbé tenni! Mi ezzel a gond? Az, hogy ha egy nagyon mély, sok fájlt tartalmazó könyvtárstruktúrára alkalmazom, akkor azt fogjuk tapasztalni, hogy a PowerShell.exe memória-felhasználása jó alaposan felmegy, mire összeáll az `$input` változóban a teljes fájlobjektum lista, és csak utána tud lefutni a `foreach` ciklus. Sokkal optimálisabb lenne, ha már az első fájl átadásával elkezdődhetne a számolás, a memóriában így mindig csak egy fájllal kellene foglalkozni. Erre is van lehetőség, bár - érdekes módon – a help erről nem ír! Merthogy egy függvénynek valójában lehet három elkülönülő végrehajtási fázisú része:

```
function <név> ( <paraméter lista> )
{
     begin {
          <parancsok>
     }
     process {
          <parancsok>
     }
     end {
          <parancsok>
     }
}
```

Lehet tehát egy függvénynek egy „`begin`” része, ami egyszer fut le, a függvény meghívásakor. Lehet egy „`process`” része, ami minden egyes csőelem megérkezésekor lefut, és lehet egy „`end`” része, ami az utolsó csőelem érkezése után fut le.

Alakítsuk át úgy a fájlhossz függvényemet, hogy a `process` szekcióba kerüljön a feldolgozás:

```
[44] PS C:\old>function fájlhossz
>> {
>>      begin {$hossz = 0}
>>      process {$hossz += $_.length}
>>      end {$hossz}
>> }
>>
[45] PS C:\old>dir | fájlhossz
395962
```

Mik a főbb változások? Egyrészt nem kell nekünk ciklust szervezni, mert a csőelemek amúgy is egyesével érkeznek. Viszont a `process` szekcióban nem a `$input` változóval kell foglalkoznunk, hanem a `$_` változóval, az tartalmazza az aktuális csőelemet. Olyannyira, hogy ha így `begin`/`process`/`end` szekciókra bontjuk a függvényt, és a $_ változót használjuk, akkor nem is generálódik `$input`!

Ez olyan fontos, és annyira nincs benne a helpben, hogy kiemelem újra:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Fontos!

Ha a függvényemben külön `begin`/`process`/`end` szekciót használok, akkor nem képződik `$input` változó, de a `process` szekcióban a `$_` változón keresztül érhetem el a bejövő csőelemeket!

Ha egy függvénynek csak `process` szekciója lenne, akkor az ilyen függvényt egy külön kulcsszóval, a `filter` -rel is definiálhatjuk, és akkor egyszerűbb a szintaxis is. Például, ha egy függvényem csak annyit csinálna, hogy a csőben beléérkező számokat megduplázza, akkor az így nézne ki:

```
[46] PS C:\ filter dupláz
>> {
>>     $_*2
>> }
>>
[47] PS C:\ 1,2,5,9 | dupláz
2
4
10
18
```

Látszik, hogy nincs szükség `process` kulcsszóra, nincs felesleges bajuszpár. Viszont nem lehetséges sem `begin`, sem `end` szekció definiálása, így ha ilyenre van szükségünk (valamilyen függvényváltozót kellene inicializálni, vagy az egész folyamat végén kellene még valamit kitenni az outputra), akkor azt nem ússzuk meg a `function` használata nélkül.

Megjegyzés

Fontos tudni, hogy hogyan viselkednek a gyűjtemények a csővezetékben! Láttuk, hogy ha egy egyszerű tömböt adunk át a csővezetéknek, akkor a tömb elemei egymás után kerülnek feldolgozásra. Nézzük, mi történik, ha összetett tömböt adunk át:

```
[22] PS C:\munka> 1,(2,3) | dupláz
2
2
3
2
3
[23] PS C:\munka> (1,(2,3) | dupláz).count
5
```

Az előbb látott dupláz filternek egy két elemű tömböt adok át a [22]-es sorban, amely tömbnek a második eleme egy kételemű tömb. Ennek a kimenete az, amire számítottunk, azaz a megkaptam az 1 duplájá, majd a (2,3) tömböt kétszer. De vajon az eredmény hány elemű? A [23]-as sorban látható, hogy a kimenet 5 elemű. Azaz a csőfeldolgozás szétszedi a tömböket elemekre. Vajon hogyan lehetne elérni, hogy az eredmény háromelemű legyen: 2, (2,3), (2,3)? (Szerintem ez is lehetett volna az alapértelmezett, de valószínű a PowerShell alkotói gyakoribbnak a mostani kimenetet tartották, azaz 2,2,3,2,3. A megoldás már egy korábban már látotthoz hasonló trükkel érhető el:

```
[24] PS C:\munka> 1,,(2,3) | dupláz
2
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
2
3
2
3
[25] PS C:\munka> (1,,(2,3) | dupláz).count
3
```

Azaz a második elem előtt egy extra vesszőt raktam, így már dupla csőelem kifejtés nem történt.

1.7.9 Szkriptblokkok

A paraméterek egyik nagyon érdekes fajtája a szkriptblokk. Enélkül csak nagyon nehézkesen, bonyolult programozással lehetne bizonyos feladatokat függvényekkel megoldani, szkripblokkokkal viszont nagyon egyszerűen. Nézzünk erre egy példát! Szeretnék egy általánosabb függvényt csinálni, az előző `dupláz` helyett, amellyel tetszőleges matematikai műveletet el tudok végezni a csővezetéken beérkező számokon. Ezt a műveletet paraméterként szeretném átadni a függvényemnek. Elsőre elég bonyolultnak tűnhet ilyesmit csinálni, de a szkriptblokkokkal pofonegyszerű:

```
[49] PS C:\>filter művelet ([scriptblock] $s)
>> {
>>     &($s)
>> }
>>
[50] PS C:\>1,2,3 | művelet {$_*3}
3
6
9
[51] PS C:\>10,5,2 | művelet {$_/2}
5
2,5
1
```

A függvényem definíciója látható a [49]-es sorban. Egy „igazi” paramétere van, ami `[scriptblock]` típusú, a másik paramétere majd a csőből jön, és majd a `$_` változón keresztül tudjuk megfogni. Maga a függvény törzse néhány karakter: „`&($s)`”, azaz csak annyi történik, hogy a paraméterként átadott szkriptet meghívjuk az „`&`” operátorral. És ezzel el is készült a nagy tudású, univerzális matematikai függvényünk! Persze a használatához tudni kell, hogy olyan kifejezést kell neki átadni paraméterként, ami a `$_` változóval játszadozik, de amúgy tényleg egyszerű a dolog: az [50]-es sorban háromszoroz a függvényem, az [51]-es sorban pedig felez.

1.7.9.1 Anonim függvények

Valójában egy szkriptblokk nem csak paraméterként szerepelhet, hanem név nélküli függvényként is felfoghatjuk őket. Nézzük a legegyszerűbb csővezetékkel végezhető műveletet, adjuk vissza egyesével, változtatás nélkül a csőbe bemenő elemeket. Ugye megtanultuk, hogy a csőelemekhez a `process` szekcióban férünk hozzá a `$_` változón keresztül. Első próbálkozás:

```
[85] PS C:\> 1, "kettő", 3.04 | process {$_}
Get-Process : A positional parameter cannot be found that accepts argument '$_
'.
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
At line:1 char:27
+ 1, "kettő", 3.04 | process <<<<  {$_}
    + CategoryInfo          : InvalidArgument: (:) [Get-Process], ParameterBi
   ndingException
    + FullyQualifiedErrorId : PositionalParameterNotFound,Microsoft.PowerShel
   l.Commands.GetProcessCommand
```

Ez nem jó, a csőjel (|) után valami kifejezést vár, valami végrehajtható dolgot, márpedig a `process` kulcsszó, nem végrehajtható kifejezés, ezért a PowerShell itt ezt úgy értelmezi, hogy az a `Get-Process` álneve, annak pedig nincs `$_` paramétere. Segítsünk neki, csináljunk belőle szkriptet, pontosabban fogalmazva szkriptblokkot, amiben már van értelme a `process` kulcsszónak:

```
[86] PS C:\> 1, "kettő", 3.04 | {process {$_}}
Expressions are only allowed as the first element of a pipeline.
At line:1 char:34
+ 1, "kettő", 3.04 | {process {$_}} <<<<
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordE
   xception
    + FullyQualifiedErrorId : ExpressionsMustBeFirstInPipeline
```

Ez sem lett sokkal jobb, merthogy – érdekes módon – a szkriptblokk sem végrehajtható! Az majdnem ugyanolyan adatszerűség, mint a változó, vagy akár egy szám. Segítsünk tovább neki, tegyünk elé egy végrehatás-operátort:

```
[87] PS C:\> 1, "kettő", 3.04 | &{process {$_}}
1
kettő
3,04
```

Na, itt már jól lefutott. Kis átalakítással ugyanazt a funkcionalitást el tudjuk érni, mint az előző alfejezet `dupláz` filteré:

```
[88] PS C:\> 1, "kettő", 3.04 | &{process {$_*2}}
2
kettőkettő
6,08
```

Azaz az ilyen szkriptblokkokat felfoghatjuk úgy is, mint anonim függvények. Ilyeneket akkor érdemes használni, ha nem akarom többször felhasználni a függvényt, hiszen akkor minek neki nevet adni?

1.7.10 Függvények törlése, módosítása

Ha egy függvényre nincs szükségem, akkor törölhetem is. Van ugye egy speciális `PSDrive`-om, a `function:`, ez tartalmazza az összes függvényt és ennek különböző elemeit ugyanúgy törölhetem, mint a fájlokat:

```
[33] PS I:\>Get-ChildItem function:

CommandType     Name                                Definition
-----------     ----                                ----------
...
Function        belső                               param($b) Write-Host "Bels...
```

```
Function        külső                           param($p) function belső (...
Function        függvénytár                     function global:első...
Function        első                            "első"
Function        második                         "második"
Function        titkos                          function private:egymegegy...
Function        egymegegy                       "1+1=2"
```

A lista végén vannak az általam definiált függvények, előtte „gyári” függvények vannak, amelyeket mindjárt átnézünk.

Ha meg szeretnék szabadulni például az `egymegegy` függvényemtől, akkor használhatom a `remove-item` cmdletet:

```
[34] PS I:\>Remove-Item function:egymegegy
```

Ha már itt tartunk, akkor vajon PSDrive-kezelő cmdletekkel is létrehozhatok-e függvényt? Természetesen igen, nézzük hogyan működik a `new-item`:

```
[35] PS C:\> new-item -path function: -name fv1 -value {"Fájlként new-item-m
el"}

CommandType     Name                            Definition
-----------     ----                            ----------
Function        fv1                             "Fájlként new-item-mel"


[36] PS C:\> fv1
Fájlként new-item-mel
```

Módosítani a függvényeimet vagy megismételt függvénydefinícióval tudom, vagy a szintén a PSDrive-kezelő `set-item` cmdlettel:

```
[37] PS C:\> Set-Item -path function:fv1 -Value {"Módosítás set-item-mel"}
[38] PS C:\> fv1
Módosítás set-item-mel
```

Hát ez sikerült! Akkor nézzük is meg, hogy amúgy mire képes a `set-item`, és az hogyan alkalmazható a függvényekre! Érdekes módon a helpben a szintaxisában nincsen `-options` paraméter:

```
[53] PS C:\old> (get-help set-item).syntax

Set-Item [-path] <string[]> [[-value] <Object>] [-force] [-include <string[
]>] [-exclude <string[]>] [-filter <string>] [-passThru] [-credential <PSCr
edential>] [-whatIf] [-confirm] [<CommonParameters>]
Set-Item [-literalPath] <string[]> [[-value] <Object>] [-force] [-include <
string[]>] [-exclude <string[]>] [-filter <string>] [-passThru] [-credentia
l <PSCredential>] [-whatIf] [-confirm] [<CommonParameters>]
```

Azonban a példákban már látható, hogy van:

```
[54] PS C:\old> (get-help set-item).examples

...

-------------------------- EXAMPLE 4 --------------------------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
C:\PS>set-item -path function:prompt -options "AllScope,ReadOnly"


This command sets the AllScope and ReadOnly options for the "prompt" functi
on. This command uses the Options dynamic parameter of the Set-Item cmdlet.
 The Options parameter is available in Set-Item only when you use it with t
he Alias or Function provider.
```

A magyarázat is ott van legalul, hogy ez egy dinamikus paraméter, és csak az `Alias` és a `Function` PSDrive használatakor létezik. Ezzel lehet beállítani a függvényem láthatóságát (scope) és – hasonlóan a változókhoz – azt is, hogy a függvényem csak olvasható (azaz nem lehet felülírni), vagy konstans (még a `-force` kapcsolóval sem lehet felülírni).

Megjegyzés

Sajnos elég esetleges ezen dinamikus paraméterek fellelése a helpben, a cmdletek mellett. Talán eredményesebb, ha a providerek irányából keressük meg ezeket. Például nézzük, hogy az `Alias:` providernek milyen dinamikus paraméterei vannak:

```
[63] PS C:\> get-help -Category provider -name alias
...
DYNAMIC PARAMETERS
    -Options <System.Management.Automation.ScopedItemOptions>
        Determines the value of the Options property of an alias.

            None
                No options. "None" is the default.

            Constant
                The alias cannot be deleted and its properties cannot be ch
                anged. Constant is available only when you are creating an
                alias. You cannot change the option of an existing alias to
                 Constant.

            Private
                The alias is visible only in the current scope (not in chil
                d scopes).

            ReadOnly
                The properties of the alias cannot be changed, except by us
                ing the Force parameter. You can use Remove-Item to delete
                the alias.

            AllScope
                The alias is copied to any new scopes that are created.


        Cmdlets Supported: New-Item, Set-Item
```

Azaz több lehetőségünk is van arra, hogy milyen módon kezeljük a függvényeinket, álneveinket.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

Kézenfekvőnek tűnhetne, hogy a függvényobjektum `definition` tulajdonságát írjuk át a függvény módosításához. Ez azonban nem megy, mert az *Read Only* attribútum, azaz nem írható át:

```
[41] PS C:\> (get-item function:fv1).definition = {"Módosítás a definition t
ulajdonságon keresztül"}
"Definition" is a ReadOnly property.
At line:1 char:25
+ (get-item function:fv1).d <<<< efinition = {"Módosítás a definition tulaj
donságon keresztül"}
```

A fenti példában látszik, hogy hibát a `definition` attribútum közvetlen módosításakor kaptam.

1.7.11 Gyári függvények

A függvényeken belül akkor most nézzük meg azokat, amelyek már az alap PowerShell környezetben is benne vannak. Ehhez nem kell a helpet hosszasan bújnunk, mivel – mint ahogy korábban már láthattuk – egy külön meghajtó van létrehozva a függvények elérésére és listázásához, a „`function:`” PSDrive:

```
PS C:\> dir function:

CommandType     Name                                Definition
-----------     ----                                ----------
Function        prompt                              'PS ' + $(Get-Location) + ...
Function        TabExpansion                        ...
Function        Clear-Host                          $spaceType = [System.Manag...
Function        more                                param([string[]]$paths);  ...
Function        help                                param([string]$Name,[strin...
Function        man                                 param([string]$Name,[strin...
Function        mkdir                               param([string[]]$paths); N...
Function        md                                  param([string[]]$paths); N...
Function        A:                                  Set-Location A:
Function        B:                                  Set-Location B:
Function        C:                                  Set-Location C:
...
Function        Z:                                  Set-Location Z:
```

Ezek közül a „*betű*:” formátumúak nagyon egyszerűek, ezek csak annyit csinálnak, hogy az aktuális meghajtónak beállítják az adott betűjellel jelzett meghajtót.

A `prompt` függvény automatikusan lefut, ha a konzolon új sort nyitunk. Ezt a függvényt is tetszőlegesen átszerkeszthetjük, például én is itt a könyvben, hogy jobban lehessen hivatkozni a begépelt kódsorokra, beraktam egy sorszámot is szögletes zárójelek közé, amit ráadásul mindig növelek eggyel.

Szintén egy függvénynek, a `TabExpansion`-nek köszönhetjük a TAB-ra történő parancs-kiegészítést, ami akkor fut le automatikusan, amikor megnyomjuk a TAB billentyűt. Ezt is testre szabhatjuk, ha nem találjuk elég okosnak.

A többi függvénynek is meg lehet nézni a definíciós részét, vagy akár újra lehet definiálni őket az előző fejezetben megismert módon.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

A „gyári” függvények átdefiniálása nem tartós, azaz ha becsukjuk a PowerShell ablakot, majd újra megnyitjuk, akkor visszaáll az eredeti függvénydefiníció. Tartóssá változtatásainkat úgy tehetjük, ha profil fájlban definiáljuk újra ezeket a függvényeket. Profilokról a *2.1.1 Profilok* fejezetben lesz szó.

A könyv tanfolyami felhasználása nem engedélyezett!

1.8 Szkriptek

A függvények után nézzük a szkripteket. Amint korábban már említettük a PowerShell előre elkészített (fájlba mentett) parancssorozatait hívjuk szkriptnek, amelyeket közvetlenül is futtathatjuk, hasonlóan a függvényekhez. Ebben a fejezetben azokkal a szerkezetekkel fogunk megismerkedni, amelyek bár a függvényeknél is előfordultak, de használatuk elsősorban szkriptekben szokásos, de természetesen minden begépelhető közvetlenül a parancssorba is.

1.8.1 Szkriptek engedélyezése és indítása

Készítsük is el mindjárt első szkriptünket! Olyan fájlt kell készítenünk, amelynek kiterjesztése `ps1`, ez a PowerShell szkriptek alapértelmezett kiterjesztése, még a 2.0-ás verziónál is. Egyszerűbb esetben még a Notepadre sincs szükség, létrehozhatjuk a fájl közvetlenül a PowerShellből is:

```
[1] PS C:\> "'Hurrá! Fut az első szkriptünk.'" | Out-File elso.ps1
```

Hát ennyi. Látható, hogy nincs szükség túl sok cicomára, amit a parancssorba beírhatnánk, az minden további nélkül jó szkriptnek is. Indítsuk is el gyorsan a szkriptet:

```
[4] PS C:\> C:\elso.ps1
File C:\elso.ps1 cannot be loaded because the execution of scripts is disabled
 on this system. Please see "get-help about_signing" for more details.
At line:1 char:12
+ C:\elso.ps1 <<<<
    + CategoryInfo          : NotSpecified: (:) [], PSSecurityException
    + FullyQualifiedErrorId : RuntimeException
```

Hoppá! A PowerShell olyan szkriptkörnyezet, amelyben nem engedélyezett a szkriptek futtatása?! Természetesen ez csak a „secure-by-default” jegyében beállított alapértelmezés, a szkriptekkel szemben tanúsított viselkedést a `Set-ExecutionPolicy` cmdlet segítségével szabályozhatjuk az alábbi állapotok közül a megfelelő kiválasztásával:

Paraméter	Hatása
Restricted	Nem engedélyezett semmilyen szkript futtatása (alaphelyzet)
AllSigned	Csak elektronikusan aláírt szkriptek futtathatók
RemoteSigned	Külső forrásból letöltött szkripteknek kell aláírtaknak lenniük (hálózat, internet), helyi szkriptek aláírás nélkül is futtathatók
Unrestricted	Korlátozás nélkül futtathatók a szkriptek, az internetről letöltött szkriptek esetén megerősítést kér
Bypass	Korlátozás nélkül futtathatók a szkriptek, megerősítést sem kér
Undefined	Az adott futtatási környezetben beállított végrehajtási házirendet figyelmen kívül hagyja, így a magasabb szinten beállított házirend, vagy (ha ilyen nincs) a Group Policy segítségével beállított házirend jut érvényre

Ahhoz, hogy mi a saját aláírás nélküli szkriptjeinket futtatni tudjuk, a RemoteSigned szintet javasolom beállítani:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

Megjegyzés

A végrehajtási házirend beállításához helyi rendszergazda jogosultság szükséges, illetve Windows Vista, vagy annál újabb operációs rendszerek esetében ezen kívül a PowerShell alkalmazást is rendszergazda jogkörrel kell futtatni, hogy ez a parancs lefusson. Ha nincs valakinek ilyen jogköre, akkor ennek a házirendnek is beállíthatunk a `-Scope` paraméterhez egy hatósugarat:

`Process`: csak az adott PowerShell ablakban állítja be a végrehajtási házirendet, nem igényel rendszergazda jogosultságot

`CurrentUser`: csak az adott felhasználóra állítja be a végrehajtási házirendet, nem igényel rendszergazda jogosultságot

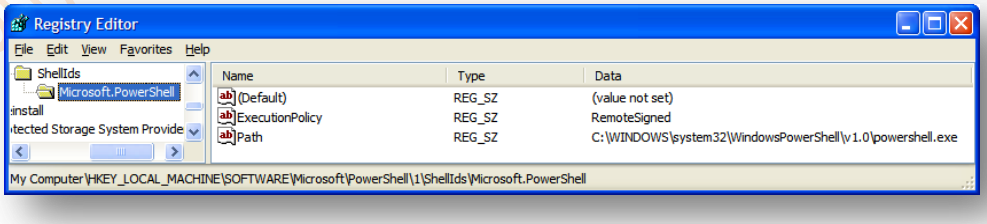
`LocalMachine`: alaphelyzet szerinti beállítás, az adott gép minden felhasználójára állítja be a végrehajtási házirendet

Nézzünk erre egy példát:

```
[65] PS C:\> Set-ExecutionPolicy -ExecutionPolicy restricted -Scope process

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
Changing the execution policy might expose you to the security risks described
 in the about_Execution_Policies help topic. Do you want to change the
execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
[66] PS C:\> . .\munka\converthashtocollection.ps1
File C:\munka\converthashtocollection.ps1 cannot be loaded because the executi
on of scripts is disabled on this system. Please see "get-help about_signing"
for more details.
At line:1 char:2
+ . <<<<  .\munka\converthashtocollection.ps1
    + CategoryInfo          : NotSpecified: (:) [], PSSecurityException
    + FullyQualifiedErrorId : RuntimeException
```

A fenti példában pont szigorítottam az adott PowerShell ablak futtatási házirendjét, minek hatására egy szkript futtatása hibára futott.

A globális biztonsági beállítást a registry-n keresztül is beállíthatjuk, például a számítógépre:



32. ábra Execution Policy helye a registry-ben

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Vagy beállíthatjuk ezt az értéket központilag is Group Policy segítségével is. Ehhez letölthető a Microsoft downloads oldaláról az „*Administrative templates for Windows PowerShell*” adm fájl.

Azt, hogy milyen végrehajtási házirendek hatnak éppen a munkamenetünkre, a `Get-ExecutionPolicy` cmdlettel tudjuk lekérdezni. Méghozzá a `-List` kapcsolóval az összes lehetséges hely fel lesz sorolva, ahonnan kaphatunk ilyen jellegű házirendet:

```
PS C:\> Get-ExecutionPolicy -List

                                Scope                         ExecutionPolicy
                                -----                         ---------------
                          MachinePolicy                               Undefined
                             UserPolicy                               Undefined
                                Process                               Undefined
                            CurrentUser                               Undefined
                           LocalMachine                            RemoteSigned
```

Állítsuk be a megfelelő biztonsági szintet a PowerShell cmdlet segítségével (legalább `RemoteSigned`) és próbáljuk meg ismét az indítást! (Természetesen csak rendszergazda jogosultságok birtokában tudjuk ezt megtenni.) Az előbb talán túl sokat is gépeltünk a fájl a c: gyökerében van, elég lesz csak ennyi:

```
[5] PS C:\> elso.ps1
The term 'elso.ps1' is not recognized as the name of a cmdlet, function, scrip
t file, or operable program. Check the spelling of the name, or if a path was
included, verify that the path is correct and try again.
At line:1 char:9
+ elso.ps1 <<<<
    + CategoryInfo          : ObjectNotFound: (elso.ps1:String) [], CommandNo
   tFoundException
    + FullyQualifiedErrorId : CommandNotFoundException


Suggestion [3,General]: The command elso.ps1 was not found, but does exist in t
he current location. Windows PowerShell doesn't load commands from the current
location by default. If you trust this command, instead type ".\elso.ps1". See
"get-help about_Command_Precedence" for more details.
```

Hát ez nem elég! Régi jó UNIX / Linux szokás szerint a PowerShell nem próbálja az aktuális mappában megtalálni a megadott futtatható állományt, így teljes útvonalat (relatív vagy abszolút) kell megadnunk. Tehát például az alábbi módon indíthatjuk el végre a korábban létrehozott szkriptet:

```
[8] PS C:\> .\elso.ps1
Hurrá! Fut az első szkriptünk.
```

Ha az aktuális mappában nem is, de a PATH környezeti változóban felsorolt mappákban természetesen próbálkozik a PowerShell, ilyen esetben nem szükséges útvonalat megadnunk.

Sajnos még mindig nem vagyunk teljesen készen, mivel ha szkriptünk útvonala szóköz karaktert is tartalmaz, újabb problémába ütközünk. Hozzunk létre egy újabb szkriptet a 'C:\Könyvtár szóközzel\második.ps1' mappában!

```
[4] PS C:\> "'Hurrá! Fut a 2. szkriptem.'" | Out-File 'C:\Könyvtár szóközzel\má
sodik.ps1'
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Rutinosabb versenyző persze nem is próbálkozik az idézőjelek nélküli indítással, ez azonban ebben az esetben kevés, mivel a parancs így nem parancs és nem is útvonal, csak egy közönséges karakterlánc:

```
[6] PS C:\> 'C:\Könyvtár szóközzel\második.ps1'
C:\Könyvtár szóközzel\második.ps1
```

Be kell vetnünk a korábban már használt „futtató” karaktert (`&`) az alábbi módon:

```
[7] PS C:\> & 'C:\Könyvtár szóközzel\második.ps1'
Hurrá! Fut a 2. szkriptem.
```

Megjegyzés

Ha a beviteli sor elején kezdünk el olyan elérési utat gépelni, amelyben szóköz karakter van, akkor a TAB billentyű lenyomására a PowerShell 2.0 a könyvtár nevének kiegészítésekor automatikusan beilleszti a sor elejére a végrehatás operátorát és az aposztrófot.

Végül foglaljuk össze röviden mi is szükséges ahhoz, hogy PowerShell szkripteket futtassunk:

- Be kell állítanunk a megfelelő végrehajtási szabályt (execution policy). Alapértelmezés szerint a PowerShell nem futtat le semmiféle szkriptet, akárhogy is adjuk meg az útvonalat.
- A szkript indításához adjuk meg annak teljes útvonalát, illetve ha a fájl az aktuális mappában van, használjuk a `.\` jelölést. Útvonal nélkül is elindíthatjuk a szkriptet, ha olyan mappában van, amely szerepel a Windows keresési útvonalán (PATH környezeti változó).
- Ha az útvonal szóközöket tartalmaz, tegyük idézőjelek közé és írjuk elé a futtató karaktert (`&`).

A korábbi szokásokkal (cmd, vbs, stb.) ellentétben (természetesen szintén biztonsági okokból) maga a szkriptfájl közvetlenül nem indítható el a shellen kívül, mivel a `ps1` kiterjesztésű fájlokat alapértelmezés szerint a notepad.exe nyitja meg. Hogy lefuttathassuk szkriptünket akár egy logonszkriptből, a powershell.exe-t kell elindítanunk (keresési útvonalon van), amelynek paraméterként adhatjuk át futtatandó szkriptfájl nevét. A cmd.exe-ből például az alábbi módon indíthatjuk el a fenti szkriptet (a `-noexit` kapcsoló hatására a powershell.exe nem áll le a szkript lefuttatása után, megfigyelhető, hogy a PowerShellre jellemző promptot kapunk vissza):

```
C:\>powershell -noexit c:\elso.ps1
Hurrá! Fut az első szkriptünk.
PS C:\>
```

Próbáljuk meg hasonló módon elindítani a `masodik.ps1` szkriptet is! Nyilván itt is kell a futtató karakter:

```
C:\Users\Administrator>powershell -noexit &"c:\Könyvtár szóközzel\második.ps1"
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

[1] PS C:\>
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ugyan látható a promptból, hogy a cmd shell átalakult PowerShellé, de sajnos a szkriptünk így sem futott le. Még egy trükkre van szükség, a cmd.exe elől el kell rejteni a PowerShellnek szóló `&` jelet egy újabb, külső idézőjelpárral:

```
C:\Users\Administrator>powershell -noexit "& 'c:\Könyvtár szóközzel\második.ps1
'"
Hurrá! Fut a 2. szkriptem.
[1] PS C:\>
```

Megjegyzés

Itt most fontos, hogy a külső idézőjel a macskaköröm legyen, a belső pedig az aposztróf, egyéb módokon nem fut le helyesen.
Ha logonszkriptekben szerepeltetjük ezt a parancsot, akkor ne használjuk a `–noexit` paramétert.

1.8.2 Változók kiszippantása a szkriptekből (dot sourcing)

A függvényeknél már látott, de a scripteknél még gyakrabban alkalmazott dotsourcing lehetőséget elevenítsük fel ismét. Ehhez hasonló szolgáltatással korábban (cmd, VBScript, stb.) egyáltalán nem találkozhattunk. Készítsük el a következő szkriptet, akár a Notepad segítségével:

```
$a = "egyik-"
$b = "másik"
$c = $a + $b
"`$c = $c"
```

Mentsük el a fájlt, mondjuk a `c:\scripts` mappába `source.ps1` néven! Ezután futtassuk le az elkészült szkriptet, ami kiírja a `$c` változó tartalmát, majd a szkript lefutása után nézzük meg, milyen érték van a `$c` változóban! A helyes tipp az, hogy semmilyen. Ez így természetes, a szkript és a konzol külön munkamenetnek tekintendő, a szkript a konzol „gyerekkörnyezete”, a konzol ezért nem éri el a gyereke adatait.

```
PS C:\> c:\scripts\source.ps1
$c = egyik-másik
PS C:\> $c
PS C:\>
```

Eddig tehát tulajdonképpen semmi meglepőt nem tapasztaltunk, inkább az lett volna furcsa, ha nem így van. Ha azonban a korában látott „dotsourcing” (*1.7.5 Függvények láthatósága, „dotsourcing”* fejezet) módon indítjuk el szkriptünket, akkor a függvényeknél már látottaknak megfelelően a változó elérhetővé válik:

```
PS C:\> . c:\scripts\source.ps1
$c = egyik-másik
PS C:\> $c
egyik-másik
```

A konzol munkamenet, a globális scope „tudja”, hogy mi volt a változó értéke a szkriptben, ami már réges-rég véget ért. Ez a jelenség a függvényeknél már megismert „dotsourcing”, amelynek hatására a szkript

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

valamennyi változója az őt meghívó környezet változójává alakult, jelen esetben globálissá vált, vagyis minden munkamenet (így a konzol és más szkriptek is) elérhetik, felhasználhatják és megváltoztathatják a bennük tárolt értékeket még akkor is, amikor az őket létrehozó munkamenet (vagyis a szkript) már bezárult.

Talán ez már nyilvánvaló, hogy milyen hasznos ez a lehetőség, hiszen így a szkript által lekérdezett, létrehozott adatszerkezetek (például egy adatbázisból, vagy az Active Directoryból származó adathalmaz) a szkript lefuttatása után interaktív üzemmódban is listázhatók, feldolgozhatók. Vagyis egyszerre élvezhetjük a szkript- és az interaktív üzemmód előnyeit. Ha egy feladat egyik részét inkább szkriptként érdemes elkészíteni, de más részeknél egyszerűbb az interaktív üzemmód használata, akkor bátran használhatjuk ezt a lehetőséget, mindent olyan módon végezhetünk el, ahogy az a legkedvezőbb.

Még egy dolog jegyzendő meg itt. Nézzünk egy olyan szkriptet, ami tartalmaz még egy függvénydefiníciót is:

```
param ($x)
$a = "Ez a script változója"
function belsőfv ($y)
{
     $a = "Ez a fv változója"
     $x
     $y
     $a
     $script:a
     $global:a
}

belsőfv "Y értéke"
```

Ebben a szkriptben minden jó van: átadott paraméter, belső változó, függvénydefiníció, a függvénynek saját változója. Ráadásul „a” nevű változó mindenütt lehet. Definiálok tehát egy „külső” `$a` változót és futtatom a szkriptet egy átadott paraméterrel:

```
PS C:\> $a = "Globális A"
PS C:\> .\_munka\scriptscope.ps1 "Ez az X értéke"
Ez az X értéke
Y értéke
Ez a fv változója
Ez a script változója
Globális A
```

Látható, hogy minden szint `$a` változóját meg tudtam jeleníteni. A szkriptnek tehát van egy külön érvényességi területe, ez pedig a `script`.

1.8.3 Paraméterek átvétele és a szkript által visszaadott érték

Szkriptjeinknek is lehetnek paraméterei, melyeket – hasonlóan a függvényekhez - az `$args` változóban is elérhetjük. Emlékeztetőül, az `$args` egy tömb, amelybe tetszés szerinti típusú értékek kerülhetnek, a megjelenő típus a beírt értéktől függ, ezért kényesebb esetekben célszerű lehet a paraméterek számán kívül azok típusát is ellenőrizni a további tényked előtt, vagy „fejlett” szkripteket készíteni a *2.4 Fejlett függvények – script cmdletek* fejezet szerint. Az alábbi szkriptben csak a megfelelő számú paraméter meglétét ellenőrizzük:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
if ($args.Length -ne 3)
{
     Write-Error "A szkript csak 3 paraméterrel indítható!"
     return "Hibás futás!"
}
Write-Host $args.GetType()
foreach($arg in $args)
{
     Write-Host $arg
     Write-Host $arg.GetType()

}
return "Jó futás!"
```

Ha a paramétereket tároló tömb nem három elemből áll, akkor hibaüzenetet írunk ki, és a `return` kulcsszó után megadjuk a szkript által visszaadott értéket (a `return` utáni rész már nem fut le). Paraméter nélküli indítási kísérlet esetén a következő hibaüzenet jelenik meg:

```
PS C:\> ./parameter.ps1
C:\parameter.ps1 : A szkript csak 3 paraméterrel indítható!
At line:1 char:15
+ ./parameter.ps1 <<<<
Hibás futás!
```

Ha megvan mindhárom paraméter, akkor visszaírjuk azok típusát és értékét a képernyőre, a szkript pedig egy másik értéket fog visszaadni.

```
PS C:\> ./parameter.ps1 egy kettő három
System.Object[]
egy
System.String
kettő
System.String
három
System.String
Jó futás!
```

A visszaadott érték a fenti esetekben egyszerűen a képernyőre került, de ez nem igazán erre való. Az érték segítségével a hívó szkript kaphat információt az elindított szkript belsejében történtekről. Természetesen semmi akadálya nincsen több érték visszaadásának sem (bár csak egyetlen `return` futhat le), szkriptünk egy tömb elemeinek képében tetszőleges számú és típusú értéket is visszaadhat.

```
PS C:\> $vissza = ./parameter.ps1 egy kettő három
System.Object[]
egy
System.String
kettő
System.String
három
System.String
PS C:\> $vissza
Jó futás!
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Természetesen nem csak az `$args` változó áll rendelkezésünkre, hanem használhatunk explicit paraméterdefiníciót is. Miután a szkriptnek nincs saját, „belső” neve, hanem csak a fájlnak, amibe rakjuk, ezért a paramétereit se tudjuk a nem létező neve mellett felsorolni, csak külön `param` kulcsszó megadásával:

```
param ($a, $b)
$a / $b
```

Elmentetem a fenti, nagyon bonyolult szkriptet „osztás.ps1” néven, és meg is hívhatom, hasonlóan egy függvényhez:

```
[6] PS C:\old>.\osztás.ps1 20 4
5
[7] PS C:\old>.\osztás.ps1 -b 3 -a 27
9
```

Akár a hely szerinti, akár a név szerinti paraméterátadás is működik. Természetesen a szkript által visszaadott eredményt felhasználhatjuk értékadás során is, megint csak ugyanúgy, mintha függvény lenne:

```
[8] PS C:\old>$eredmény = .\osztás 50 11
[9] PS C:\old>$eredmény
4,54545454545455
```

1.8.4 Szkriptek írása a gyakorlatban

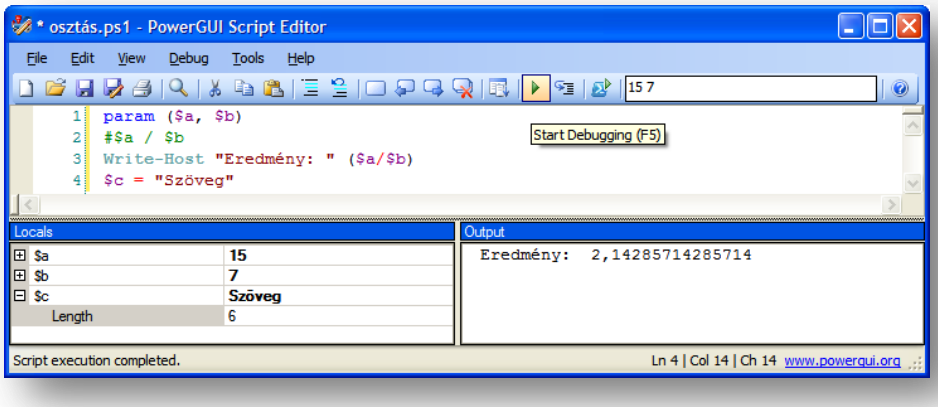
Szkripteket a gyakorlatban ritkán írunk magával a PowerShell konzolablakon, még csak nem is a Notepad az ideális eszköz, hanem valamilyen szkriptszerkesztő. A PowerShell 2.0 beépítet szkriptszerkesztője (ld. a *1.2.12 A grafikus PowerShell felület – Integrated Scripting Environment* fejezet), vagy a *2.2.1 PowerGUI, PowerGUI Script Editor* fejezetben már említett PowerGUI Script Editora talán a legegyszerűbb ilyen jellegű eszközök, amelyek ráadásul pont eleget tudnak ahhoz, hogy hatékonyan használhassuk őket. Ráadásul mindkettő ingyenes!

1.8.4.1 PowerGUI Script Editor

A korábbi „osztás” szkriptpéldát kicsit még kiegészítettem, mindez így néz ki a PowerGUI Script Editorban:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



33. ábra Szkriptem a PowerGUI Script Editorában

Nyomtatásban nem biztos, hogy színes, de a képernyőn látható a fenti ábra színeinek kavalkádja: külön színnel jelzi a program a kulcsszavakat, cmdleteket, változókat, megjegyzéseket, operátorokat.

Az eszközsorban látható „Start Debugging (F5)" gombbal lehet futtatni ott helyben a szkriptet. Az eszközsorban, a beviteli mezőben lehet paramétereket is átadni (a képen ez most „`15 7`"). A futás során generálódó output kikerül a jobb alsó ablakrészbe és a belső változók tartalma is azonnal megtekinthető a baloldali részben. A jobb szemléltethetőség kedvéért még egy felesleges `$c` változót is felvettem, hogy látható legyen az, ahogyan a „Locals" részben a változókban tárolt objektumok tulajdonságai is azonnal elérhetők és kiolvashatók, nem kell állandóan `get-member`-t futtatni.

1.8.4.2 Megjegyzések, kommentezés (#, <# #>)

A fájlba mentett parancssorozatainkat érdemes kommentekkel ellátni, hogy ha pár hónap után nyitjuk meg őket, akkor is értsük, hogy mit csinál a szkript, a paraméterek, változók hogyan használhatók.

A komment jele a # jel.

```
[10] PS C:\old>$a = 5 #értékadás
[11] PS C:\old>$a
5
```

A fejlesztés során a megjegyzéseket, kommenteket arra is felhasználhatjuk, hogy különböző kódváltozatok tesztelésekor az egyik változatot „kikommentezzük", míg a másikat teszteljük, aztán meg cserélünk, és megnézhetjük, hogy a másik változat hogyan működik. Vagy az is gyakori, hogy a szkriptünk kifejlesztése során jóval több átmeneti változó és állapot értékét íratjuk ki, míg a végleges verziónál ezt már nem szeretnénk. Ilyenkor szintén jól jön a „kikommentezési" lehetőség. Kitörölni nem akarjuk ezeket a kiíratási lehetőségeket, mert hátha jól jönnek majd a szkript továbbfejlesztésekor.

Ezt a gyors ki-be kommentezést jól támogatja a PowerGUI Script Editora, az eszközsorban található gombokkal egy kattintással tudjuk ezt ki-be kapcsolni, ráadásul több sor kijelölésével is működik.

A # jel hatása csak az adott sorra érvényes a sor végéig. Ha többsoros kommenteket szeretnénk beszúrni, akkor vagy minden sort külön ellátunk ilyen jellel, vagy használhatjuk a PowerShell 2.0 új lehetőségét a többsoros megjegyzés jelölőt (`<# #>`):

```
<# Hosszú megjegyzés,
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
ami egy új lehetőség a
PowerShell 2.0-ban. #>
$a = 1
```

Majd a *2.4.7 Súgó készítése* fejezetben látni fogjuk, hogy a megjegyzésekkel profi súgót is készíthetünk szkriptjeinkhez, függvényeinknek.

1.8.5 Adatbekérés (Read-Host)

A szkriptek esetében gyakori, hogy egy kis interakciót várunk el tőlük, azaz ne kelljen mindig parancssorban felparaméterezni őket, hanem kérdezzék meg tőlünk, hogy mit szeretnének. A következő példában egy kis interaktív vezérlést mutatok be egy kis menüstruktúra segítségével:

```
do
{
     Write-Host "============================================================"
     Write-Host "0 - Folyamatok listázása"
     Write-Host "1 - Szolgáltatások listázása"
     Write-Host "2 - Folyamat indítása"
     Write-Host "3 - Folyamat leállítása"
     Write-Host "9 - Kilépés"
     $v = Read-Host
     Write-Host "============================================================"
     switch ($v)
     {
           "0" {Get-Process}
           "1" {Get-Service}
           "2" {&(Read-Host "Folyamat neve")}
           "3" {Stop-Process -name (&Read-Host "Folyamat neve")}
           "9" {break}
           default {Write-Host "Érvénytelen billentyű!";break  }
     }
}    while ($v -ne "9")
```

A szkript egy jó kis hátul tesztelő `do-while` ciklus, ami egészen addig fut, amíg a középtájon található `Read-Host` cmdletnek nem adunk át egy 9-est.

Ha 0, 1, 2, 3 értékek valamelyikét adjuk be neki, akkor egy `switch` kifejezés a megfelelő PowerShell cmdletet futtatja le. Ráadásul bizonyos parancsok még további paramétert igényelnek, így azokat egy újabb `Read-Host` kéri be.

A `Read-Host` nem billentyű leütésére vár, hanem komplett sort kér be Enter leütéséig, és addig vár, amíg ezt meg nem kapja. A fenti példában látható, hogy nem muszáj a `Read-Host` kimenetét változóba tölteni, azt azon melegében is felhasználhatjuk.

Például a „2”-es választásával rögtön meghívjuk azt, amit begépeltünk:

```
&(Read-Host "Folyamat neve")
```

Ugye itt megint használjuk a „végrehajtatás” operátorát, az `&` jelet.

A `Read-Host`-nak van egy szöveg paramétere, ezt teszi fel kérdésként a sor beadására való várakozás előtt.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.8.6 Szkriptek digitális aláírása

Láttuk a fejezet elején, hogy a szkriptek futtatását engedélyezni kell. Ha a legbiztonságosabb futtatási házirenddel dolgozunk, akkor csak olyan szkript futhat, amelyik digitálisan alá van írva, és az aláírást igazoló tanúsítványt az adott gép elismeri biztonságos tanúsítványnak. Na de honnan lesz nekünk ilyen tanúsítványunk?

Az első lehetőség, hogy van már kiépített PKI infrastruktúránk, amelyben van „code signing” típusú tanúsítványt kiosztani képes CA kiszolgáló. Innen igényelni kell ilyen tanúsítványt, és ezzel fogunk tudni dolgozni a később leírt módon.

A problematikusabb helyzet az, amikor nincs ilyen PKI infrastruktúránk. Például én otthon szeretnék aláírt szkripteket létrehozni, hogy tesztelhessem a működésüket. Ebben az esetben létrehozhatok un. önaláírt tanúsítványt, amelyet csak azok a gépek fognak elismerni hitelesnek, ahova manuálisan telepítem azokat.

Nézzünk ennek lépéseit! Honnan szedek én önaláírt tanúsítványt? Direkt ilyen tesztelési célokra van a .NET Framework Software Development Kit-ben (letölthető: http://msdn2.microsoft.com/en-us/netframework/aa731542.aspx) egy `makecert.exe` segédprogram, amely erre képes.

Ez egy parancssori eszköz, nézzük meg a meghívásának paramétereit:

```
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0>"C:\Program Files\Micros
oft Visual Studio 8\SDK\v2.0\Bin\makecert.exe" -!
Usage: MakeCert [ basic|extended options] [outputCertificateFile]
Extended Options
 -sc  <file>          Subject's certificate file
 -sv  <pvkFile>       Subject's PVK file; To be created if not present
 -ic  <file>          Issuer's certificate file
 -ik  <keyName>       Issuer's key container name
 -iv  <pvkFile>       Issuer's PVK file
 -is  <store>         Issuer's certificate store name.
 -ir  <location>      Issuer's certificate store location
                         <CurrentUser|LocalMachine>.  Default to 'CurrentUser
'
 -in  <name>          Issuer's certificate common name.(eg: Fred Dews)
 -a   <algorithm>     The signature algorithm
                         <md5|sha1>.  Default to 'md5'
 -ip  <provider>      Issuer's CryptoAPI provider's name
 -iy  <type>          Issuer's CryptoAPI provider's type
 -sp  <provider>      Subject's CryptoAPI provider's name
 -sy  <type>          Subject's CryptoAPI provider's type
 -iky <keytype>       Issuer key type
                         <signature|exchange|<integer>>.
 -sky <keytype>       Subject key type
                         <signature|exchange|<integer>>.
 -l   <link>          Link to the policy information (such as a URL)
 -cy  <certType>      Certificate types
                         <end|authority>
 -b   <mm/dd/yyyy>    Start of the validity period; default to now.
 -m   <number>        The number of months for the cert validity period
 -e   <mm/dd/yyyy>    End of validity period; defaults to 2039
 -h   <number>        Max height of the tree below this cert
 -len <number>        Generated Key Length (Bits)
 -r                   Create a self signed certificate
 -nscp                Include netscape client auth extension
 -eku <oid[<,oid>]>   Comma separated enhanced key usage OIDs
 -?                   Return a list of basic options
 -!                   Return a list of extended options
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

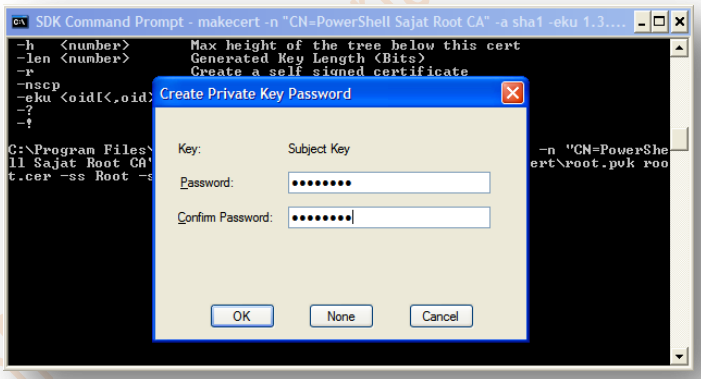
Nem egyszerű, de kezdjünk neki. Első lépésként a `makecert`-et rá kell bírni, hogy tanúsítványkiosztó hatóság szerepét vegye fel:

```
makecert -n "CN=PowerShell Sajat Root CA" -a sha1 -eku 1.3.6.1.5.5.7.3.3 -r -sv
c:\cert\root.pvk c:\cert\root.cer -ss Root -sr localMachine
```

Nézzük az egyes paraméterek jelentését:

Paraméter	Jelentés
-n "CN=PowerShell Sajat Root CA"	A tanúsítványkiosztóm neve
-a sha1	A digitális aláírás algoritmusa
-eku 1.3.6.1.5.5.7.3.3	A kiosztandó tanúsítványok fajtája, ez a szám jelenti a „Code signing” lehetőséget
-r	Önaláírt legyen a tanúsítványkiosztó tanúsítvány
-sv c:\cert\root.pvk c:\cert\root.cer	A tanúsítványkiosztó tanúsítványainak tárolási helye fájl szinten
-ss Root	A tanúsítványkiosztó tanúsítványainak tárolási helye a tanúsítványtárban
-sr localMachine	Méghozzá a helyi gépen

Amikor ezt a parancsot futtatom, akkor kétszer is megjelenik egy jelszóbekérő ablak, ami a privát kulcsok kezelésének védelmét szolgálja. Az első esetben adjuk meg a privátkulcshoz való hozzáférés jelszavát:



34. ábra Makecert.exe futtatása tanúsítványkiosztó szerep céljára

A második esetben pedig már használja is a háttérben a `makecert` ezt a privát kulcsot, ezért kéri be az előbb megadott jelszót:

A könyv tanfolyami felhasználása nem engedélyezett!

35. ábra Jelszóbeadás a privát kulcshoz való hozzáféréskor

Tehát idáig azt csináltuk, hogy létrehoztunk egy tanúsítványkiosztó önaláírt tanúsítványt, amellyel alá lesznek írva az ezután legenerálható, immár ténylegesen kódaláírásra szánt tanúsítványaink. Hozzunk is mindjárt létre egy ilyet:

```
makecert -pe -n "CN=Soos Tibi PS Guru" -ss PSCS -a sha1 -eku 1.3.6.1.5.5.7.3.3
-iv c:\cert\root.pvk -ic c:\cert\root.cer
```

Nézzük ezeknek a jelentését:

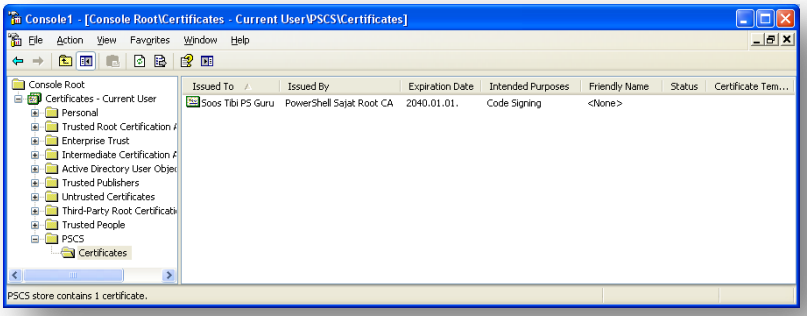
Paraméter	Jelentés
-pe	Exportálható privát kulcsot hoz létre (exportálással biztonságba tudjuk helyezni)
-n "CN=Soos Tibi PS Guru"	A kódaláíró tanúsítványon szereplő név
-ss PSCS	A tanúsítvány tárolásának a helye, én itt egy új tárolót hoztam létre, ha személyes tárolóba szeretném berakni, akkor arra a „My” névvel lehet hivatkozni
-a sha1	Digitális aláírás algoritmusa
-eku 1.3.6.1.5.5.7.3.3	Tanúsítvány célja (code signing)
-iv c:\cert\root.pvk	A kiosztó hatóság privát kulcsa
-ic c:\cert\root.cer	Az én tanúsítványomat viszonthitelesítő tanúsítvány

Miután ezen parancs futtatása során is használjuk az első lépésben létrehozott privát kulcsot, így újra meg kell adni az ott megadott jelszót:

36. ábra A kódaláíró tanúsítvány generálása során megjelenő jelszóbekérő

A könyv tanfolyami felhasználása nem engedélyezett!

Ha ezt is sikeresen lefuttattam, akkor megnézhetem, hogy a tanúsítványtáramban tényleg ott van-e a kódaláíró tanúsítvány:



37. ábra Kódaláíró tanúsítványom a tanúsítványtárban

Akkor most már van alkalmas kódaláíró tanúsítványom. Itt csatlakoznak be azok az olvasók, akiknek van „igazi” PKI infrastruktúrájuk, és egy normál tanúsítványkiosztótól szereztek kódaláíró tanúsítványt.

Most lehet aláírni a szkriptemet. Elsőként ragadjuk meg az aláírásra szánt tanúsítványunkat (itt nagyon nagy segítség a TAB-kiegészítés, azaz nem kell azt a hosszú számsort begépelni, ha a „könyvtárra” rálelünk, akkor elég a TAB-bal léptetni):

```
[9] PS C:\old> $signcert = (get-item cert:\CurrentUser\PSCS\8ED2798A04D5F794
DA6C8C3C167EB033CB8BE6C2)
[10] PS C:\old> $signcert


    Directory: Microsoft.PowerShell.Security\Certificate::CurrentUser\PSCS


Thumbprint                                Subject
----------                                -------
8ED2798A04D5F794DA6C8C3C167EB033CB8BE6C2  CN=Soos Tibi PS Guru
```

És akkor következzen a lényeg, írjuk alá a szkriptet az előbb betöltött tanúsítvánnyal:

```
[11] PS C:\old> Set-AuthenticodeSignature safe.ps1 $signcert


    Directory: C:\old


SignerCertificate                         Status          Path
-----------------                         ------          ----
8ED2798A04D5F794DA6C8C3C167EB033CB8BE6C2  Valid           safe.ps1
```

Állítsuk át a végrehajtási házirendet úgy, hogy mindenképpen megkövetelje az aláírást, és futtatom a `safe.ps1` szkriptemet:

```
[12] PS C:\old> Set-ExecutionPolicy allsigned
[13] PS C:\old> .\safe.ps1

Do you want to run software from this untrusted publisher?
```

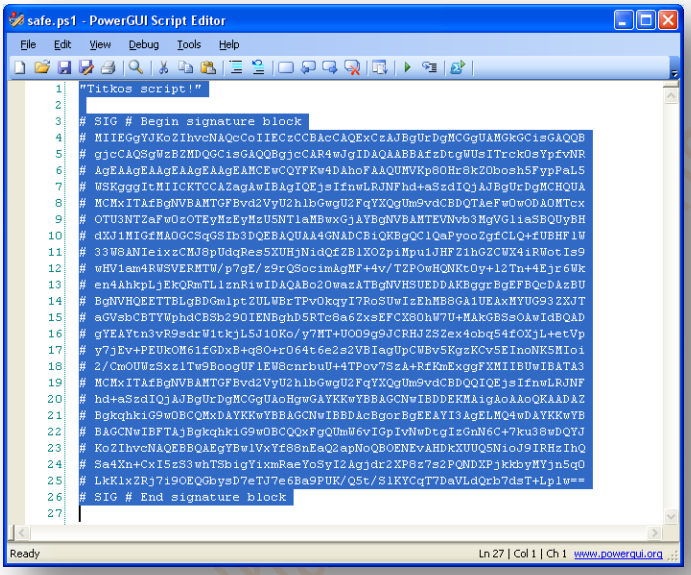
A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
File C:\old\safe.ps1 is published by CN=Soos Tibi PS Guru and is not
trusted on your system. Only run scripts from trusted publishers.
[V] Never run  [D] Do not run  [R] Run once  [A] Always run  [?] Help
(default is "D"):a
Titkos script!
```

Miután ez egy önaláírt tanúsítványon alapuló tanúsítvány, így a rendszer rákérdez, hogy elfogadom-e hitelesnek? Én azt válaszoltam, hogy igen, sőt, minden alkalommal fogadja el ezt hitelesnek (`Always`) innentől kezdve.

Nézzük meg, hogy hogyan is néz ki egy ilyen aláírt szkript:



38. ábra Az elektronikusan aláírt szkript

Az eredeti szkriptem nagyon egyszerű volt, csak annyit csinált, hogy kiírta „Titkos script!”. Ez olvasható is az első sorban. Ami utána áll az az elektronikus aláírás.

Ha bármit módosítok ebben a szkriptben, akár egy karaktert is, és futtatni akarom, akkor a következő hibajelzést fogom kapni:

```
[14] PS C:\old> .\safe.ps1
File C:\old\safe.ps1 cannot be loaded. The contents of file C:\old\safe.ps1
 may have been tampered because the hash of the file does not match the has
h stored in the digital signature. The script will not execute on the syste
m. Please see "get-help about_signing" for more details..
At line:1 char:10
+ .\safe.ps1 <<<<
```

Ha sértetlen a szkriptem, akkor ellenőrizhetem, hogy ki is írta alá, azaz értelmeztethetem azt a sok zűrös karaktert a szkriptem végén:

```
[17] PS C:\old> Get-AuthenticodeSignature C:\old\safe.ps1 | fl


SignerCertificate      : [Subject]
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                            CN=Soos Tibi PS Guru

                          [Issuer]
                            CN=PowerShell Sajat Root CA

                          [Serial Number]
                            123B087E7C0B44934585DF9A4B374842

                          [Not Before]
                            2008.04.17. 21:57:56

                          [Not After]
                            2040.01.01. 0:59:59

                          [Thumbprint]
                            8ED2798A04D5F794DA6C8C3C167EB033CB8BE6C2

TimeStamperCertificate :
Status                 : Valid
StatusMessage          : Signature verified.
Path                   : C:\old\safe.ps1
```

1.8.7 Végrehajtási preferencia

Beszéltünk az álnevekről, cmdletekről, függvényekről, szkriptekről. Láttuk azt is, hogy a PowerShell ugyanúgy végrehajtja az elérési úton található futtatható állományokat, mint ahogy a hagyományos parancssor.

Azt is láttuk, hogy elnevezések tekintetében elég szabad a PowerShell, nagyon kevés megszorítás van a neveket illetően, így például lehet egyforma neve egy általunk létrehozott függvénynek és aliasnak, mint egy meglevő cmdletnek. Például elnevezhetek egy függvény így is:

```
[14] PS C:\old> function script.ps1 {"Ez most függvény"}
[15] PS C:\old> script.ps1
Ez most függvény
```

Annak ellenére is, hogy ugyanilyen névvel van nekem egy szkriptem is, ráadásul éppen az aktuális könyvtárban:

```
[16] PS C:\old> Get-Content script.ps1
"Ez most a szkript"
```

Ennek ellenére a [15]-ös sorban a „script.ps1” beírására nem a szkriptem, hanem a függvényem futott le. És ez még nem elég, ugyanilyen névvel létrehozhatok egy álnevet is:

```
[17] PS C:\old> New-Alias script.ps1 get-command
```

Ha most futtatom a „script.ps1”-et, akkor az alias fut le:

```
[18] PS C:\old> script.ps1

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Add-Content                         Add-Content [-Path] <Strin...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Cmdlet          Add-History                         Add-History [[-InputObject...
Cmdlet          Add-Member                          Add-Member [-MemberType] <...
Cmdlet          Add-PSSnapin                        Add-PSSnapin [-Name] <Stri...
Cmdlet          Clear-Content                       Clear-Content [-Path] <Str...
...
```

Hogyan lehet mégis a szkriptet lefuttatni? Hát például így:

```
[19] PS C:\old> &(get-item script.ps1)
Ez most a szkript
```

Hogyan tudom a függvényt futtatni? Hasonló logikával, mint a szkriptet, csak más a tárolásának helye:

```
[20] PS C:\old> &(get-item function:script.ps1)
Ez most függvény
```

Nézzük összefoglalva a végrehajtási preferenciát:

1. PowerShell kulcsszó
2. Alias
3. Függvény
4. Cmdlet
5. Futtatható fájlok (exe, com, stb.)
6. Szkriptek
7. Kiterjesztés alapján a hozzá tartozó alkalmazás futtatása

Azaz nagyon vigyázni kell, hogy milyen néven hozunk létre álneveket és függvényeket, mert ezek a végrehajtási preferenciában megelőzik a cmdleteket is. Például egy gonosz szkript átdefiniálhatja a gyári cmdleteket ugyanolyan nevű álnevekkel vagy függvényekkel és a PowerShell teljesen másként kezd el viselkedni, mint ahogy várjuk. Egyedül a PowerShell kulcsszavai élveznek viszonylagos védettséget, hiszen ha készítünk egy „FOR” nevű függvényt, akkor a FOR beírására az FOR ciklusként értelmezi a PowerShell, és az ugyanilyen nevű függvényem eléréséhez a fenti, körülményesebb módszert kell alkalmazni.

A könyv tanfolyami felhasználása nem engedélyezett!

1.9 Fontosabb cmdletek

A PowerShell főbb nyelvi elemeit áttekintettük az előző fejezetekben, most szemelgessünk a legfontosabb cmdletek között. Ezek a cmdletek annyira általánosan használhatók, hogy szinte nyelvi elemként lehet őket tekinteni, ezért mindenképpen itt, az elméleti részben kell őket tárgyalni.

1.9.1 Véletlen szám generálás és annál sokkal több (Get-Random)

A PowerShell 1.0 verziójában nem volt véletlen szám generátor, de nagyon egyszerűen segítségül lehetett hívni a `system.random` .NET osztályt és azzal tudtunk véletlen számot generálni. Gondoltam, hogy azért illene ilyen cmdletet belerakni a következő verzióba, de hogy ilyen sokat ki lehet hozni egy véletlen szám generátor cmdletből, arra nem számítottam. Nézzük sorjában, mit tud ez a `get-random` cmdlet!

```
[1] PS C:\> get-random
887627512
[2] PS C:\> Get-Random 100
55
[3] PS C:\> Get-Random -Minimum 50 -Maximum 100
70
[4] PS C:\> Get-Random 0.999999999999999
0,568479508426263
```

Az [1]-es sorban láthatjuk a legegyszerűbb alapesetet, a `get-random` paraméter nélkül. Ilyenkor egy 0 és `[int32]::maxvalue`, azaz 2147483647 közötti egész számot kapunk eredményül. A [2]-es sorban 0 és 100 közötti véletlen számot kapunk, látható, hogy a maximum érték a pozícionális paraméter. A [3]-as sorban 50 és 100 közötti véletlen számot kapunk, ha valamelyik határérték nem egész, akkor lebegőpontos számot kapunk, mint ahogy a [4]-es sorban látható.

És most jön az igazán érdekes!

```
[29] PS C:\> "Aladár", "Bence", "Csaba", "Dénes", "Ferenc", "Hugó", "Istvá
n" | Get-Random -Count 3
István
Csaba
Bence
```

Csövezhető tehát a `get-random`, ilyenkor a `-count` paramétert kell használni, ami megadja, hogy a gyűjteményből hány elemet adjon vissza! Ez un. „nem visszatevéses” kiválasztás, azaz egy elem csak egyszer kerülhet bele a kiválasztásba. Ha a `-count` értéke megegyezik vagy több, mint a csőelemek száma, akkor az összes elemet visszakapjuk, csak véletlen sorrendben:

```
[30] PS C:\> "Aladár", "Bence", "Csaba", "Dénes", "Ferenc", "Hugó", "Istvá
n" | Get-Random -Count 7
Hugó
Bence
Ferenc
Dénes
Aladár
István
Csaba
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Mire lehet használni egy ilyen cmdletet? Természetesen a véletlen számok generálása elsősorban a játékoknál és különböző algoritmusok tesztelésénél jönnek jól, ezekre a PowerShellben talán kevesebbet van szükség, viszont ez a csövezhetőség nagyon jól jön szkriptjeink tesztelésénél! Gondoljunk valamilyen fájlfeldolgozó szkriptre. Valószínű a tesztelés fázisában nem akarjuk nagyon sok fájlra ráereszteni, viszont az sem lenne jó, ha mindig ugyanazzal a 2 fájllal tesztelnénk. Ilyenkor a fájlok egy nagy halmazából véletlenül választunk ki néhányat és azzal tesztelünk.

Vagy pl. sokkal olcsóbb lenne a pörgő golyók helyett ezzel lottószámokat generálni hétről hétre:

```
[32] PS C:\> 1..90 | Get-Random -Count 5 | Sort-Object
12
15
23
65
80
```

Sok szerencsét! ☺

1.9.2 Csővezeték feldolgozása (Foreach-Object) – újra

Ugyan már volt szó a `ForEach-Object` cmdletről a *1.6.5 ForEach-Object cmdlet* fejezetben, de most már ismerjük a függvényeket és filtereket, így tisztában vagyunk azzal, hogy hogyan történik a csővezetéken érkező objektumok feldolgozása paraméterként átadott szkriptblokkok segítségével.

Tulajdonképpen már mi magunk is meg tudnánk írni egy `ForEach-Object` cmdlethez hasonló működésű függvényt, de mivel ezt olyan gyakran használnánk, hogy érdemesebb volt ez gyorsabban lefutó, natív PowerShell parancsként implementálni.

Ennek ellenére elég tanulságos lenne egy ilyen függvényt írni. Ehhez először nézzük meg a `ForEach-Object` szintaxisát:

```
[1] PS C:\> (get-help foreach-object).syntax

ForEach-Object [-process] <ScriptBlock[]> [-inputObject <psobject>] [-begin
 <scriptblock>] [-end <scriptblock>] [<CommonParameters>]
```

Most a `CommonParameters` részt kihagyom, de a többit én is definiálom a függvényemben:

```
function saját-foreachobject ([scriptblock] $process = $(throw "Kötelező!"),
     [PSObject] $inputObject, [scriptblock] $begin, [scriptblock] $end)
{
     begin
     {
       if($inputObject)
       {
          $inputObject | saját-foreachobject -p $process -b $begin -e $end
          return
       }
       if($begin) {&$begin}
     }
     process {&$process}
     end
     {
         if($end) {&$end}
     }
```

```
}
```

Itt a `$process` paraméter kötelezően kitöltendő, hibát ad a függvényem, ha ez hiányzik:

```
[3] PS C:\> 222 | saját-foreachobject
Kötelező!
At line:1 char:63
+ function saját-foreachobject ([scriptblock] $process = $(throw <<<<  "Kötele
ző!"),
    + CategoryInfo          : OperationStopped: (Kötelező!:String) [], Runtim
   eException
    + FullyQualifiedErrorId : Kötelező!
```

Ha ez helyesen van kitöltve, akkor akár csővezetékkel is jól működik:

```
[4] PS C:\> 1,3,44, "q" | saját-foreachobject {$_*3}
3
9
132
qqq
```

De működik úgy is, ha nem „belecsövezzük” a feldolgozandó objektumot, hanem `-inputobject` paraméterként átadjuk:

```
[5] PS C:\> saját-foreachobject {$_*2} -i 654
1308
```

Ehhez a függvénydefinícióban egy kicsit trükköztem, ha ez utóbbi módszerrel akarnék objektumot átadni, akkor nem lenne `$_` változó, ezért a `begin` szekcióban meghívom rekurzív módon a függvényből saját magát, immár csövezős módszerrel. Azért, hogy a rekurzív hívásból visszatérve ne fusson le még egyszer a függvényem, ezért ott egy `return`-nel kilépek.

Ilyen jellegű csövezhető paraméterátadást a PowerShell 2.0 fejlett függvényeivel sokkal egyszerűbben és profibban meg tudunk valósítani, ott nem kell rekurzív hívást külön leprogramozni. (Ld. *2.4 Fejlett függvények – script cmdletek* fejezet!)

Megjegyzés

Vizsgáljuk meg egy kicsit jobban ezt a futószalag-rendszert! Nézzük a következő példát, ahol a bemeneti elemeket ráteszem a futószalagra, majd 3 `foreach-object` „gépen” is keresztül megy a nyersanyag:

```
PS C:\> 'a', 'b', 'c' |
>>     ForEach-Object -Begin {Write-Host "Az 1. gép beindul"} -Process {"$_ 1"}
 -End {write-host "Az 1. gép leáll"} |
>>     ForEach-Object -Begin {Write-Host "Az 2. gép beindul"} -Process {"$_ 2"}
 -End {write-host "Az 2. gép leáll"} |
>>     ForEach-Object -Begin {Write-Host "Az 3. gép beindul"} -Process {"$_ 3"}
 -End {write-host "Az 3. gép leáll"}
>>
Az 1. gép beindul
Az 2. gép beindul
Az 3. gép beindul
a 1 2 3
b 1 2 3
```

www.tanfolyam-elhasználása nem engedélyezett!

```
c 1 2 3
Az 1. gép leáll
Az 2. gép leáll
Az 3. gép leáll
```

A kimenetből látható, hogy az egymás utáni szakaszok gépeinek `begin` szekciói szépen egymás után lefutnak, mint ahogy az elinduló futószalag is minden gép előtt elindul. Ezután az egyes elemek ahogy végig haladnak a gépek előtt, úgy dolgoznak rajtuk a gépek, először az első, majd második, végül a harmadik. A futószalag leállása megint mindhárom gépet egyszerre érinti.

1.9.3 A csővezeték elágaztatása (Tee-Object)

A `Tee-Object` cmdlet a csővezeték megcsapolására szolgál. Tetszés szerinti pontra beillesztve az arra járó objektumkupacot egy változóba vagy egy megadott fájlba írja, de eközben változatlan formában továbbküldi a csövön is, a következő parancs pontosan úgy kapja meg az objektumokat, mintha a `Tee-Object` ott sem lett volna. Az alábbi parancs például a c: meghajtó mappalistáját kérdezi le, a `Tee-Object` ennek szöveges megfelelőjét beleírja a c:\dir.txt fájlba, a lekérdezett eredeti objektumokat pedig továbbadja a `Where-Object` cmdletnek:

```
PS C:\> Get-ChildItem | Tee-Object -FilePath $home\dir.txt | Where-Object {$_.N
ame -eq "Windows"}


    Directory: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       2010.02.14.     10:34            Windows
```

Leggyakrabban a fentihez hasonló esetekben használjuk ezt a cmdletet, azaz amikor a kimenetet kétfelé szeretnénk ágaztatni: egyrészt például naplózási céllal beirányítjuk a csővezeték tartalmát egy fájlba, másrészt valami egyéb tevékenységet is végzünk ezekkel az objektumokkal.

1.9.4 Csoportosítás (Group-Object)

A `Group-Object` cmdlet segítségével egy (vagy több) megadott tulajdonság értéke szerint csoportosíthatunk objektumokat. A csoportokba az azonos tulajdonságértékkel rendelkező objektumok kerülnek.

Készítsünk listát, amely az „g” betűvel kezdődő cmdleteket csoportosítja ige szerint! A cmdleteket lekérdező `Get-Command` kimenetét a `Group-Object`-nek kell megkapnia, paraméterként pedig meg kell adnunk, hogy melyik tulajdonság értékei szerint akarjuk elvégezni a csoportosítást, ez a `Verb` tulajdonság lesz:

```
[56] PS C:\> get-command g* -commandtype cmdlet | group-object -property verb -
ashashtable -asstring

Name                           Value
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
----                                -----
Get                                 {Get-Acl, Get-Alias, Get-AuthenticodeSignatu...
Group                               {Group-Object}
```

A csoportok és a bennük található elemek száma jól látható, de maga a csoport tagsága ebben a nézetben nem igazán kivehető, ha túl sok az adott csoportban található elem.

Megjegyzés

A csoportokat alaphelyzetben csak maximum 4 elemig jeleníti meg a PowerShell, feltéve, hogy a konzol szélessége ezt megengedi. Ha ennél több elemet is meg akarunk jeleníttetni, akkor a `$FormatEnumerationLimit` automatikus változó értékét kell átállítani.
Hogy a képernyőszélesség ne legyen korlát, az egészet még táblázatosan formázom tördeléses módon:

```
[11] PS C:\> Get-Command g* -CommandType cmdlet | Group-Object -Property verb |
 Format-Table -Wrap

Count Name                      Group
----- ----                      -----
   46 Get                       {Get-Acl, Get-Alias, Get-AuthenticodeSignature
                                , Get-ChildItem...}
    1 Group                     {Group-Object}
```

Nézzük mi lesz, ha a `$FormatEnumerationLimit` változót megnövelem:

```
[12] PS C:\> $FormatEnumerationLimit = 8
[13] PS C:\> Get-Command g* -CommandType cmdlet | Group-Object -Property verb |
 Format-Table -Wrap

Count Name                      Group
----- ----                      -----
   46 Get                       {Get-Acl, Get-Alias, Get-AuthenticodeSignature
                                , Get-ChildItem, Get-Command, Get-ComputerRest
                                orePoint, Get-Content, Get-Counter...}
    1 Group                     {Group-Object}
```

A kimeneten látszik, hogy most az első csoportban több elem vált láthatóvá.

Készítsünk listát, amely a PowerShell súgóinak különféle kategóriáinak darabszámát tartalmazza! Ebben a feladatban tulajdonképpen semmi újdonság nincsen a megelőzőhöz képest, csak azért került ide, mert igen szép példát ad a PowerShell objektumkezelésének szinte tökéletesen egységes voltára. Mindegy, hogy cmdletekről vagy a PowerShell súgótémáinak listájáról van szó, a szükséges parancsok gyakorlatilag azonosak, a tökéletesen különböző típusú adatoktól függetlenül. A megoldás:

```
[14] PS C:\> get-help * | group-object -property category

Count Name                      Group
----- ----                      -----
  137 Alias                     {@{Name=ac; Category=Alias; Synopsis=Add-Co...
  236 Cmdlet                    {@{relatedLinks=@{navigationLink=System.Man...
    8 Provider                  {@{CmdletHelpPaths=@{CmdletHelpPath=System....
   90 HelpFile                  {TOPIC...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ha valakinek ez sem elég, akkor még olyat is lehet csinálni a `Group-Object` segítségével, hogy egyszerre több szempont szerinti csoportot is létrehozhatunk:

```
[60] PS C:\> get-process | Group-Object -Property processname, company

Count Name                      Group
----- ----                      -----
    3 conhost, Microsoft Cor... {System.Diagnostics.Process (conhost), Syst...
    2 csrss, Microsoft Corpo... {System.Diagnostics.Process (csrss), System...
    1 dfsrs, Microsoft Corpo... {System.Diagnostics.Process (dfsrs)}
    1 dfssvc, Microsoft Corp... {System.Diagnostics.Process (dfssvc)}
    1 dns, Microsoft Corpora... {System.Diagnostics.Process (dns)}
    1 dwm, Microsoft Corpora... {System.Diagnostics.Process (dwm)}
…
```

A fenti paranccsal a processzek neve és gyártójuk alapján együttesen csoportosítottam a futó folyamatokat. Ez nem igazi kétszintű csoportosítás, hanem képez egy „látszólagos” tulajdonságot, amely a processz nevéből és gyártójából áll, és ezen összetett tulajdonság alapján csoportosít.

Ha nincs szükségünk az egyes elemekre, csak a csoportokra, akkor használhatjuk a `-NoElement` kapcsolót:

```
[21] PS C:\> get-process | Group-Object -Property company -noelement

Count Name
----- ----
   38 Microsoft Corporation
    2
    2 Sun Microsystems, Inc.
```

A fenti példában csak a gyártókra voltam kíváncsi és a darabszámokra.

Alaphelyzetben a `Group-Object` kimenete egy gyűjtemény, nézzük meg, hogy egy-egy eleme ennek hogyan is néz ki:

```
[70] PS C:\> $csoportok = get-command g* -commandtype cmdlet | group-object -pr
operty verb
[71] PS C:\> $csoportok[0]

Count Name                      Group
----- ----                      -----
   46 Get                       {Get-Acl, Get-Alias, Get-AuthenticodeSignat...


[72] PS C:\> $csoportok[1]

Count Name                      Group
----- ----                      -----
    1 Group                     {Group-Object}
```

Ebből kinyerhetők az egyes csoportosított elemek a következő módon:

```
[81] PS C:\> $csoportok[0].group

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Get-Acl                             Get-Acl [[-Path] <String[]>...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Cmdlet          Get-Alias                           Get-Alias [[-Name] <String[...
Cmdlet          Get-AuthenticodeSignature           Get-AuthenticodeSignature [...
Cmdlet          Get-ChildItem                       Get-ChildItem [[-Path] <Str...
Cmdlet          Get-Command                         Get-Command [[-ArgumentList...
…
```

Ennél egyszerűbb módja is van ennek, ha arra kérjük a `Group-Object`-et, hogy ne gyűjteményt készítsen nekünk, hanem hashtáblát:

```
[82] PS C:\> $csoportok = get-command g* -commandtype cmdlet | group-object -pr
operty verb -AsHashTable
[83] PS C:\> $csoportok

Name                           Value
----                           -----
Get                            {Get-Acl, Get-Alias, Get-AuthenticodeSignatu...
Group                          {Group-Object}


[84] PS C:\> $csoportok.get

CommandType     Name                           Definition
-----------     ----                           ----------
Cmdlet          Get-Acl                        Get-Acl [[-Path] <String[]>...
Cmdlet          Get-Alias                      Get-Alias [[-Name] <String[...
Cmdlet          Get-AuthenticodeSignature      Get-AuthenticodeSignature [...
Cmdlet          Get-ChildItem                  Get-ChildItem [[-Path] <Str...
Cmdlet          Get-Command                    Get-Command [[-ArgumentList...
…
```

Látható, hogy a `–AsHashTable` kapcsolóval ezt elérhetjük, így sokkal egyszerűbb az egyes csoportokba sorolt elemeket elérni.

Ezekből a példákból még az is kiderülhetett számunkra, hogy a `Group-Object` használata előtt nem kell külön sorba rendezni a csővezetéken érkező objektumokat, hogy a csoportok jól kialakuljanak. Emlékezhetünk a `format-table` cmdlet `-groupby` paraméterére az *1.3.17 Egyszerű formázás* fejezetből, ott azt láthattuk, hogy a `format-table` számára előbb sorba kellett rendezni az objektumokat, hogy jó legyen a csoportosítás. Itt ilyesmire nincs szükség, hiszen a `Group-Object` halmazokat készít, nem csak megjeleníti a csoportokat.

1.9.5 Objektumok átalakítása (Select-Object)

A `Select-Object` cmdlet a kapott objektumok megcsonkítását képes elvégezni, a kimenetként kapott objektumokban már csak a paraméterlistában megadott tulajdonságok fognak szerepelni, a többi tulajdonságot (és a bemenetként kapott objektumreferenciát is) a cmdlet egyszerűen eldobja. Viszont így az eredeti objektum tulajdonságait átalakíthatjuk, újabb tulajdonságokat számolhatunk ki, hozhatunk létre, amelyekkel esetleg könnyebben tudunk dolgozni.

Példaként alakítsuk át a `Get-Process`-től érkező objektumokat úgy, hogy csak a folyamat nevét, gyártóját és leírását tartalmazzák! A `Select-Object` cmdletnek egyszerűen azt kell megadnunk, hogy melyik tulajdonságokat szeretnénk megtartani:

```
PS C:\> get-process | select-object name, company, description
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name                      Company                   Description
----                      -------                   -----------
ctfmon                    Microsoft Corporation     CTF Loader
CTHELPER                  Creative Technology Ltd   CtHelper MFC Application
CTSVCCDA                  Creative Technology Ltd   Creative Service for ...
csrss
explorer                  Microsoft Corporation     Windows Intéző
...
```

Mi is történik ebben az esetben? A `Get-Process` Process objektumokat dobál a csőbe, ezeket kapja meg a `Select-Object`, amely a paraméterlista által meghatározott tulajdonságokat egy újonnan létrehozott egyedi objektum azonos nevű tulajdonságaiba másolja. Ezek az objektumok fognak aztán továbbvándorolni a csövön.

Természetesen nemcsak azt mondhatjuk meg, hogy mely tulajdonságokra van szükségünk, hanem azt is, hogy melyekre nincs: az `-excludeproperty` paraméter után felsorolt tulajdonságok nem fognak szerepelni a kimenő objektumokban (de az összes többi igen).

A kimenő objektumok szerkezetét egy `Get-Member` hívás mutatja meg:

```
PS C:\> get-process | select-object -property name, description, company |
get-member


   TypeName: System.Management.Automation.PSCustomObject

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       System.Boolean Equals(Object obj)
GetHashCode Method       System.Int32 GetHashCode()
GetType     Method       System.Type GetType()
ToString    Method       System.String ToString()
Company     NoteProperty System.Management.Automation.PSObject Company=...
Description NoteProperty System.Management.Automation.PSObject Descript...
Name        NoteProperty System.String Name=alg
```

Látszik, hogy ez már nem az eredeti objektum, hanem egy `PSCustomObject`.

A fentieken kívül a `Select-Object` néhány más funkcióval is rendelkezik: nemcsak elemi objektumokat képes átalakítani, hanem gyűjteményeket is átalakít. Például képes a bemenetként kapott tömb elejéről vagy végéről meghatározott számú elemet leválasztani (`-first` és `-last` paraméter), illetve képes egy gyűjtemény elemei közül csak a különbözőeket (vagyis az egyforma elemek közül csak egyet) továbbadni a `-unique` kapcsoló segítségével:

```
PS C:\> PS C:\> 1,1,3,56,3,1,1,1,3 | Select-Object -unique
1
3
56
```

Ilyenkor a gyűjtemény, tömb egyes elemei természetesen megőrzik típusukat.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

De vajon mi számít a `-unique` kapcsoló használata mellett egyformának és mi különbözőnek? Nézzünk erre egy példát:

```
PS C:\> Get-Process s*

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    681      91   107680     134448   735    14,52   3640 ScriptEditor
    145      14     4708       9080    67            1240 SDWinSec
   1094      70    45188      33320   162            3008 SearchIndexer
    225      13     5888       9120    42             524 services
    348      26    40220      36316   191    18,19   2776 sidebar
     87      10     3168       6484    72     0,14   2416 SJelite3Launch
     30       2      456       1052     5             276 smss
    322      20     7120      12052    94            1360 spoolsv
    371      14     4440       9400    54             660 svchost
    369      16     4744       8736    42             756 svchost
    513      25    17420      19168    90             900 svchost
    671      33   108432     115488   240             932 svchost
   1564      80    32140      48064   481             960 svchost
    286      20     6136      10852    49            1108 svchost
    761      51    32136      38768   174            1220 svchost
    306      33    12540      14112    69            1556 svchost
    319      26     6580      13072    90            1704 svchost
     95       8     1864       5164    33            1996 svchost
     91       8     1640       4408    47            2496 svchost
    769       0      124       6300    11               4 System
```

Nézzük ezeket „megegyelve”:

```
PS C:\> Get-Process s* | Select-Object -Unique

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    681      91   107680     134448   735    14,52   3640 ScriptEditor
    145      14     4708       9080    67            1240 SDWinSec
   1094      70    45188      33320   162            3008 SearchIndexer
    225      13     5888       9120    42             524 services
    348      26    40220      36316   191    18,16   2776 sidebar
     87      10     3168       6484    72     0,14   2416 SJelite3Launch
     30       2      456       1052     5             276 smss
    322      20     7120      12052    94            1360 spoolsv
    369      14     4388       9380    53             660 svchost
    769       0      124       6300    11               4 System
```

Láthatólag a sok svchost lett összevonva, de vajon milyen alapon? Pusztán a neve alapján? És az olyan objektumok esetében, amelyeknek nincs nevük, ott mi lesz? Az algoritmus nagyon egyszerű: az a két objektum egyforma, amelyiknek meghívva a `ToString` metódusát egyforma lesz a kimenet. Ez két különböző svchost-példány esetében is ugyanaz, ezért lettek összevonva:

```
PS C:\> Get-Process svchost | ForEach-Object {$ .tostring()}
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
```

```
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
System.Diagnostics.Process (svchost)
```

Sajnos itt nem tudjuk testre szabni, hogy mit tekintsen egyformának és mit nem, így kerülő megoldást kell készítenünk erre a problémára a később tárgyalásra kerülő `Compare-Object` cmdlet segítségével.

Nézzünk végül arra is példát, hogy a `Select-Object`-tel újabb, számolt tulajdonságokat is létrehozhatunk. Itt most a jobb értelmezhetőség kedvéért áttördeltem a kódot:

```
dir c:\old | Select-Object name,
@{   n = "Méret";
     e =   {switch ($m = $_.Length)
                {      {$m -lt 1kb} {"Pici"; break}
                       {$m -lt 3kb} {"Közepes"; break}
                       {$m -lt 100kb} {"Nagy"; break}
                       default {"Óriási"}
                }
            }
 }
```

A fenti példában nem vagyok megelégedve a `dir`, azaz a `get-childitem` által a fájloknál megmutatott tulajdonságokkal, nekem kellene egy pici/közepes/nagy/óriási besorolása a fájloknak. Ehhez a `Select-Object`-nek a fájlok „igazi” `name` tulajdonsága mellett egy „képzett” tulajdonságot is megadok. Ennek formája:

```
@{ name = "Tulajdonságnév"; expression =  Tulajdonságérték}
```

A „name” és „expression” paramétereket lehet rövidíteni. Nálam az „expression” rész egy `switch` kifejezés, mellyel besorolom a fájlokat a méretük alapján.

És nézzük a kimenetet:

```
Name                                Méret
----                                -----
alice.txt                           Pici
coffee.txt                          Pici
lettercase.txt                      Pici
numbers.txt                         Pici
presidents.txt                      Pici
readme.txt                          Közepes
skaters.txt                         Nagy
symbols.txt                         Pici
vertical.txt                        Pici
votes.txt                           Nagy
wordlist.txt                        Óriási
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Még egy gyakori felhasználási területe van a `select-object`-nek. Ennek felvezetésére nézzünk egy egyszerű példát:

```
[1] PS C:\>get-service a* | Group-Object -Property status

Count Name                      Group
----- ----                      -----
    3 Stopped                   {Alerter, AppMgmt, aspnet_state}
    2 Running                   {ALG, AudioSrv}
```

Az [1]-es sorban lekérdezem az „a” betűvel kezdődő szolgáltatásokat, majd csoportosítottam őket a status paraméterük alapján. A kimeneten az egyes csoportokat alkotó szolgáltatások tömbbe rendezve (kapcsos zárójelek között) láthatók.

Ez most a kevés szolgáltatásnál akár jó is lehet, de ha az összes szolgáltatásra futtattam volna, akkor nem nagyon fért volna ki az egyes csoportok listája. A kibontásban segíthet a `select-object` az `expandproperty` paraméterrel:

```
[2] PS C:\>get-service a* | Group-Object -Property status | Select-Object -E
xpandProperty group

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Alerter
Stopped  AppMgmt            Application Management
Stopped  aspnet_state       ASP.NET State Service
Running  ALG                Application Layer Gateway Service
Running  AudioSrv           Windows Audio
```

A [2]-es sor utolsó tagjában a `select-object` kifejti a `group` tulajdonságot, eredményeképpen visszakapjuk a szolgáltatások listáját, immár csoportosítás utáni sorrendben. Gyakorlatilag ugyanahhoz az eredményhez jutottunk, mint az *1.3.17 Egyszerű formázás* fejezet végén a `get | sort | format-table -groupby` kifejezéssorral. Mivel itt is gyűjteményen végeztem a `select-object`-tel a műveletet, az egyes elemek eredeti objektumtípusa megőrződött:

```
[3] PS C:\> get-service a* | Group-Object -Property status | Select-Object -
ExpandProperty group | Get-Member


   TypeName: System.ServiceProcess.ServiceController

Name                        MemberType     Definition
----                        ----------     ----------
Name                        AliasProperty  Name = ServiceName
add_Disposed                Method         System.Void add_Disposed(EventHa...
Close                       Method         System.Void Close()
Continue                    Method         System.Void Continue()
...
```

Láthatjuk, hogy a művelet végén `ServiceController` típusú objektumokat kaptunk.

Még tud pár dolgot a `Select-Object`, például az `-Index` paraméterrel valahányadik elemet választja ki a gyűjteményből:

```
PS C:\> 1..10 | Select-Object -Index 5,8
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
6
9
PS C:\> 1..10 | Select-Object -Index 8
9
```

Vagy valahány darab elemet elhagy az elejéből:

```
PS C:\> 1..10 | Select-Object -Skip 7
8
9
10
```

Vagy a végéből:

```
PS C:\> 1..10 | Select-Object -Skip 7 -Last 10
1
2
3
```

1.9.6 Rendezés (Sort-Object)

Listázzuk ki a tíz legtöbb memóriát foglaló folyamatot a memóriafoglalás sorrendjében!

```
PS C:\> get-process | sort-object -property workingset -desc
```

A folyamatok listáját természetesen ismét a `Get-Process` adja. A folyamat által igénybe vett fizikai memória mennyiségét a „`Workingset`” tulajdonság adja meg, ami az alapértelmezett táblázatban is szerepel. A `Sort-Object` tetszőleges tulajdonság értékei szerint tudja sorba rendezni a kimenetet. A listázás alapértelmezés szerint a legkisebb értéktől indul, de ha megadjuk a `-desc` paramétert, akkor a legnagyobb értékek kerülnek a lista tetejére.

A lista tehát már rendezett, de nekünk csak az elején szereplő tíz sorra lenne szükségünk. A korábban már használt `Select-Object` cmdlet egyik alfunkciójának segítségével ez is könnyen megoldható, egyszerűen paraméterként kell megadnunk, hogy hány sorra van szükségünk. A megoldás tehát:

```
PS C:\> get-process | sort-object -property workingset -desc | select-object -
first 10

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   2779      45   197548      45732   439   407,91    460 iexplore
    420       9    41488      34552   159     6,13    612 powershell
    463      18    34580      23060   343   114,03   3952 WINWORD
    693      20    22396      20312   123    39,34    960 explorer
    317      19    45132      20048   125   121,17    624 Mcshield
   1729      82    25332      16368   215    96,22   1436 svchost
    557      16    46872       7720   180    42,11   2896 OUTLOOK
    779      20    67944       7200   198    19,30   2496 iexplore
    340      11    17876       6716   122   143,34   1540 svchostl
    456      13    48196       6452   139     4,84   3604 iexplore
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nem mindegy, hogy milyen kultúrkör, nyelv ábécéje szerint rendezzük az objektumokat. Alaphelyzetben az adott gép területi beállításait veszi figyelembe a PowerShell, de ezt a `-culture` paraméterrel megváltoztathatjuk:

```
[24] PS C:\> "zug", "zsuga", "csak", "cuki" | Sort-Object -Culture "hu-hu"
cuki
csak
zug
zsuga
[25] PS C:\> "zug", "zsuga", "csak", "cuki" | Sort-Object -Culture "en-us"
csak
cuki
zsuga
zug
```

A [24]-es sorban magyar nyelv szabályai szerint rendeztem sorba a szavakat, míg a [25]-ös sorban ugyanezeket a szavakat az angol ábécé szerint. Látjuk, hogy jelentős eltérés van a két sorrend között, hiszen a magyar nyelvben a kettős betűk külön sorolódnak be a az ábécébe.

A különböző nyelvekhez tartozó nyelvi kódok jelölését a http://msdn2.microsoft.com/en-us/library/ms970637.aspx oldalon meg lehet nézni.

1.9.7 Még egyszer formázás (Format-Table, Format-Wide)

Az előzőekben láthattuk, hogy a `select-object` segítségével kiszámoltathatunk új tulajdonságértékeket is. Azonban ilyesmire a `format-table` is alkalmas:

```
[9] PS C:\old>Get-ChildItem | Format-Table name,@{Expression={if($_.psiscont
ainer){"Könyvtár"}else{"Fájl"}};Label="Típus";width=10}

Name                                                                 Típus
----                                                                 -----
alfolder                                                             Könyvtár
alice.txt                                                            Fájl
coffee.txt                                                           Fájl
dir.xml                                                              Fájl
```

Azaz itt is megadható egy hashtábla kifejezés, amit a `format-table` kiszámol, illetve beépít a táblázatba. A hashtábla lehetséges címkéi:

Mező	Jelentése
Expression	A kiszámolandó kifejezés
Label	A táblázat oszlopának címkéje
Width	Az oszlop szélessége
FormatString	Formázó operátornál használatos formázó kifejezés
Alignment	Rendezés, lehetséges értékek: „Left”, „Center” és „Right”

Ez annyiban különbözik a `select-object`-es átalakítástól, hogy itt az eredeti objektumhoz nem nyúlunk, az minden eredeti tulajdonságát és a típusát megőrzi, csak a megjelenítést változtatjuk meg.

Nézzünk még egy példát:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> Get-Process | Format-Table name, id, @{
>> label = "Mióta";
>> expression = {((get-date) - $_.starttime).totalseconds};
>> formatstring = "n0";
>> alignment = "right"}
>>

Name                                                  Id                    Mióta
----                                                  --                    -----
audiodg                                              688
btdna                                               1696                36 611
conhost                                             1028                   456
conhost                                             4876                 2 451
csrss                                                416
csrss                                                504
daemon                                              2524                36 613
…
```

Ha nagyon sok elemet akarunk megjeleníteni és nem annyira érdekes több tulajdonság megjelenítése, akkor használhatjuk a `Format-Wide` megjelenítést:

```
[3] PS C:\> dir | Format-Wide


    Directory: C:\



[Lurdy]                                 [munka]
[PerfLogs]                              [Program Files]
[Program Files (x86)]                   [reflector]
[sokfájl]                               [Users]
[Windows]                               fájl.txt
kon.psc1                                obj.xml
service.html
```

Itt is van jó néhány lehetőségünk a további testre szabásra. Például optimálisabban használhatjuk ki a rendelkezésünkre álló képernyőt az `-AutoSize` kapcsolóval:

```
[6] PS C:\> dir | Format-Wide -Property name -AutoSize


Lurdy                   munka                   PerfLogs
Program Files           Program Files (x86)     reflector
sokfájl                 Users                   Windows
fájl.txt                kon.psc1                obj.xml
service.html
```

Látható, hogy viszont itt elvesztettünk néhány „okos” formázást, mint például a „fejlécet” és a könyvtárak nevének szögletes zárójelbe tételét, mert ezeket nem alaphelyzetben a `Format-Wide` tudja, hanem a fájl és mappatípushoz definiált formátumleíró XML állomány, ezekről majd később az *1.4.16 Formázás testre szabása (Export-FormatData, Get-FormatData, Update-FormatData)* fejezetben olvashatnak. Cserébe kérhetünk csoportosítást és megadhatjuk, hogy hány oszlopot kérünk:

```
[9] PS C:\> dir | Format-Wide -Property name -Column 2 -GroupBy psiscontainer
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

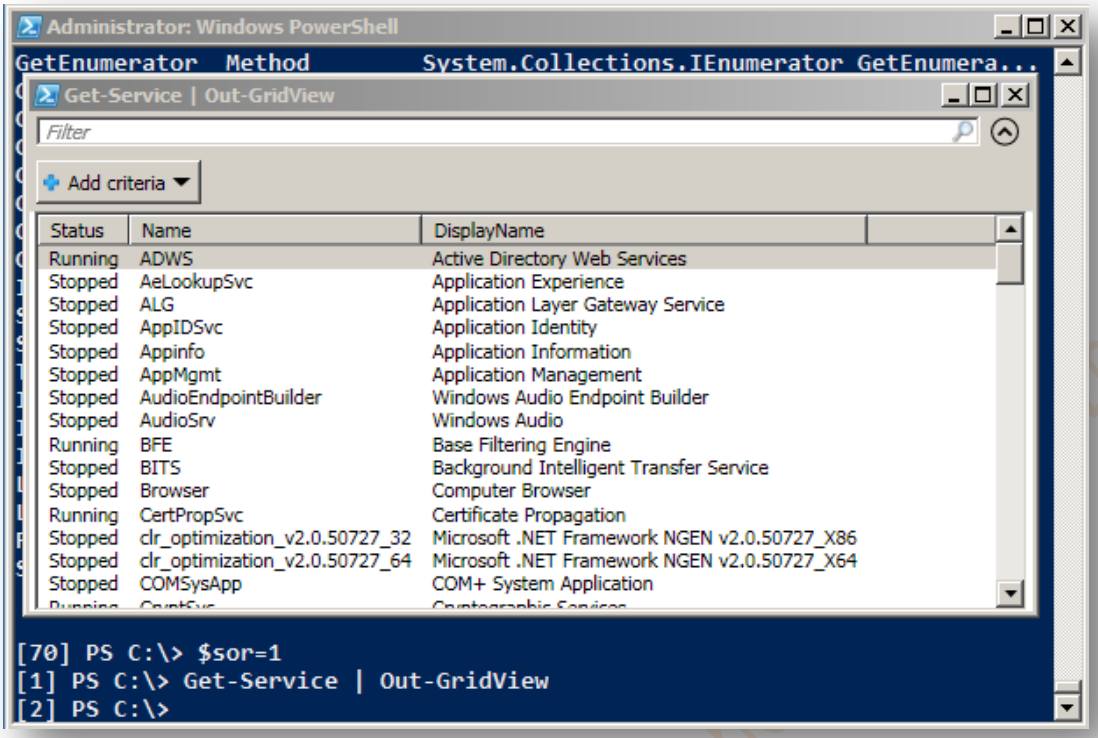
```
   PSIsContainer: True


Lurdy                                          munka
PerfLogs                                       Program Files
Program Files (x86)                            reflector
sokfájl                                        Users
Windows


   PSIsContainer: False


fájl.txt                                       kon.psc1
obj.xml                                        service.html
```

Ebben a példában a `PSIsContainer` tulajdonság alapján csoportosítást is kértem.

1.9.8 Kimenet megjelenítése grafikus rácsban (Out-GridView)

Az előzőekben is láthattuk, hogy a PowerShell cmdletek kimente gyakran táblázatos formátumú. Sajnos csak akkor látjuk, hogy nem férnek ki megfelelően az oszlopokban található információk, vagy hogy nekünk rossz sorrendben olvashatóak a sorok, amikor már lefuttattuk a parancssorunkat. Ilyenkor előhívva a történettárból a parancssort lehet javítgatni, alakítgatni, sorrendet változtatni, de ha ez egy hosszadalmasabban futó kód, akkor elég bosszantó a várakozás, amíg a kívánt kimenetet megkapjuk. Ennek felgyorsítására született a grafikus rács kimenet a PowerShell 2.0-ban. Ennek cmdletje az `Out-Gridview`, amit nézzünk meg egy egyszerű példa segítségével az alábbiakban:

A könyv tanfolyami felhasználása nem engedélyezett!



39. ábra Out-GridView rácsa

Az így megjelenő rács – bár nincs menüje – rengeteg kényelmi szolgáltatást nyújt számunkra:

- Teljes szöveges kereső: a felső kereső a teljes táblázatban keresve leszűkíti a listát azokra a sorokra, ahol a begépelt karaktersorozat megtalálható
- Sorba rendezés: oszlopfejlécekre kattintva az adott oszlop tartalma szerint sorba rendezve jeleníti meg a sorokat
- Összetett szűrő: az oszlopokban megjelenített tulajdonságok értékeire beállított szűrőfeltételekkel lehet szűkíteni a megjelenített sorokat

Sajnos az `Out-GridView` csak azokat az oszlopokat jeleníti meg alaphelyzetben, amik az adott osztály táblázatos nézetében is alaphelyzetben megjelennek. Ha az összes tulajdonságot meg akarjuk jeleníteni, akkor egy `Select-Object` kifejezést is be kell illesszünk a csővezetékünkbe:

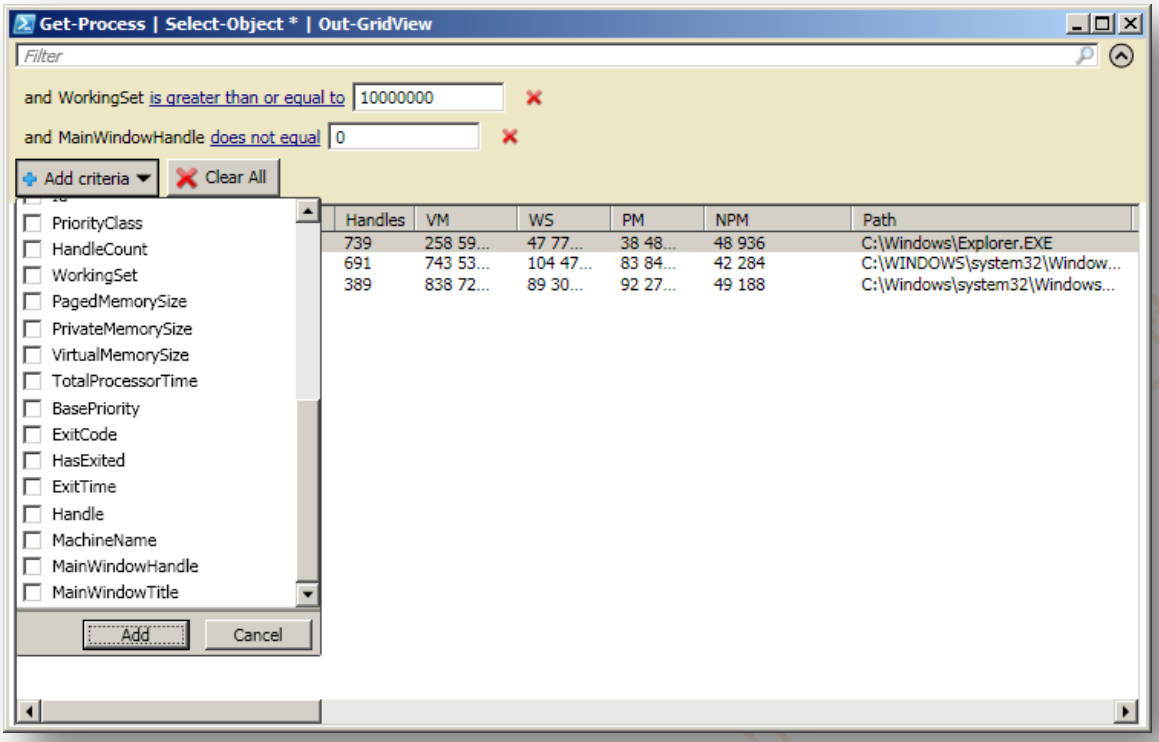
```
PS C:\> get-service | select-object -Property * | Out-GridView
```

Ez egy picit nehézkessé teszi a használatát.

Nézzünk egy példát az összetett szűrő használatára:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



40. ábra GridView összetett lekérdezője

1.9.9 Gyűjtemények összehasonlítása (Compare-Object)

A `Compare-Object` cmdlet segítségével két tetszőleges gyűjteményt hasonlíthatunk össze, kimenetül a gyűjtemények közötti különbséget leíró objektumokat kapunk. Az alábbi példában egy változóba mentettem a gépen futó folyamatok listáját. Ezután leállítottam, illetve elindítottam néhány folyamatot, majd ezt az új állapotot egy másik változóba írtam. A `Compare-Object`-nek odaadtam a két változót, ő pedig kilistázta a különbségeket:

```
[44] PS C:\> $a = Get-Process # bezár notepad, xmlnotepad
[45] PS C:\> $b = Get-Process # megnyit másik notepad példány, mspaint
[46] PS C:\> Compare-Object $a $b

InputObject                              SideIndicator
-----------                              -------------
System.Diagnostics.Process (mspaint)     =>
System.Diagnostics.Process (XmlNo...     <=
```

A [44]-es sor futtatásakor a Notepad egy példánya futott, meg az XMLnotepad program. A [45]-ös sor futtatása előtt becsuktam a Notepadet és újra megnyitottam (másik processz lett belőle), és becsuktam az XMLnotepad-et és megnyitottam az MSPaint-et. A [46]-os sorban összehasonlítottam a két állapotban mintavételezett processzek listáját. Valamilyen szempontból jó eredményt kaptunk, de ha igazán belegondolunk, és precízek szerettünk volna lenni, akkor ez mégsem jó eredmény, hiszen a két notepad.exe folyamat az nem ugyanaz. Vajon hogyan gondolkodott a PowerShell? Szegény `compare-object` bármilyen bonyolult objektumok gyűjteményét kaphatja paraméterként, így ha az objektumok összes tulajdonságának

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

összehasonlításával döntené el az egyezőséget, akkor nagyon sokat kellene dolgoznia. Így alaphelyzetben nagyon egyszerű algoritmust alkalmaz: veszi az objektumok szöveggé alakított formáját. A `ToString` metódus minden objektumnál kötelező elem, így ez garantáltan meghívható. Nézzük meg, hogy ez mit az a fenti esetben:

```
[47] PS C:\> $a | ForEach-Object {$_.tostring()}
System.Diagnostics.Process (conhost)
System.Diagnostics.Process (csrss)
System.Diagnostics.Process (csrss)
System.Diagnostics.Process (dfsrs)
System.Diagnostics.Process (dfssvc)
```

Hiszen pont ilyen adatokat láthatunk a [46]-os sor futtatása után, és ebben tényleg csak a processz objektumok típusa és neve látszik, azaz elrejtődik, ha egy processzt újra nyitunk. Hogyan lehetne precízebbé tenni a `compare-object`-et? Használjuk a `-property` paramétert, ahol felsorolhatjuk, hogy pontosan mely tulajdonság(ok) összehasonlításán alapuljon az egyes objektumok egyformaságának eldöntése. A mi esetünkben legyen a processz azonosító és a processz neve:

```
[48] PS C:\> Compare-Object $a $b -Property id, name

                      id name                      SideIndicator
                      -- ----                      -------------
                    2076 mspaint                   =>
                    1632 notepad                   =>
                    2944 notepad                   <=
                     396 XmlNotepad                <=
```

Ez már precízebb eredményt adott!

Mire használható ez még? Szeretnénk például megtudni, hogy az internetről gyűjtött csodaprogram telepítője pontosan mit garázdálkodik a gépünkön? Semmi gond, készítsünk pillanatfelvételt az érzékeny területekről (futó folyamatok, fájlrendszer, registry) a telepítés előtt, majd hasonlítsuk össze a telepítőprogram lefutása utáni állapottal. Lesz nagy meglepetés! Nem kell elaprózni, bátran lekérhetjük például a teljes c: meghajtó állapotát, a gép majd beleizzad kicsit az összehasonlításba, de így mindenre fény derül:

```
PS C:\> $a = Get-ChildItem c: -recurse
PS C:\> $b = Get-ChildItem c: -recurse
PS C:\> Compare-Object $a $b
```

Vigyázzunk azonban a `compare-object`-tel! Ha túl sok a különbség a két gyűjtemény között, akkor elég sokáig eltarthat az összehasonlítgatás, hiszen az első kupac minden eleméhez megnézi, hogy van-e egyező elem a másik kupacban. Ha mindkét kupac közel azonos számú elemből áll és az elemek sorrendben vannak, akkor használhatjuk `-SyncWindow` paramétert, mellyel leszűkíthetjük azt a tartományt, amin belül egyezést keres a másik kupacban. Nézzünk erre egy példát:

```
[56] PS C:\> Compare-Object 1,2,3 3,4,5 -SyncWindow 1

                       InputObject SideIndicator
                       ----------- -------------
                                 3 =>
                                 1 <=
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                                    4 =>
                                    2 <=
                                    5 =>
                                    3 <=
```

A fenti példában az egyik gyűjteményem 1-től 3-ig a számok, a másik gyűjteményem 3-tól 5-ig. Azaz a 3 valójában nem különbség a két gyűjtemény között, mégis az eredményben, ami ugye az eltéréseket adja meg, a 3-as is szerepel, ráadásul kétszer is. Ennek az az oka, hogy a `compare-object` 1-es `synwindow` paraméterrel nem minden elemhez néz meg minden elemet, hanem alaphelyzetben ±1 elem távolságra. Azaz az első halmaz 3-asát összehasonlítja a második tömbbeli 4-gyel, 5-tel, de az ottani első 3-assal már nem. A `compare-object` alaphelyzetben ±maxint távolságra vizsgál, azaz gyakorlatilag minden elemhez minden elemet megkeres:

```
[57] PS C:\> Compare-Object 1,2,3 3,4,5

                        InputObject SideIndicator
                        ----------- -------------
                                  4 =>
                                  5 =>
                                  1 <=
                                  2 <=
```

Így megtalálta, hogy mindkét halmazban benne van a 3-as.

1.9.10 Különböző objektumok (Get-Unique)

Sokszor előfordulhat, hogy egyedi objektumokat keresünk, azaz a duplikációkra nem vagyunk kíváncsiak. Például a futó processzek esetében ugyanazokat nem akarom többször látni. Alaphelyzetben ezt a listát kapom:

```
[45] PS C:\>Get-Process | sort

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
...
      0       0        0         28     0                0 Idle
    758      20    28864      24160   162    29,53   2504 iexplore
    807      24    54592      28636   207    93,50   1296 iexplore
     97       3      740       2864    23     0,09   2788 igfxpers
     80       3     1216       3572    34     0,16   2532 igfxtray
    473      10     4284       1444    43     7,66   1016 lsass
...
```

Nézzük csak az egyedieket a `get-unique` segítségével:

```
[47] PS C:\>Get-Process | get-unique | sort

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
...
      0       0        0         28     0                0 Idle
    807      24    54592      28636   207    93,50   1296 iexplore
     97       3      740       2864    23     0,09   2788 igfxpers
```

```
     80       3     1216       3572    34     0,16   2532 igfxtray
    481      11     4316       1880    43     7,66   1016 lsass
...
```

Látszik, hogy az `iexplorer` csak egyszer szerepel a második listámban.

Vajon mit lehet tenni akkor, ha pont a duplikáltakra vagyok kíváncsi? Sajnos nincs „`get-duplicate`" cmdlet, de ilyen jellegű működést mi magunk is elő tudunk idézni a `get-unique` és az előbb látott `compare-object` ötvözetével:

```
[55] PS C:\>$elemek = "a","b","c","d","e","a","b","a"
[56] PS C:\>Compare-Object ($elemek) ($elemek| sort | Get-Unique)

InputObject                              SideIndicator
-----------                              -------------
a                                        <=
b                                        <=
a                                        <=
```

Nézzük meg ugyanezt a processzekre:

```
[57] PS C:\> Compare-Object (Get-Process ) (Get-Process| Get-Unique)

InputObject                            SideIndicator
-----------                            -------------
System.Diagnostics.Process (conhost) <=
System.Diagnostics.Process (csrss)   <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
System.Diagnostics.Process (svchost) <=
```

Természetesen a `Get-Unique` is az adott objektumon végzett `ToString()` metódus alapján dönti el az egyediséget, hasonlóan, ahogy a `Compare-Object`-nél láttuk, azzal a különbséggel, hogy ott meg lehetett adni, hogy mely tulajdonságok alapján vizsgálja az egyformaságot, ha az alapműködés nem tetszett nekünk, itt sajnos ilyet nem lehet.

1.9.11 Számlálás (Measure-Object)

Tetszőleges objektumcsoport elemeivel kapcsolatos összegzést átlagolást, stb. végezhetünk el a `Measure-Object` cmdlet segítségével. A cmdlet két különböző üzemmódban használható, és ennek megfelelően két különböző típusú objektumot adhat kimenetül. Szöveges üzemmódot a `-line`, `-word`, `-character` paraméterek valamelyikével kérhetünk, ekkor a bemenetként érkező objektumok szöveges reprezentációjából származó értékek kerülnek a kimenetbe, vagyis a cmdlet megszámolja a szövegben található sorokat, szavakat és karaktereket.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Például, számoljuk meg egy fájlba kitett könyvtárlista `dir.txt` sorait, szavait és karaktereit! A megoldás egyszerűen a következő (az `-ignorewhitespace` paraméter használata esetén a szóközök és tabulátorok nem számítanak bele a karakterek számába):

```
PS C:\> get-content c:\dir.txt | measure-object -ignorewhitespace -line -word -
char | format-table -autosize

Lines Words Characters Property
----- ----- ---------- --------
   13    55        403
```

Ha a `-property` paraméter után megadjuk a bemenő objektumok valamelyik tulajdonságának nevét, akkor a kimeneten a megadott tulajdonság összege, átlaga, maximális és minimális értéke fog megjelenni.

Például összegezzük a rendszerfolyamatok fizikai memóriafoglalását! A megoldás egyszerűen a következő:

```
PS C:\> get-process | measure-object -property workingset -sum -average -m
ax -min


Count    : 42
Average  : 4400664,38095238
Sum      : 184827904
Maximum  : 80818176
Minimum  : 16384
Property : WorkingSet
```

A `count` sorban lévő szám ebben az esetben azt jelenti, hogy 42 darab „workingset” tulajdonságot átlagolt, illetve összegezett a cmdlet, vagyis ennyi Process objektumot kapott bemenetként.

Ha csak a fenti statisztika egyik számadatával szeretnénk dolgozni, akkor a szokásos tulajdonsághivatkozással ezt is megtehetjük. Például szeretném a fenti memóriafoglalást megabájtokban kijelezni:

```
[7] PS I:\>(get-process | measure-object -property workingset -sum).sum / 1m
b
968,5390625
```

1.9.12 Nyomtatás (Out-Printer)

Az `Out-Printer` cmdlet a bemenetként kapott adatokat az alapértelmezett, vagy a paraméterként megadott nyomtatóra írja.

```
PS C:\> Get-Process | Out-Printer \\server\HPLJ5si
```

1.9.13 Kiírás fájlba (Out-File, Export-)

Alapértelmezés szerint a PowerShell parancsok kimenete (vagyis a kimenetként kapott objektumok szöveges reprezentációja) a konzolablakban jelenik meg. A szöveges kimenet fájlba irányítását az `Out-File` és a `Set-Content` cmdlet segítségével végezhetjük el, amelyek paramétereként a létrehozandó fájl nevét

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

kell megadnunk. Az `Export-Csv` és az `Export-CliXML` cmdletek nevükhöz méltóan csv, illetve xml formátumban írják fájlba a bemenetül kapott objektumok adatait.

Példaként készítsünk szövegfájlt, csv és xml fájlt a c: meghajtó mappalistájából! A megoldás mindhárom esetben nagyon egyszerű, viszont az eredmény a formátumon túl, a további felhasználhatóság szempontjából is lényegesen különbözik egymástól.

```
PS C:\> Get-ChildItem | Out-File c:\dir.txt
```

Ebben az esetben a fájlba csak az kerül, amit az eredeti parancs a képernyőre írt volna. Ha a fájlt visszaolvassuk (`Get-Content`), az eredeti kimenet sorainak megfelelő karakterláncokat kapunk.

A kimeneti fájlban alaphelyzetben 80 karakter szélességűre tördelt sorokat kapunk, ha ennél szélesebb sorokat szeretnénk, akkor használjuk a `-width` paramétert.

```
[27] PS C:\>Get-ChildItem c:\old | ft -property * -auto | Out-File c:\old\di
r2.txt -Width 800
```

A fenti példában a fájllistámat táblázatos formában akarom kirakni egy szöveges állományba, de az összes tulajdonsággal. Ez jó széles táblázatot eredményez, így az `out-file`-nál egy brutálisan széles sorméretet adok meg lehetséges értéknek. De hogy feleslegesen ne terpeszkedjen szét a táblázatom, ezért használtam a `format-table` cmdletnél az `-auto` kapcsolót, mert ez olyan sűrűn teszi az oszlopokat, amilyen sűrűn adatvesztés nélkül lehet. Így tehát kaptam egy optimális szélességű szöveget, amely ténylegesen csak 399 karakter széles lett, azaz nem kellett kihasználni a 800 karakteres maximumot.

Nézzük meg, hogy még milyen formátumokba tudjuk kitenni fájlba az outputot. A következő a csv formátum, ami táblázatos forma:

```
PS C:\> Get-ChildItem | Export-Csv c:\dir.csv
```

Ha belenézünk az eredményül kapott csv fájlba (Notepad, vagy Excel is jó), akkor örömmel láthatjuk, hogy ebben az esetben az objektumok minden tulajdonsága belekerült a kimenetbe, sőt a fejlécben még a típus megnevezése is megtalálható. Van egy `Import-Csv` cmdletünk is, ami a fájlt visszaolvasva újra létrehozza az eredeti objektumot, jobban mondva egy ahhoz hasonló objektumot. Próbáljuk is ki:

```
PS C:\> Import-Csv c:\dir.csv


PSPath            : Microsoft.PowerShell.Core\FileSystem::C:\Config.Msi
PSParentPath      : Microsoft.PowerShell.Core\FileSystem::C:\
PSChildName       : Config.Msi
PSDrive           : C
PSProvider        : Microsoft.PowerShell.Core\FileSystem
PSIsContainer     : True
Mode              : d----
```

Az eredmény érdekesen néz ki, viszont sajnos nem hasonlít túlságosan az eredeti mappalistára. Egy `Get-Member` hívás segítségével megtudhatjuk, hogy mi is történt az adatainkkal:

```
PS C:\> Import-Csv c:\dir.csv | get-member
```

folyamatfejlesztés nem engedélyezett!

```
   TypeName: CSV:System.IO.DirectoryInfo

Name              MemberType   Definition
----              ----------   ----------
Equals            Method       System.Boolean Equals(Object obj)
GetHashCode       Method       System.Int32 GetHashCode()
GetType           Method       System.Type GetType()
ToString          Method       System.String ToString()
Attributes        NoteProperty System.String Attributes=Directory
CreationTime      NoteProperty System.String CreationTime=2006.11.15. 8...
CreationTimeUtc   NoteProperty System.String CreationTimeUtc=2006.11.15...
Exists            NoteProperty System.String Exists=True
Extension         NoteProperty System.String Extension=
FullName          NoteProperty System.String FullName=C:\752e9949d08195...
LastAccessTime    NoteProperty System.String LastAccessTime=2007.07.06....
...
```

Látható, hogy nem az igazi `DirectoryInfo` típust kaptuk vissza (ráadásul a `FileInfo` típus teljesen eltűnt), hanem csak valamiféle pótlékot, az eredeti objektum leegyszerűsített változatát, amelynek nincsenek metódusai (csak amelyek az `Object` osztályból öröklődnek, ezekkel minden objektum rendelkezik), tulajdonságai pedig elvesztették eredeti típusukat. Minden tulajdonságérték megvan ugyan, de csak karakterláncként, az eredeti típusinformáció tárolására a csv formátum nem képes.

Ha kevesebb információvesztéssel szeretnénk tárolni és visszaolvasni objektumainkat, akkor az xml formátumot kell választanunk.

```
PS C:\> Get-ChildItem | Export-CliXML c:\dir.xml
```

Az xml fájl mérete közel tízszerese a csv-nek, de ebből pontosan az eredeti adathalmaz tulajdonságai olvashatók vissza, mindenféle veszteség, vagy torzulás nélkül. Ilyen módon tehát tetszőleges .NET objektum, gyűjtemény, akármi teljes tartalmát lemezre menthetjük, és később, a fájlt egyetlen paranccsal visszaolvasva helyreállíthatjuk az eredeti objektumot[11]. Nem rossz!

```
PS C:\> Import-CliXML c:\dir.xml


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
            2007.06.10.     12:41            Config.Msi
            2007.06.13.     21:30            Documents and Settings
            2003.10.25.     18:59            Inetpub
            2007.06.04.     20:36            Program Files
```

Azért egy kis csalás itt is van. Ha megnézzük `get-member`-rel, hogy mit is kaptunk, akkor kiderül, hogy itt sem ugyanolyan objektumtípust kaptunk vissza, például a metódusokat ez is elveszítette:

```
[13] PS C:\old>import-clixml C:\dir.xml | gm
```

[11] A Mode oszlop azért hiányzik, mert az eredeti .NET objektumnak sincs ilyen tulajdonsága, ezt csak a PowerShell hazudja oda.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
   TypeName: Deserialized.System.IO.DirectoryInfo

Name              MemberType   Definition
----              ----------   ----------
PSChildName       NoteProperty System.String PSChildName=alfolder
PSDrive           NoteProperty System.Management.Automation.PSObject PSD...
PSIsContainer     NoteProperty System.Boolean PSIsContainer=True
PSParentPath      NoteProperty System.String PSParentPath=Microsoft.Powe...
PSPath            NoteProperty System.String PSPath=Microsoft.PowerShell...
PSProvider        NoteProperty System.Management.Automation.PSObject PSP...
Attributes        Property     System.String {get;set;}
CreationTime      Property     System.DateTime {get;set;}
CreationTimeUtc   Property     System.DateTime {get;set;}
Exists            Property     System.Boolean {get;set;}
Extension         Property     System.String {get;set;}
FullName          Property     System.String {get;set;}
LastAccessTime    Property     System.DateTime {get;set;}
LastAccessTimeUtc Property     System.DateTime {get;set;}
LastWriteTime     Property     System.DateTime {get;set;}
LastWriteTimeUtc  Property     System.DateTime {get;set;}
Name              Property     System.String {get;set;}
Parent            Property     System.String {get;set;}
Root              Property     System.String {get;set;}


   TypeName: Deserialized.System.IO.FileInfo

Name              MemberType   Definition
----              ----------   ----------
PSChildName       NoteProperty System.String PSChildName=alice.txt
PSDrive           NoteProperty System.Management.Automation.PSObject PSD...
PSIsContainer     NoteProperty System.Boolean PSIsContainer=False
PSParentPath      NoteProperty System.String PSParentPath=Microsoft.Powe...
PSPath            NoteProperty System.String PSPath=Microsoft.PowerShell...
PSProvider        NoteProperty System.Management.Automation.PSObject PSP...
Attributes        Property     System.String {get;set;}
CreationTime      Property     System.DateTime {get;set;}
CreationTimeUtc   Property     System.DateTime {get;set;}
Directory         Property     System.String {get;set;}
DirectoryName     Property     System.String {get;set;}
Exists            Property     System.Boolean {get;set;}
Extension         Property     System.String {get;set;}
FullName          Property     System.String {get;set;}
IsReadOnly        Property     System.Boolean {get;set;}
LastAccessTime    Property     System.DateTime {get;set;}
LastAccessTimeUtc Property     System.DateTime {get;set;}
LastWriteTime     Property     System.DateTime {get;set;}
LastWriteTimeUtc  Property     System.DateTime {get;set;}
Length            Property     System.Int64 {get;set;}
Name              Property     System.String {get;set;}
```

Azaz az `Export-CliXML` és `Import-CliXML` párossal az objektumot tulajdonságai megőrződnek, ugyanúgy felhasználhatjuk ezeket műveletek végzésére, mint az eredeti objektumok tulajdonságait, viszont a metódusok elvesztek, ha azokra is szükségünk lenne, akkor sajnos azon melegében, még exportálás előtt meg kell hívnunk ezeket.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.9.14 Egyéni objektumok létrehozása CSV adatokból (ConvertFrom-CSV)

Gyakran előfordul, hogy teljesen egyedi adataink vannak, amelyek még csak nem is hasonlítanak a PowerShell, de még csak a .NET által használt típusokhoz sem, viszont ha tudnánk belőlük speciális típusú objektumokat létrehozni, akkor fel tudjuk használni kezelésükre a számos PowerShell cmdletet. Nézzünk például egy egyszerű futóverseny eredménylistáját, az lenne a feladat, hogy állapítsuk meg az első három helyezettet. Legyen a kiinduló adatunk egy CSV formátumú fájlban:

```
[76] PS C:\> Get-Content C:\munka\futók.txt
Név, Idő
Béla, 5:12
Dezső, 5:37
Karcsi, 5:36
Marci, 5:49
Tódor, 5:25
Tibi, 4:45
Gabi, 5:18
Andris, 4:52
Bence, 4:48
```

Nem csalás, nem ámítás, ez tényleg egy csv formátumú fájl. Ez soronként egy-egy sztring, ebből szövegként kibányászni az idő paramétert elég nehéz lehet. Szerencsére erre nincs szükség, mert van nekünk egy `ConvertFrom-Csv` cmdletünk, ami pont az átalakítást elvégzi:

```
[77] PS C:\> Get-Content C:\munka\futók.txt | ConvertFrom-Csv

Név                                  Idő
---                                  ---
Béla                                 5:12
Dezső                                5:37
Karcsi                               5:36
Marci                                5:49
Tódor                                5:25
Tibi                                 4:45
Gabi                                 5:18
Andris                               4:52
Bence                                4:48
```

Itt már láthatjuk, hogy strukturálttá vált az adathalmazunk, ezt már a korábban látott PowerShell cmdletekkel fel lehet dolgozni:

```
[78] PS C:\> Get-Content C:\munka\futók.txt | ConvertFrom-Csv | Sort-Objec
t -Property idő | Select-Object -First 3

Név                                  Idő
---                                  ---
Tibi                                 4:45
Bence                                4:48
Andris                               4:52
```

Egy picit csaltam, hiszen az idő azért nem igazi időformátum:

```
[79] PS C:\> (Get-Content C:\munka\futók.txt | ConvertFrom-Csv)[1].idő.get
```

```
type()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object
```

Ha teljesen precíz akarok lenni, akkor érdemes változóba tölteni az eredményeket és akkor lehetőségünk van adattranszformációt végezni:

```
[83] PS C:\> $futók = Get-Content C:\munka\futók.txt | ConvertFrom-Csv
[84] PS C:\> $futók | ForEach-Object { $_.idő = [timespan] $_.idő}
[85] PS C:\> $futók

Név                                Idő
---                                ---
Béla                               05:12:00
Dezső                              05:37:00
Karcsi                             05:36:00
Marci                              05:49:00
Tódor                              05:25:00
Tibi                               04:45:00
Gabi                               05:18:00
Andris                             04:52:00
Bence                              04:48:00
[86] PS C:\> $futók[0].idő.gettype().fullname
System.TimeSpan
```

Itt már tényleg idő típusú a futók időeredménye, de itt most ez a sorba rendezésben nem játszott szerepet.

1.9.15 Táblázatok kezelése (Import-Csv, ConvertFrom-Csv, ConvertTo-Csv)

Előrebocsátom, hogy nem Excel fájlok kezeléséről lesz itt szó, hanem olyan szövegfájlokról, amelyek valamilyen elválasztójellel értékeket tartalmaznak. Ilyet elő tudunk állítani a korábban már látott `Export-Csv` cmdlettel, de természetesen bármilyen más forrásból származó CSV fájlokat és adatokat tudunk ezekkel a cmdletekkel kezelni.

Az `Import-Csv`, `Export-Csv` cmdletek párja a `ConvertTo-Csv` és `ConvertFrom-Csv` cmdletek, majdnem ugyanolyan a paraméterezésük, csak az első kettő fájlokkal dolgozik, a második kettő meg a memóriában, változókban található adatokkal. A CSV szó szerint „comma separated values”, azaz „vesszővel elválasztott értékek” jelentéssel bír, de szerencsére nem csak vessző, hanem tetszőlegesen megadható elválasztó karakterrel tudnak dolgozni. Nézzünk erre egy példát. Van egy eredetileg vesszőkkel elválasztott fájlunk, szeretnénk kicserélni a vesszőket mondjuk pontosvesszőre:

```
[3] PS C:\munka> Import-Csv .\futók.txt | ConvertTo-Csv -NoTypeInformation -Del
imiter ";"
"Név";"Idő"
"Béla";"5:12"
"Dezső";"5:37"
"Karcsi";"5:36"
"Marci";"5:49"
"Tódor";"5:25"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
"Tibi";"4:45"
"Gabi";"5:18"
"Andris";"4:52"
"Bence";"4:48"
```

A `-NoTypeInformation` kapcsolóval nem rak egy extra kezdősort a kimenetre, amely az adatok típusát hivatott volna jelezni.

1.9.16 Átalakítás szöveggé (Out-String)

Láthattuk, hogy jó dolog az objektumorientált megközelítés, az esetek döntő többségében jelentősen egyszerűbbé vált tőle az élet a hagyományos shellekkel összehasonlítva. Bizonyos esetekben mégis szükség lehet arra, hogy az objektumokból álló kimenetet szövegként dolgozzuk fel, és például egy szöveges keresés alapján válogassuk ki belőle azokat a sorokat, amelyekre szükségünk van. Nem szerencsés például az objektumos kimenet, ha az, amit keresünk, több tulajdonság értékében is előfordulhat, és nekünk azok a sorok kellenek, amelyekben akárhol is, de előfordul (vagy éppen hiányzik) a keresett érték.

Listázzuk ki egy `Get-Process` alapértelmezett kimenetéből azokat a sorokat, amelyekben (bármelyik oszlopban!) előfordul a „44” karakterlánc! Az objektumok szöveges változatának előállítására az `Out-String` cmdlet képes. Bemenetére tetszőleges objektumokat küldhetünk (például formázott táblázatot, listát, stb. is), a kimenetén egy hosszú karakterlánc fog megjelenni, amely mindazt tartalmazza, amit a bemenet a képernyőre írt volna. Próbáljuk ki a következőt:

```
PS C:\> get-process | out-string


Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     66       3      864       2808    30     0,08    416 ctfmon
    173       5     3564       5340    33     0,19   1100 CTHELPER
     29       1      380        756    15     0,02   1436 CTSVCCDA
    473       6     2144       2420    28    11,45    612 csrss
    689      18    25440      18048   107    50,84   3804 explorer
```

Látszólag semmi különbség nincs az önálló `Get-Process`-hez képest, de ha a kimenetre kérünk egy `Get-Member`-t, akkor látható, hogy az már nem `System.Diagnostics.Process` objektumokból áll, hanem egyszerű karakterlánccá alakult.

Már csak a szöveges keresés van hátra, amelyet a `Select-String` cmdlet segítségével fogunk elvégezni (ezzel részletesen a *2.5.4 Szövegfájlok feldolgozása (Get-Content, Select-String)* fejezetben lesz szó):

```
PS C:\> get-process | out-string | select-string -pattern "44"
```

Valami még nem egészen kerek, mivel ismét csak a teljes listát kaptuk vissza. Mi lehet a baj? Az `Out-String` cmdlet alapértelmezés szerint **egyetlen** karakterláncot ad vissza, ami a teljes listát tartalmazza a sortörésektől függetlenül. A teljes listában persze ott van a keresett minta, így az egyetlen karakterlánc közül azt az egyet kiírtuk a képernyőre. Soronként tördelt kimenetet (karakterláncokból álló tömböt) a `-stream` paraméter használatával kérhetünk, a helyes megoldás tehát:

```
PS C:\> get-process | out-string -stream | select-string -pattern "44"
```

A könyv tanfolyami felhasználása nem engedélyezett!

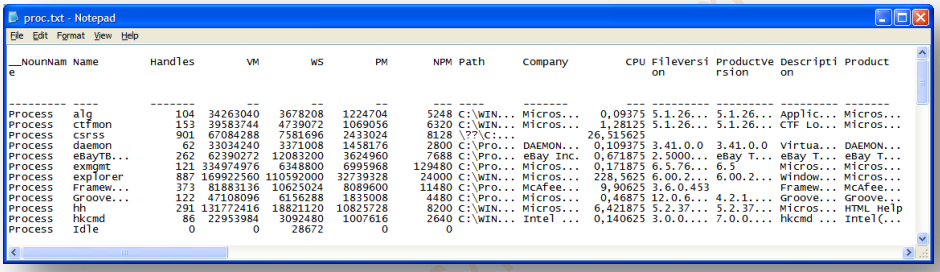
© Soós Tibor – soost@ IQJB.hu

```
    37       2     2016          40     30     0,09   3144 cmd
    63       3     1068         372     30     0,09    844 daemon
   779      20    67944        7200    198    19,30   2496 iexplore
    21       1      168         144      4     0,08    920 smss
   162       7     4960        2652     60     0,44   1968 spoolsv
   142       4     2396         372     44     0,50    640 VsTskMgr
```

Az `out-string`-nek van még egy praktikus paramétere, ez pedig a `-width`. Ez akkor jöhet jól, ha nagyon széles táblázatot akarunk megjeleníteni, és az alaphelyzet szerinti 80 karakteres szélesség túl kevésnek tűnik. Mi van akkor, ha a `get-process`-szel a folyamatok összes tulajdonságát meg akarom jeleníteni. Ez 80 karakterben gyakorlatilag reménytelen, nézzük meg, hogy 600 karakterbe hogyan fér el:

```
[19] PS C:\>get-process | ft -Property * | out-string -Width 600 -Stream > c
:\old\proc.txt
```

Ezek után nézzük Notepad-del a fájlt:



41. ábra 600 karakter széles szöveg Notepadben

A sortörést kikapcsoltam, és látszik, hogy viszonylag jól elférünk már. Az alsó gördítő sáv méretéből látható, hogy elég jócskán lehet vízszintesen görgetni ezt a táblázatot. Ezzel gyakorlatilag ugyanazt az eredményt értem el, mint az `out-file` cmdlet használatával, megfelelő `-width` paraméterrel.

1.9.17 Lista-tulajdonságok módosítása (Update-List)

Majd az ActiveDirectory-nál fogjuk látni, hogy nagyon gyakori az, hogy egy objektumnak olyan tulajdonsága van, ami nem egy konkrét érték, hanem az értékeknek egy tömbje. Ezt az ActiveDirectory-nál *multivalued property*-nek hívjuk. Ezeket többféleképpen lehet módosítani: hozzáadunk a listához egy újabb tulajdonságértéket, elveszünk egy tulajdonságértéket vagy eldobjuk az eddigi értékeket és valami mást töltünk be helyette. Nézzünk erre egy példát, egyelőre még nem ActiveDirectory objektumokkal, az `Update-List` cmdlet segítségével:

```
[52] PS C:\> $ember = New-Object psobject
[53] PS C:\> $ember | Add-Member -Name Név -Value "Kovács Ödön" -MemberTyp
e noteproperty
[54] PS C:\> $tul = New-Object system.collections.arraylist
[55] PS C:\> $tul.add("okos")
0
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[56] PS C:\> $tul.add("ügyes")
1
[57] PS C:\> $ember | Add-Member -Name Tulajdonságok -Value $tul -MemberTy
pe noteproperty
[58] PS C:\> $ember

Név                                 Tulajdonságok
---                                 -------------
Kovács Ödön                         {okos, ügyes}
```

Készítettem egy egyedi objektumot az `$ember` változóba [52]. Hozzáadtam egy `Név` tulajdonságot [53], és definiáltam egy `$tul` változót, ami bővíthető tömbtípusú [54]. Ebbe a `$tul` változóba betöltöttem két értéket [55], [56], majd ezt is hozzáadtam az `$ember`-emhez [57]. Legvégén kiírattam az `$ember` tartalmát [58], látszik, hogy ott van a „multivalued property” jellegű `Tulajdonságok` nevű tulajdonság. Ez tehát az alaphelyzet.

Hogyan lehetne ehhez a tulajdonságlistához további értékeket betölteni?

```
[61] PS C:\> $ember | Update-List -Property Tulajdonságok -Add "szép", "sz
erény"

Név                                 Tulajdonságok
---                                 -------------
Kovács Ödön                         {okos, ügyes, szép, szerény}
```

Azaz nem nekem kellett külön tömbműveletekkel kibővíteni a `Tulajdonságok` tömböt, hanem ezt elvégezte helyettem az `Update-List`.

Vagy hogyan lehet eltávolítani egy tulajdonságértéket?

```
[62] PS C:\> $ember | Update-List -Property Tulajdonságok -Remove "okos"

Név                                 Tulajdonságok
---                                 -------------
Kovács Ödön                         {ügyes, szép, szerény}
```

Ezt még körülményesebb lenne tömbműveletekkel elvégezni, így meg ez is nagyon egyszerű. Nézzünk egy teljes tulajdonság-cserét:

```
[63] PS C:\> $ember | Update-List -Property Tulajdonságok -Replace "buta",
 "csúnya"

Név                                 Tulajdonságok
---                                 -------------
Kovács Ödön                         {buta, csúnya}
```

Ez a cmdlet tehát csak tulajdonságként szereplő tömböket tud módosítani, közvetlenül tömbelemeket nem tud kiszedni, betenni vagy kicserélni. A másik előfeltétel, hogy tömb nem lehet fix méretű, azaz a PowerShell által használt alap tömbtípus nem jó erre a célra.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.9.18 Kimenet törlése (Out-Null)

Bizonyos feldolgozások során (elsősorban automatikusan futtatott szkriptekben) zavaró lehet a konzolon megjelenő kimenet. Ebben az esetben tehet jó szolgálatot az `Out-Null` cmdlet, amelyet a sor végére biggyesztve mindenféle esetleg megjelenő üzenettől szövegtől (kivéve a hibaüzeneteket) megszabadulhatunk.

A könyv tanfolyami felhasználása nem engedélyezett!

1.10 Futtatás háttérben

A PowerShell 2.0 egyik újdonsága, hogy képes kifejezéseket háttérben futtatni. Korábban, ha egy időigényes kifejezést futtattunk, akkor a konzolunk addig „használhatatlan” volt, amíg a futó program be nem fejezte a tényked ését. Most viszont nyithatunk külön munkamenetet a hosszadalmas tevékenységek számára és a konzolunkon tovább dolgozhatunk. Eközben lekérdezhetjük a háttérben futó programunk státusát, és megtekinthetjük a kimenetét. Nézzük mindezt meg a gyakorlatban!

Az ezzel kapcsolatos cmdletek főneve a „Job”:

```
PS C:\> Get-Command -noun job

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Get-Job                            Get-Job [[-Id] <Int32[]>] ...
Cmdlet          Receive-Job                        Receive-Job [-Job] <Job[]>...
Cmdlet          Remove-Job                         Remove-Job [[-Id] <Int32[]...
Cmdlet          Start-Job                          Start-Job [-ScriptBlock] <...
Cmdlet          Stop-Job                           Stop-Job [[-Id] <Int32[]>]...
Cmdlet          Wait-Job                           Wait-Job [[-Id] <Int32[]>]...
```

A munkát a `Start-Job`-bal kezdhetjük el:

```
PS C:\> start-job -Name Háttérmunka -ScriptBlock {get-process}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
3               Háttérmunka     Running    True            localhost
```

Látható, hogy adhatunk a munkamenetnek egy nevet és a `ScriptBlock` paraméterként átadott futtatható parancssort elindítja a PowerShell egy külön menetben. A `Start-Job` visszatérési értéke maga a „job” objektum. Ennek egyik legfontosabb tulajdonsága a `State`, azaz a státus. Rögtön indítás után természetesen ez még azt mutatja, hogy `Running`, azaz futó állapotban van.

Ha később újra le akarjuk kérdezni ennek a munkamenetnek az állapotát, akkor a `get-job` cmdlettel ezt megtehetjük:

```
PS C:\> get-job -Name Háttérmunka

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
3               Háttérmunka     Completed  True            localhost
```

Ha azt látjuk, hogy a munkamenet tulajdonságai között a `HasMoreData` értéke `$true`, akkor megnézhetjük annak kimenetét is a `Receive-Job` cmdlet segítségével:

```
PS C:\> Receive-Job -Name Háttérmunka

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
```

```
     34       2      568       2132    17      0,04   2548 conhost
     87       4     2016       5976    59      1,18   3148 conhost
    378       5     1160       1308    31      0,59    360 csrss
    202       6     1468       2296   158      1,19    404 csrss
     78       4      912       3320    20               2132 dllhost
     72       4     1056        864    37      0,11     980 dwm
...
```

Ha ezután újra megnézzük a munkamenet tulajdonságait és az már korábban lefutott, és nem lett újabb kimenetünk, akkor a `HasMoreData` értéke `$false` lesz.

Ha egy futó munkamenethez kapcsolódunk a `Receive-Job` segítségével, akkor az aktuális kimenetet kapjuk meg folyamatosan a konzolra, de természetesen egy Ctrl+C billentyűzetkombinációval ez megszakítható – de a munkamenet maga fut tovább! - és később újra „belepillanthatunk” a folyamatba. Ilyenkor az előző `Receive-Job` óta generálódott kimenetet kapjuk meg, szintén `HasMoreData = $true` mellett:

```
PS C:\> start-job -ScriptBlock { for($i=0; $i -lt 60; $i++){$i; Start-Sleep
-Seconds 1}}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
13              Job13           Running    True            localhost


PS C:\> Receive-Job 13
0
1
2
3
4
5
6
7
PS C:\> Receive-Job 13
8
9
10
```

Ha lefutott a munkamenet, akkor a teljes kimenetet újra megnézhetjük a `Receive-Job` segítségével. Szintén újra megkaphatunk egy már korábban kinyert kimenetet, ha a `receive-job` cmdletet a `-keep` kapcsolójával használjuk:

```
PS C:\> start-job -ScriptBlock { for($i=0; $i -lt 60; $i++){$i; Start-Sleep
-Seconds 1}}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
38              Job38           Running    True            localhost

PS C:\> receive-job -id 38 -keep
0
1
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
2
PS C:\> receive-job -id 38 -keep
0
1
2
3
4
```

Ha kifejezetten az a célunk, hogy egy munkamenet végét megvárjuk, akkor használhatjuk a `Wait-Job` cmdletet:

```
PS C:\> start-job -Name Hosszú -ScriptBlock {Get-ChildItem c:\ -Recurse}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
5               Hosszú          Running    True            localhost
PS C:\> Wait-Job -Id 5
```

A `Wait-Job` után csak akkor kapjuk vissza a promptot, ha az adott munkamenet befejeződött.

Egy-egy munkamenet komoly memóriamennyiséget is lefoglalhat, mint például a fenti teljes C meghajtó fájljainak a kilistázása. Ha már nincs szükségünk az eredményre, akkor érdemes a `Remove-Job` cmdlettel eltávolítani a munkamenetet:

```
PS C:\> Remove-Job -Id 5
```

Ha futtatjuk a munkamenetet, de mégsem akarjuk kivárni a befejeződését, akkor megszakíthatjuk a működését a `Stop-Job` cmdlet segítségével:

```
PS C:\> start-job -Name Hosszú -ScriptBlock {Get-ChildItem c:\ -Recurse}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
7               Hosszú          Running    True            localhost


PS C:\> Stop-Job -id 7
PS C:\> Get-Job

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
7               Hosszú          Stopped    False           localhost
```

Hogyan lehet paramétert átadni egy ilyen háttérben futó folyamatnak?

```
[3] PS C:\> $keresendőfájl = "szöveg.txt"
```

Létrehoztam egy változót, amiben egy keresendő fájl nevét tettem.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[4] PS C:\> $job = Start-Job -Name "Keres" -ScriptBlock {Get-ChildItem c:\
munka\$keresendőfájl -Recurse}
[5] PS C:\> $job

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
17              Keres           Completed  True            localhost

[6] PS C:\> Receive-Job 17


    Directory: C:\munka


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---      2009. 11. 11.     21:41     105837 ActiveDirectoryRecycleBin.po
                                              werpack
-a---      2009. 11. 21.     19:45        254 futók.txt
-a---      2009. 11. 18.     17:46         40 script.ps1
-a---      2009. 11. 17.     19:49          7 szöveg.txt
```

Ezután létrehoztam egy háttérben futó folyamatot, amiben a `Get-ChildItem` cmdlettel keresem a keresendő fájlt. A `scriptblock` részben átadtam szándékaim szerint a `$keresendőfájl` változót, ennek ellenére, a [6]-os sorban lekérdezett eredményben a „`munka`” könyvtár össze fájlja szerepelt, nem csak a keresett. Azaz a globális scope-ban létrehozott változóm nem látszott a job számára.

Szerencsére lehet azért paramétert átadni, ezt pedig az `-inputobject` paraméter használatával tehetjük meg:

```
[7] PS C:\> $job = Start-Job -Name "Keres" -ScriptBlock {Get-ChildItem c:\
munka\$input -Recurse} -InputObject $keresendőfájl
[8] PS C:\> Receive-Job $job


    Directory: C:\munka


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---      2009. 11. 17.     19:49          7 szöveg.txt
```

Itt a [7]-es sorban az `-inputobject` paramétereként adom meg a `$keresendőfájl` változót, erre a szkriptblokkban a `$input` változóval tudok hivatkozni. És a [8]-as sor után látható eredményben már tényleg csak a keresett fájlt látjuk.

Háttérfolyamatok tehát főleg olyan parancsok futtatása esetén fontosak, amelyek hosszú ideig futnak. Néhány parancs már előre tudhatóan hosszú ideig fog futni, így ezek esetében van egy `-AsJob` kapcsoló, amely a fenti külön munkamenet létrehozása nélkül már eleve háttérfolyamatként futtatja a parancsot. Keressük ki ezeket a parancsokat:

```
[1] PS C:\> PS C:\> Get-Help * -Parameter asjob

Name                                Category  Synopsis
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
----                                --------  --------
Invoke-Command                      Cmdlet    Runs commands on local and...
Get-WmiObject                       Cmdlet    Gets instances of Windows ...
Invoke-WmiMethod                    Cmdlet    Calls Windows Management I...
Remove-WmiObject                    Cmdlet    Deletes an instance of an ...
Set-WmiInstance                     Cmdlet    Creates or updates an inst...
Test-Connection                     Cmdlet    Sends ICMP echo request pa...
Restart-Computer                    Cmdlet    Restarts ("reboots") the o...
Stop-Computer                       Cmdlet    Stops (shuts down) local a...
```

Például a `Restart-Computer` biztos hosszadalmas folyamat, hiszen le kell állítani a futó folyamatokat, ami nem két pillanat. Vagy hasonlóan a `Test-Connection` is ilyen cmdlet, amellyel gyakorlatilag „pingetni” lehet gépeket:

```
PS C:\Users\TEMP> Test-Connection dc -AsJob

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
5               Job5            Running    False           .


PS C:\Users\TEMP> Receive-Job 5

WARNING: 2 columns do not fit into the display and were removed.

Source        Destination     IPV4Address      IPV6Address
------        -----------     -----------      -----------
MEMBER        dc              192.168.1.10     {}
```

Itt tehát a parancs maga hozta létre a munkamenetet, a lekérdezése ugyanúgy `Receive-Job` cmdlet segítségével történhet.

És ilyen lehet az `invoke-command` cmdlet is, amellyel parancsokat hajtathatunk végre akár saját gépünkön, akár más gépeken is távolról. Ez utóbbi már átvezet minket a következő témánkba, a távoli parancsfuttatás világába.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

1.11 Távoli futtatás

Még nagyobb hiánya volt az 1.0-ás verziónak, hogy nem volt lehetőség PowerShell cmdletek távoli futtatására. Persze bizonyos WMI metódusok rendelkeztek távoli futtatás lehetőségével, amelyeket PowerShellből meghívva persze olyan érzésünk lehetett, hogy táv-futtatunk, de ez nem volt az igazi. Most már viszont (majdnem[12]) bármit futtathatunk távolról is!

A háttérben ennek és a később előkerülő távoli futtatás infrastruktúráját a WinRM szolgáltatás biztosítja. Ez Vista operációs rendszertől fölfelé érhető el alaphelyzetben, de a Management Framework részeként (amelyben a PowerShell 2.0 is benne van) Windows XP-re és Windows Server 2003-ra is telepíthető. Ez a szolgáltatás telepítés után magától nem fut, legegyszerűbben az `Enable-PSRemoting` paranccsal lehet bekapcsolni:

```
[14] PS C:\> Enable-PSRemoting

WinRM Quick Configuration
Running command "Set-WSManQuickConfig" to enable this machine for remote
management through WinRM service.
 This includes:
    1. Starting or restarting (if already started) the WinRM service
    2. Setting the WinRM service type to auto start
    3. Creating a listener to accept requests on any IP address
    4. Enabling firewall exception for WS-Management traffic (for http
only).

Do you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
WinRM already is set up to receive requests on this machine.
WinRM already is set up for remote management on this machine.

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name:
microsoft.powershell SDDL:
O:NSG:BAD:P(A;;GA;;;BA)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This will
allow selected users to remotely run Windows PowerShell commands on this
computer".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name:
Microsoft.PowerShell32 SDDL: O:NSG:BAD:P(A;;GA;;;BA)S:P. This will allow
selected users to remotely run Windows PowerShell commands on this
computer".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Set-PSSessionConfiguration" on Target "Name:
microsoft.ServerManager SDDL:
O:NSG:BAD:P(A;;GA;;;BA)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD). This will
```

[12] Nem teszteltem le az összes cmdletet, de a BITS fájlátvitellel kapcsolatos cmdleteket biztos nem lehet távolról futtatni.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
allow selected users to remotely run Windows PowerShell commands on this
computer".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
```

Ez egy összetett folyamatot indít el. Egyrészt a `winrm quickconfig` parancs PowerShelles megfelelőjét, a `Set-WSMandQuickCOnfig`-ot futtatja, másrészt a távoli csatlakozás pontjait állítja be. Ez egy 64 bites gépen tartalmazza az alaphelyzet szerinti `Microsoft.Powershell` csatlakozási pontot (`PSSessionConfiguration`-t) és a `Microsoft.Powershell32` csatlakozási pontokat. Ezekről majd részletesebben a gyakorlati részben, a *2.14 Távoli futtatási környezet testre szabása* fejezetben lesz szó.

A WinRM szolgáltatás http protokollba ágyazott kommunikációt folytat, így akár tűzfalon keresztül is képes két gép ezzel egymással kommunikálni. A kommunikáció eleve titkosított, így nem feltétlenül szükséges SSL titkosítás használata, de arra is van lehetőség. Ha sikeresen beindítottuk a WinRM szolgáltatást, akkor létrehozhatjuk a távoli futtatáshoz szükséges munkameneteket.

Visszatérve a távoli futtatás engedélyezésére, látható volt, hogy sok jóváhagyás után tudtunk csak eljutni a funkció bekapcsolásáig. Ha ezeket a jóváhagyásokat ki akarjuk hagyni, akkor használjuk a `-force` kapcsolót:

```
[15] PS C:\> Enable-PSRemoting -force
```

A távoli futtatással kapcsolatos cmdletek főneve a `PSSession`:

```
PS C:\> Get-Command -noun pssession

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Enter-PSSession                    Enter-PSSession [-Computer...
Cmdlet          Exit-PSSession                     Exit-PSSession [-Verbose] ...
Cmdlet          Export-PSSession                   Export-PSSession [-Session...
Cmdlet          Get-PSSession                      Get-PSSession [[-ComputerN...
Cmdlet          Import-PSSession                   Import-PSSession [-Session...
Cmdlet          New-PSSession                      New-PSSession [[-ComputerN...
Cmdlet          Remove-PSSession                   Remove-PSSession [-Session...
```

Nyissunk is gyorsan egy ilyen `PSSession`-t a `New-PSSession` cmdlet segítségével, rögtön két gépre:

```
[16] PS C:\> New-PSSession -ComputerName dc, member

 Id Name            ComputerName    State    ConfigurationName     Availab
                                                                     ility
 -- ----            ------------    -----    -----------------     -------
  1 Session1        member          Opened   Microsoft.PowerShell ...able
  2 Session2        dc              Opened   Microsoft.PowerShell ...able
```

Ha egy picit részletesebb listanézetre váltunk, akkor még több információt kapunk:

```
[17] PS C:\> Get-PSSession 1 | fl


ComputerName            : member
ConfigurationName       : Microsoft.PowerShell
InstanceId              : 3c8eafa1-1efe-4f7f-ae2d-f1bab9f749f9
Id                      : 1
Name                    : Session1
```

```
Availability           : Available
ApplicationPrivateData : {PSVersionTable}
Runspace               : System.Management.Automation.RemoteRunspace
State                  : Opened
```

Megjegyzés

Egy géphez egy időben akár több `PSSession`-t is nyithatunk. Vigyázzunk, ezt csak akkor alkalmazzuk, ha tényleg erre van szükség. Például sok gépnek akarjuk újraindítani ugyanazt a szolgáltatást, akkor nem sok értelme van ezt kétszer megcsinálni egy adott gépen.

Bírjuk munkára ezt a fajta munkamenetet is, kíváncsi vagyok az összes bevont gépen futó, „g” betűvel kezdődő szolgáltatásra:

```
[23] PS C:\> Invoke-Command -Session (Get-PSSession) -ScriptBlock {get-ser
vice g*}

Status   Name               DisplayName                            PSComp
                                                                   uterNa
                                                                   me
------   ----               -----------                            ------
Running  gpsvc              Group Policy Client                    member
Running  gpsvc              Group Policy Client                    dc
```

Itt nem kell külön lekérni a futtatás eredményét, hiszen itt most nem „job”-ként hajtattam végre a scriptet, azaz az eredményt rögtön megkaptam. A kimenet elvileg akár lehetne „vegyes” is, azaz a különböző gépekről érkező információk keveredhetnének is, ilyenkor szükség lehet a kimenet `PSComputerName` szerinti rendezésére is.

Ha mégis háttérben szeretném futtatni ezeket, akkor van lehetőség erre is, a korábban már említett `-AsJob` kapcsoló használatával:

```
[33] PS C:\> Invoke-Command -Session (Get-PSSession) -ScriptBlock {get-ser
vice g*} -AsJob

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
23              Job23           Running    True            member,dc


[34] PS C:\> Receive-Job 23

Status   Name               DisplayName                            PSComp
                                                                   uterNa
                                                                   me
------   ----               -----------                            ------
Running  gpsvc              Group Policy Client                    member
Running  gpsvc              Group Policy Client                    dc
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Az első lépésben tehát most nem magát a szolgáltatások listáját kaptam meg, hanem az elindított munkamenet-objektumot. Itt láthatjuk, hogy a `Location` tulajdonság tartalmazza a számítógépneveket, ahol a munkamenet fut. A `Receive-Job`-bal kérdezhetjük le a futtatott szkript kimenetét.

Még érdekesebb, hogy interakcióba is léphetünk ilyen PSSession objektumokkal az `Enter-PSSession` cmdlet segítségével, azaz úgy dolgozhatunk, mintha tényleg azon a gépen nyitottunk volna meg egy PowerShell ablakot:

```
[35] PS C:\> Get-PSSession

 Id Name            ComputerName    State    ConfigurationName    Availab
                                                                   ility
 -- ----            ------------    -----    -----------------    -------
  1 Session1        member          Opened   Microsoft.PowerShell ...able
  2 Session2        dc              Opened   Microsoft.PowerShell ...able


[36] PS C:\> Enter-PSSession 1
[member]: PS C:\Users\TEMP\Documents> cd c:\
[member]: PS C:\> $env:computername
MEMBER
```

A fenti példában az 1-es PSSession-höz csatlakoztam, minek következtében a prompt is megváltozott jelezvén, hogy amit ezután látok és gépelek az nem a saját gépemen, hanem a távoli gépen érvényes. Kilépni ebből az `Exit-PSSession` cmdlettel lehet:

```
[member]: PS C:\> Exit-PSSession
[37] PS C:\>
```

A létrehozott `PSSession` objektumokat a `Remove-PSSession` cmdlettel tudjuk megszüntetni, például az összes megszüntetése a következő kifejezéssel lehetséges:

```
[38] PS C:\> Get-PSSession | Remove-PSSession
```

Ha a `PSSession` objektumokra nincs szükségünk tartósan, mert csak egy szkript futtatásának erejéig akarjuk használni, akkor az `invoke-command` dinamikusan is képes ilyeneket létrehozni, futtatni bennük a szkriptet és megszüntetni. Ilyenkor a `ComputerName` paramétert kell megadni:

```
[39] PS C:\> Invoke-Command -ComputerName dc, member -ScriptBlock {$env:co
mputername}
MEMBER
DC
```

A futtatás után nem is marad semmilyen `PSSession` objektum, amit meg kellene szüntetni.

A paraméterátadás ugyanolyan módon történik, mint ahogy a háttérfolyamatok során láttuk, itt is az `-inputobject` paraméteren keresztül lehet átadni a változókat:

```
[40] PS C:\> $a = " gép"; Invoke-Command -Session (get-pssession) -ScriptB
lock {$env:computername + $input} -InputObject $a
MEMBER gép
DC gép
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

Vigyázat! A PSSession-nek átadott objektumok némi csonkításon mennek keresztül. Az alábbi példában egy mappaobjektumot adok át a PSSesson-nek, és a munkamenetből nézve kérek egy `get-member`-t erre az objektumra:

```
[44] PS C:\> $a = get-item C:\munka
[45] PS C:\> Invoke-Command -Session (Get-PSSession) -ScriptBlock {$input
| gm} -inputobject $a


   TypeName: Deserialized.System.IO.DirectoryInfo

WARNING: column "PSComputerName" does not fit into the display and was re
moved.

Name              MemberType   Definition
----              ----------   ----------
ToString          Method       string ToString(), string ToString(stri...
BaseName          NoteProperty System.String BaseName=munka
Mode              NoteProperty System.String Mode=d----
PSChildName       NoteProperty System.String PSChildName=munka
PSDrive           NoteProperty Deserialized.System.Management.Automati...
PSIsContainer     NoteProperty System.Boolean PSIsContainer=True
PSParentPath      NoteProperty System.String PSParentPath=Microsoft.Po...
…
```

Látható, hogy a `System.IO.DirectoryInfo` objektumomból „`Deserialized`” változat lett, azaz elvesztette az objektum az összes metódusát és csak a tulajdonságai élnek tovább. A háttérben pont az játszódik le, mint amit korábban láttunk az objektumok XML adatblokkokká történő konvertálása során, hiszen a hálózaton keresztül itt is pont XML adatok formájában utazgatnak az objektumok.

A könyv tanfolyami felhasználása

1.12 Összefoglaló: PowerShell programozási stílus

Ez elméleti rész lezárásaként összefoglalnám azokat a főbb jellemzőket, amelyek a PowerShellben történő programozás jellegzetességeinek érzek:

- Csövezés minden mennyiségben!
  Egy-egy művelet eredményét legtöbb esetben felesleges változókba rakni, érdemes azonnal továbbküldeni további feldolgozásra a csővezetéken keresztül egy újabb parancs számára. Ezzel nem csak egyszerűbb, tömörebb lesz a programunk, hanem egy csomó átmeneti adattárolás memória-felhasználását spóroljuk meg.

- Gyűjtemény vagy objektum?
  Mint ahogy a láttuk például a `get-member` cmdlet működésénél, a PowerShell néha túl „okosan” próbál gyűjteményeket kezelni, azaz nem mint objektumokat használja, hanem kifejti az egyes elemeit. Míg ha egy elemet adunk neki, akkor meg azt az adott objektumként kezeli. Ez gyakran félrevezető, főleg amikor egy objektum tulajdonságait próbáljuk feltérképezni, és azt hisszük, hogy már egy indexelhető tömbnél tartunk, és akkor derül ki számunkra, hogy nem, még mélyebbre kell ásnunk, mert amit látunk az még mindig egy egyetlen elemet tartalmazó tömbobjektum.

- Hagyományos „programolós” ciklus helyett bármi más!
  Akinek valamilyen klasszikus programnyelvben van gyakorlata, az PowerShellben is hajlamos eleinte minden többelemű adat feldolgozásakor `FOR` ciklust írni. De ha nincs szükségünk az elemek ilyen jellegű indexelésére, nyugodtan használjunk `FOREACH` ciklust. Ha csővezetéken érkeznek az adatok, akkor meg használjunk `ForEach-Object` cmdletet. Sőt, mint ahogy az „Exchange 12 Rocks” példámban szerepelt (*1.4.8 Típuskonverzió* fejezet), a típuskonverzió is kiválthat sok esetben ciklust, illetve még a `SWITCH` kifejezés is használható ciklusként.

- Minden objektum!
  Soha ne feledkezzünk meg arról, hogy a PowerShellben minden kimenet objektum. Ugyan a konzolon szövegeket, karaktereket látunk, de ezek az esetek túlnyomó többségében nem egyszerű szövegek, hanem összetett objektumok, melyeknek csak néhány tulajdonságát látjuk a képernyőn szövegként. Ráadásul ezek a tulajdonságok is általában szintén objektumok. Ne legyünk restek ezeknek a mélyére ásni. Ebben segít bennünket a `get-member` cmdlet, a különböző szkriptszerkesztők (mint például a PowerGUI Script Editora) és a Reflector segédprogram, valamint az MSDN weboldal. Nézzünk utána a tulajdonságoknak, metódusoknak, konstruktoroknak, statikus metódusoknak, nehogy leprogramozzunk valami olyasmit, ami már készen megtalálható az objektum jellemzői között vagy a .NET keretrendszer valamelyik osztályában.

- Szabjuk testre a PowerShellt, de ez ne menjen a kompatibilitás kárára
  Ugyan a 2.0-ás verzióval a PowerShellnek egyre kevesebb hiányossága van, de ha mégis szeretnénk valamivel kibővíteni, akkor azt nagyon egyszerűen megtehetjük. Újabb függvényekkel új funkciókat valósíthatunk meg. Új álnevekkel kevesebbet kell gépelnünk. A típusok kiegészítésével újabb

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

tulajdonságokat és metódusokat készíthetünk objektumainkhoz. Modulokkal újabb funkció-halmazokat tudunk egy egységként kezelni és hordozni gépek között.
A beépített cmdleteket, álneveket lehetőleg ne definiáljuk újra, mert ez a programjaink, parancssoraink értelmezhetőségének kárára megy.

A könyv tanfolyami felhasználása nem engedélyezett!

2.Gyakorlat

Az elméleti részben áttekintettük a PowerShell telepítését, nyelvi elemeit. A könyv ezen részében gyakorlatiasabb példákkal folytatom. Természetesen itt sem fogok több oldalas példaprogramokat írni, hiszen ez a könyv nem fejlesztőknek, hanem gyakorló rendszergazdáknak szól, akik gyorsan, kevés programozással szeretnének eredményre jutni.

2.1 PowerShell környezet

A gyakorlati rész első fejezete a PowerShell környezet komfortosabbá tételéről szól, milyen lehetőségek állnak rendelkezésre, hogy a PowerShell ablak ne ugyanolyan tudással induljon, mint korábban, hanem építse be az általunk elkészített függvényeket, szkripteket, illetve töltse be azokat a külső bővítményeket, amelyekkel szintén kiterjeszthetjük a képességeit.

2.1.1 Szkriptkönyvtárak, futtatási információk ($myinvocation)

Munkánk során valószínű jó néhány hasznos függvényt készítünk, amelyeket rendszeresen használni szeretnénk. Ezekhez úgy férünk hozzá legegyszerűbben, ha ezeket a függvényeket szkriptfájlokban elmentjük egy könyvtárba, majd a sok kis szkriptfájlunkat egy „beemelő”, „include” jellegű központi szkripttel lefuttatjuk (vagy modult készítünk belőlük, lásd *2.1.7 Modulok* fejezetet.

Ennek modellezésére készítettem egy „scripts” könyvtárat, amelyben három szkriptem három függvényt definiál. Ezen kívül van egy `include.ps1` szkriptem, ami csak annyit csinál, hogy a saját könyvtárában levő másik három szkriptet meghívja „dotsourcing” jelleggel, azaz úgy, hogy a szkriptek által definiált függvények bárhonnan elérhetők, meghívhatók legyenek.

```
[17] PS C:\powershell2\egyik> Get-ChildItem C:\powershell2\scripts


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\powershell2\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.04.19.    12:31           42 fv1.ps1
-a---   2008.04.19.    12:31           45 fv2.ps1
-a---   2008.04.19.    12:31           45 fv3.ps1
-a---   2008.04.19.    12:32           40 include.ps1


[18] PS C:\powershell2\egyik> get-content C:\powershell2\scripts\fv1.ps1
function fv1
{
        "Első függvény"
}
[19] PS C:\powershell2\egyik> get-content C:\powershell2\scripts\include.ps1

. .\fv1.ps1
. .\fv2.ps1
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
. .\fv3.ps1
```

Ez egyes függvények nagyon egyszerűek, csak annyit írnak ki, hogy hányadik függvényről van szó. Ez így külön-külön nagyon szépnek és logikusnak tűnik, próbáljuk meg futtatni az `include.ps1` szkriptünket az „egyik” nevű könyvtárból:

```
[22] PS C:\powershell2\egyik> C:\powershell2\scripts\include.ps1
The term '.\fv1.ps1' is not recognized as the name of a cmdlet, function, scri
pt file, or operable program. Check the spelling of the name, or if a path was
 included, verify that the path is correct and try again.
At C:\_munka\powershell2\scripts\include.ps1:3 char:2
+ . <<<<  .\fv1.ps1
    + CategoryInfo          : ObjectNotFound: (.\fv1.ps1:String) [], CommandN
   otFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

The term '.\fv2.ps1' is not recognized as the name of a cmdlet, function, scri
pt file, or operable program. Check the spelling of the name, or if a path was
 included, verify that the path is correct and try again.
At C:\_munka\powershell2\scripts\include.ps1:4 char:2
+ . <<<<  .\fv2.ps1
    + CategoryInfo          : ObjectNotFound: (.\fv2.ps1:String) [], CommandN
   otFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

The term '.\fv3.ps1' is not recognized as the name of a cmdlet, function, scri
pt file, or operable program. Check the spelling of the name, or if a path was
 included, verify that the path is correct and try again.
At C:\_munka\powershell2\scripts\include.ps1:5 char:2
+ . <<<<  .\fv3.ps1
    + CategoryInfo          : ObjectNotFound: (.\fv3.ps1:String) [], CommandN
   otFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Valami nem jó! Nyomozzunk utána, cseréljük le az `include.ps1` belsejét egy olyan vizsgálatra, amely megmutatja, hogy mit érez a szkript aktuális könyvtárnak. Amíg nem találom meg a hibát, a függvényeket definiáló szkriptek hívását kikommenteztem:

```
#. .\fv1.ps1
#. .\fv2.ps1
#. .\fv3.ps1
Get-Location
```

Ezt futtatva a következőket kapjuk:

```
[23] PS C:\powershell2\egyik> C:\powershell2\scripts\include.ps1

Path
----
C:\powershell2\egyik
```

Kiderült a hiba oka, annak ellenére, hogy az `include.ps1` a `scripts` könyvtárban fut, számára is az aktuális könyvtár az „`egyik`”. Hogyan lehetne azt megoldani, hogy az `include.ps1` számára a saját könyvtára legyen az aktuális?

könyv - folyamatban

Szerencsére van egy `$MyInvocation` nevű automatikus változó, amely a szkriptek számára a futtatásukkal kapcsolatos információkat árulja el. Nézzük is ezt meg, módosítottam az `include.ps1` szkriptemet:

```
#. .\fv1.ps1
#. .\fv2.ps1
#. .\fv3.ps1
$MyInvocation
```

Ezt futtatva kapjuk a következőket:

```
[30] PS C:\powershell2\egyik> C:\powershell2\scripts\include.ps1


MyCommand        : include.ps1
ScriptLineNumber : 1
OffsetInLine     : -2147483648
ScriptName       :
Line             : C:\powershell2\scripts\include.ps1
PositionMessage  :
                   At line:1 char:34
                   + C:\powershell2\scripts\include.ps1 <<<<
InvocationName   : C:\powershell2\scripts\include.ps1
PipelineLength   : 1
PipelinePosition : 1
```

Ha átlépünk a szülőkönyvtárba, és onnan hívjuk meg a szkriptet a következőképpen alakul a `$MyInvocation` értéke:

```
[31] PS C:\powershell2\egyik> cd ..
C:\powershell2
[32] PS C:\powershell2> .\scripts\include.ps1


MyCommand        : include.ps1
ScriptLineNumber : 1
OffsetInLine     : -2147483648
ScriptName       :
Line             : .\scripts\include.ps1
PositionMessage  :
                   At line:1 char:21
                   + .\scripts\include.ps1 <<<<
InvocationName   : .\scripts\include.ps1
PipelineLength   : 1
PipelinePosition : 1
```

Ebből az látszik, hogy se a `Line`, se az `InvocationName` tulajdonság nem a szkript tényleges elérési útját tartalmazza, hanem azt, ahonnan meghívtuk. Így ezekkel a tulajdonságokkal nem tudunk dolgozni, mert ezek sem adnak iránymutatást arra vonatkozólag, hogy hol van ténylegesen az `include.ps1`, és vele együtt a függvényeimet tartalmazó szkriptek.

Nézzük, hogy vajon a `MyCommand`-nak vannak-e tulajdonságai:

```
#. .\fv1.ps1
#. .\fv2.ps1
#. .\fv3.ps1
```

```
$MyInvocation.MyCommand
```

A futtatás eredménye:

```
[34] PS C:\powershell2> .\scripts\include.ps1 | fl


Path        : C:\powershell2\scripts\include.ps1
Definition  : C:\powershell2\scripts\include.ps1
Name        : include.ps1
CommandType : ExternalScript
```

Itt már van egy sokat sejtető `Path` tulajdonság, ami egy teljes elérési út, így remény van rá, hogy ebből kiindulva már jól el fogjuk tudni érni a függvényeket tartalmazó szkripteket.

Létezik egy `split-path` cmdlet, amellyel le tudjuk választani egy elérési útról a könyvtár-hierarchia részt, ennek segítségével már el tudjuk készíteni az univerzális, bárhonnan működő `include.ps1` szkriptünket:

```
Push-Location
Set-Location (split-path $MyInvocation.MyCommand.Path)
. .\fv1.ps1
. .\fv2.ps1
. .\fv3.ps1
Pop-Location
```

És a futtatásának eredménye:

```
[41] PS C:\powershell2> .\scripts\include.ps1
[42] PS C:\powershell2> fv1
The term 'fv1' is not recognized as the name of a cmdlet, function, script fil
e, or operable program. Check the spelling of the name, or if a path was inclu
ded, verify that the path is correct and try again.
At line:1 char:4
+ fv1 <<<<
    + CategoryInfo          : ObjectNotFound: (fv1:String) [], CommandNotFoun
   dException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Na, most mi a hiba? Hát az, hogy bár jól lefutott az `include.ps1`, de a betöltött függvényszkriptek csak az ő szintjére lettek „dotsource”-olva. Ahhoz, hogy a globális scope-ból is elérhessük ezeket a függvényeket, magát az `include.ps1`-et is dotsource-szal kell meghívni ([43]-as sorban a plusz pont és szóköz a prompt után):

```
[43] PS C:\powershell2> . .\scripts\include.ps1
[44] PS C:\powershell2> fv1
Első függvény
[45] PS C:\powershell2> fv2
Második függvény
```

Így már tökéletesen működik a szkriptkönyvtárunk.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.1.1.1 A $MyInvocation felhasználása parancssor-elemzésre

Nézzük kicsit alaposabban meg ezt a `$myinvocation` változót. Ha interaktívan szeretnénk megnézni, akkor ezt kapjuk:

```
[1] PS C:\> "kakukk"; $myinvocation
kakukk


MyCommand        : "kakukk"; $myinvocation
ScriptLineNumber : 0
OffsetInLine     : 0
ScriptName       :
Line             :
PositionMessage  :
InvocationName   :
PipelineLength   : 2
PipelinePosition : 1
```

Látszik, hogy a `MyCommand` property tartalmazza, hogy mi is az éppen futtatott parancs. Mivel ezt ritkán használjuk interaktívan, nézzük meg, hogy egy szkriptből futtatva mit ad. Maga a szkript nagyon egyszerű:

```
$myinvocation | fl
```

És a kimenet:

```
[7] PS C:\> C:\powershell2\scripts\get-mycommand.ps1


MyCommand        : get-mycommand.ps1
ScriptLineNumber : 1
OffsetInLine     : -2147483648
ScriptName       :
Line             : C:\powershell2\scripts\get-mycommand.ps1
PositionMessage  :
                   At line:1 char:40
                   + C:\powershell2\scripts\get-mycommand.ps1 <<<<
InvocationName   : C:\powershell2\scripts\get-mycommand.ps1
PipelineLength   : 1
PipelinePosition : 1
```

Nézzük meg, hogy egy függvényben hogyan alakul ez a változó:

```
[8] PS C:\> function get-myinvocation {$myinvocation | fl *}
[9] PS C:\> "kakukk" | get-myinvocation


MyCommand        : get-myinvocation
ScriptLineNumber : 1
OffsetInLine     : -2147483648
ScriptName       :
Line             : "kakukk" | get-myinvocation
PositionMessage  :
                   At line:1 char:27
                   + "kakukk" | get-myinvocation <<<<
InvocationName   : get-myinvocation
PipelineLength   : 1
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PipelinePosition : 1
```

A fenti példákból látszik, hogy elsősorban a `MyCommand` és a `Line` tulajdonság hasznos. Ha csak a szűken vett futtatási környezetre vagyunk kíváncsi, akkor a `MyCommand` kell nekünk, ha a teljes parancssor, akkor `Line`.

Ezt felhasználva készítsünk egy olyan függvényt, ami megismétli valahányszor az előtte formailag csővezetékként megadott kifejezést ilyen formában:

```
kifejezés | repeat-commandline 5
```

Az ezt megvalósító függvény:

```
[1] PS C:\> function repeat-commandline ([int] $x = 2)
>> {
>>     $s = $myinvocation.line
>>     $last = $s.LastIndexOf("|")
>>     if ($last -lt 0) {throw "Nincs mit ismételni!"}
>>     $cropped = $s.substring(0,$last)
>>     for ($r = 0; $r -lt $x; $r++)
>>     {
>>         invoke-expression $cropped
>>     }
>> }
>>
[2] PS C:\> "többször" | repeat-commandline 5
többször
többször
többször
többször
többször
```

A függvény lényegi része a `$myinvocation` automatikus változó `Line` tulajdonságának felhasználása. Nekünk itt a teljes parancssor kell, így a `Line` tulajdonságot használom. Megkeresem az utolsó csőjelet, (ha nincs ilyen benne, akkor a `LastIndexOf` metódus eredménye -1 lesz, ilyenkor hibát jelzek) és csonkolom odáig a kifejezést, majd egy ciklussal végrehajtom ezt annyiszor, amennyi a függvénynek átadott paraméter.

2.1.2 Automatikus változók

A PowerShell számos változót definiál, amelyek a futtatási környezetre vonatkozó információkkal látnak el bennünket. Nézzük ezeket:

Név	Érték (pl.)	Magyarázat
$	c:\hkh.txt	Az utolsó paraméter vagy kifejezés az utolsó végrehajtott sorból
?	FALSE	Az utolsó kifejezés végrehajtásának sikeressége
^	Get-Item	Az utolsó cmdlet
_		A csőelem tartalma
args	{}	A nem nevesített paraméterekhez rendelt paraméterek
ConfirmPreference	High	Rákérdezési preferencia, High = csak a

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

		"nagyon veszélyes" cmdletekre kérdez rá.
ConsoleFileName		A felhasznált konzolfájl
DebugPreference	SilentlyContinue	Hibakeresési preferencia, a Write-Debug cmdlet kezelése
Error	{Cannot find path 'C:\hkh.txt' because it do...	Az utolsó $MaximumErrorCount hiba
ErrorActionPreference	Continue	Nem megszakító hiba esetén továbblépés
ErrorView	NormalView	Hiba megjelenítésének formája
ExecutionContext	System.Management.Automation.EngineIntrinsics	PowerShell gazdaalkalmazás futtatási környezete
FALSE	FALSE	A "hamis" értéke
FormatEnumerationLimit	4	Gyűjteménytulajdonságok megjelenítésének maximum darabszáma, utána "…"
HOME	C:\Users\Administrator	Felhasználó Home könyvtára
Host	System.Management.Automation.Internal.Host.I...	PowerShellt megjelenítő alkalmazás
input	System.Collections.ArrayList+ArrayListEnumer...	Csővezeték tartalma
MaximumAliasCount	4096	Álnevek maximális száma
MaximumDriveCount	4096	Meghajtók maximális száma
MaximumErrorCount	256	A $Error változóban megőrzött hibák száma
MaximumFunctionCount	4096	Függvények maximális száma
MaximumHistoryCount	64	Parancstörténet hossza
MaximumVariableCount	4096	Változók maximális száma
MyInvocation	System.Management.Automation.InvocationInfo	A parancs futtatási környezetének jellemzői
NestedPromptLevel	0	Aktuális beágyazottsági szint
null		A Null értéket tartalmazó változó
OutputEncoding	System.Text.ASCIIEncoding	A csővezeték elemei
PID	3468	A PowerShell folyamat azonosítója
PROFILE	C:\Users\Administrator\Documents\WindowsPowe...	A PowerShell alkalmazás profilja
ProgressPreference	Continue	A Progressbar megszakítási módja
PSBoundParameters	{}	A függvény meghívásakor megadott paraméterekből és értékeikből képzett hashtábla
PSCulture	hu-HU	Futtatási környezet nyelvi beállítása
PSEmailServer		PowerShell számára megadott SMTP kiszolgáló
PSHOME	C:\Windows\System32\WindowsPowerShell\v1.0	PowerShell telepítési helye
PSSessionApplicationName	wsman	Távoli végrehajtás csatlakozási pontjánál meghatározott alkalmazás
PSSessionConfigurationName	http://schemas.microsoft.com/powershell/Micr...	Távoli végrehajtás alaphelyzet szerinti környezete
PSSessionOption	System.Management.Automation.Remoting.PSSess...	Távoli csatlakozás beállításai
PSUICulture	en-US	A Windows nyelvi verziója

A könyv tanfolyami felhasználása nem engedélyezett!

PSVersionTable	{CLRVersion, BuildVersion, PSVersion, WSManS...	PowerShell által használt technológiák verziószáma
PWD	C:\	Aktuális könyvtár
ReportErrorShowExceptionClass	0	Még nincs implementálva
ReportErrorShowInnerException	0	Még nincs implementálva
ReportErrorShowSource	1	Még nincs implementálva
ReportErrorShowStackTrace	0	Még nincs implementálva
ShellId	Microsoft.PowerShell	PowerShell környezet azonosítója
StackTrace		Legutóbbi hiba nyomkövetési információi
TRUE	TRUE	Az "igaz" értéke
VerbosePreference	SilentlyContinue	Write-Verbose kezelése
WarningPreference	Continue	Write-Warning kezelése
WhatIfPreference	FALSE	Alaphelyzet szerinti -WhatIf hozzáillesztése

Nézzük kicsit részletesebben ezek közül a fontosabbak gyakorlati felhasználását!

2.1.2.1 Paraméterezés vizsgálata ($PSBoundParameters)

Elsőként nézzük a `$PSBoundParameters` változót! Létrehoztam egy nagyon egyszerű függvényt, aminek csak egy paramétere van, és a függvény törzsében csak annyi történik, hogy kiíratom a `$PSBoundParameters` változót. Nézzük, hogyan működik:

```
[14] PS C:\> function vars ($egy) { write-host "PSBound:"; $psboundparameters}
[15] PS C:\> vars
PSBound:
```

A [15]-ös sorban paraméter nélkül hívtam meg a `vars` nevű függvényemet és - nem meglepő módon – mivel a `$PSBoundParameters` nem is kapott semmi értéket, ezért a függvényemnek nincs a címkén kívül igazi kimenete. Ezzel szemben mi történik, ha a paraméternek kifejezetten `$null` értéket akarunk átadni?

```
[16] PS C:\> vars -egy $null
PSBound:

Key                                    Value
---                                    -----
egy
```

Itt már van kimenete a függvényemnek. A `$PSBoundParaméters` segítségével tehát el lehet különíteni a kétféle függvényhívási módot. De ez a változó még további dolgokra is képes. Készítek egy újabb függvényt, ami ennek tagjellemzőit írja ki:

```
[19] PS C:\> function varsmembers ($egy) { write-host "PSBound:"; $psboundparam
eters | Get-Member }
[20] PS C:\> varsmembers 1
PSBound:


   TypeName: System.Collections.Generic.Dictionary`2[[System.String, mscorlib,
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
 Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089],[System.Ob
ject, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e
089]]

Name                MemberType            Definition
----                ----------            ----------
Add                 Method                System.Void Add(string key, System....
Clear               Method                System.Void Clear()
ContainsKey         Method                bool ContainsKey(string key)
ContainsValue       Method                bool ContainsValue(System.Object va...
Equals              Method                bool Equals(System.Object obj)
GetEnumerator       Method                System.Collections.Generic.Dictiona...
GetHashCode         Method                int GetHashCode()
GetObjectData       Method                System.Void GetObjectData(System.Ru...
GetType             Method                type GetType()
OnDeserialization   Method                System.Void OnDeserialization(Syste...
Remove              Method                bool Remove(string key)
ToString            Method                string ToString()
TryGetValue         Method                bool TryGetValue(string key, System...
BoundPositionally   NoteProperty          System.Collections.Generic.List`1[[...
Item                ParameterizedProperty System.Object Item(string key) {get...
Comparer            Property              System.Collections.Generic.IEqualit...
Count               Property              System.Int32 Count {get;}
Keys                Property              System.Collections.Generic.Dictiona...
Values              Property              System.Collections.Generic.Dictiona...
```

Látható, hogy ez nagyjából hashtábla, de mégsem teljesen. Ennek vizsgálatára készítettem egy egyszerű hashtáblát és összehasonlítottam a két dolog tagjellemzőit:

```
[21] PS C:\> $h = @{egy = 1}
[22] PS C:\> $o1 = varsmembers 1
PSBound:
[23] PS C:\> $o2 = $h | gm
[24] PS C:\> Compare-Object $o2 $o1 -Property name

name                                SideIndicator
----                                -------------
TryGetValue                         =>
BoundPositionally                   =>
Comparer                            =>
Clone                               <=
Contains                            <=
CopyTo                              <=
IsFixedSize                         <=
IsReadOnly                          <=
IsSynchronized                      <=
SyncRoot                            <=
```

Látható, hogy ennek a `$PSBoundParameters` hashtábla-szerűségnek van `TryGetValue`, `BoundPositionally` és `Comparer` metódusa, de nincs `Clone`, `Contains`, `CopyTo` metódusa és néhány tulajdonsága. Ezen kívül van számos közös jellemzőjük:

```
[29] PS C:\> Compare-Object $o2 $o1 -Property name -IncludeEqual -ExcludeDiffer
ent

name                                SideIndicator
----                                -------------
Add                                 ==
```

```
Clear                                    ==
ContainsKey                              ==
ContainsValue                            ==
Equals                                   ==
GetEnumerator                            ==
GetHashCode                              ==
GetObjectData                            ==
GetType                                  ==
OnDeserialization                        ==
Remove                                   ==
ToString                                 ==
Item                                     ==
Count                                    ==
Keys                                     ==
Values                                   ==
```

Azaz menet közben hozzá lehet adni a paraméterekhez még egyet az `Add` metódussal, vagy el lehet távolítani közülük (`Remove`, `Clear`), meg lehet vizsgálni, hogy van-e használatban valamelyik paraméter (`ContainsKey`). Ezekkel a lehetőségekkel majd a fejlett függvényeknél fogunk játszani a *2.4.5 Meglevő cmdletek kiegészítése, átalakítása* fejezetben, amikor is meglevő cmdletekhez készítünk olyan „burkoló” függvényeket, amelyekkel további paramétereket tudunk megadni, vagy meglevőket tudunk elrejteni.

2.1.2.2 Preferencia-változók

Az automatikus változók egy jó része a PowerShell működésének néhány alapvető jellemzőjét befolyásolja. Ezeknek a nevében ott van a Preference kifejezés:

```
[43] PS C:\> Get-Variable *preference

Name                           Value
----                           -----
ConfirmPreference              High
DebugPreference                SilentlyContinue
ErrorActionPreference          Continue
ProgressPreference             Continue
VerbosePreference              SilentlyContinue
WarningPreference              Continue
WhatIfPreference               False
```

Nézzük az elsőt kicsit részletesebben. A `$ConfirmPreference` változó lehetséges értékei és hatásuk:

Érték	Működés
None	Automatikusan nem kérdez rá egyetlen cmdlet végrehajtására sem. De a `-Confirm` paraméter használatával egyedileg beállítható ettől eltérő működés.
Low	Azok a cmdletek, melyek bármilyen veszélyességűek (magas, közepes, alacsony) megerősítést kérnek. Egyedileg a `-confirm` paraméterrel kikapcsolható a rákérdezés.
Medium	A közepes és magas kockázatú cmdletek kérnek megerősítést, az alacsonyak lefutnak enélkül. Megint csak a `-confirm` paraméterrel felülbírálható ez a működés.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

High	Ez az alaphelyzet szerinti beállítás. Csak a magas kockázatú cmdletek kérnek megerősítést.

A `$WhatIfPreference` két értéket vehet fel: `$true` vagy `$false`. Ha `$true` értéket adunk, akkor minden „ShouldProcess” típusú provideren végrehajtott változást okozó cmdlet csak számot ad arról, hogy mit csinálna, de tényleges változást nem okoz, csak ha használjuk a `-WhatIf:$false` paramétert.

A többi preferenciaváltozó a szkriptek és a csőfeldolgozás továbbhaladást szabályozza. Látható, hogy alaphelyzetben minden esetben (kivéve a megszakító hibák) a szkriptjeink futása továbbhalad.

2.1.2.3 Lépjünk kapcsolatba a konzolablakkal ($host)

Jó lenne a PowerShell ablakot is minél komfortosabbá tenni. Ennek néhány lehetősége (QuickEdit mode, ablakszélesség, stb.) már a *1.2.5 Gyorsbillentyűk, beállítások* fejezetben szerepelt. Az ott leírtaknál kicsit több is rendelkezésünkre áll, ehhez tudni kell, hogy létezik egy `$host` automatikus változó, ami a PowerShell ablak sok jellemzőjét tartalmazza.

```
[1] PS C:\> $host


Name             : ConsoleHost
Version          : 2.0
InstanceId       : f6cc65f2-c8aa-4c7b-87c9-f1d2e1e6ade8
UI               : System.Management.Automation.Internal.Host.InternalHos
                   tUserInterface
CurrentCulture   : hu-HU
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

Közvetlenül a `$host` még nem igazán a mi barátunk, de nézzük ennek tagjellemzőit:

```
[3] PS C:\> $host | gm


   TypeName: System.Management.Automation.Internal.Host.InternalHost

Name                      MemberType Definition
----                      ---------- ----------
EnterNestedPrompt         Method     System.Void EnterNestedPrompt()
Equals                    Method     bool Equals(System.Object obj)
ExitNestedPrompt          Method     System.Void ExitNestedPrompt()
GetHashCode               Method     int GetHashCode()
GetType                   Method     type GetType()
NotifyBeginApplication    Method     System.Void NotifyBeginApplication()
NotifyEndApplication      Method     System.Void NotifyEndApplication()
PopRunspace               Method     System.Void PopRunspace()
PushRunspace              Method     System.Void PushRunspace(runspace runs...
SetShouldExit             Method     System.Void SetShouldExit(int exitCode)
ToString                  Method     string ToString()
CurrentCulture            Property   System.Globalization.CultureInfo Curre...
CurrentUICulture          Property   System.Globalization.CultureInfo Curre...
InstanceId                Property   System.Guid InstanceId {get;}
IsRunspacePushed          Property   System.Boolean IsRunspacePushed {get;}
```

A könyv tanfolyami felhasználása nem engedélyezett!

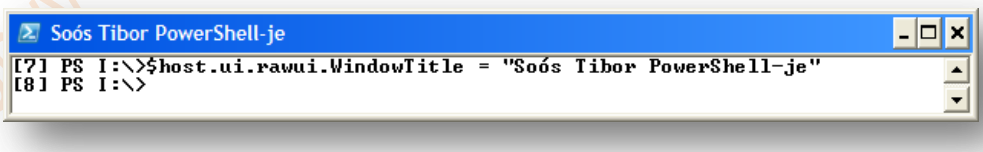
© Soós Tibor – soost@ IQJB.hu

```
Name                    Property   System.String Name {get;}
PrivateData             Property   System.Management.Automation.PSObject ...
Runspace                Property   System.Management.Automation.Runspaces...
UI                      Property   System.Management.Automation.Host.PSHo...
Version                 Property   System.Version Version {get;}
```

Még itt sem biztos, hogy felcsillan a szemünk, de nézzük a sokat sejtető `RawUI`-t:

```
[5] PS C:\> $host.ui.rawui | gm


   TypeName: System.Management.Automation.Internal.Host.InternalHostRawUserI
nterface

Name                   MemberType Definition
----                   ---------- ----------
Equals                 Method     bool Equals(System.Object obj)
FlushInputBuffer       Method     System.Void FlushInputBuffer()
GetBufferContents      Method     System.Management.Automation.Host.Buffer...
GetHashCode            Method     int GetHashCode()
GetType                Method     type GetType()
LengthInBufferCells    Method     int LengthInBufferCells(string str), int...
NewBufferCellArray     Method     System.Management.Automation.Host.Buffer...
ReadKey                Method     System.Management.Automation.Host.KeyInf...
ScrollBufferContents   Method     System.Void ScrollBufferContents(System....
SetBufferContents      Method     System.Void SetBufferContents(System.Man...
ToString               Method     string ToString()
BackgroundColor        Property   System.ConsoleColor BackgroundColor {get...
BufferSize             Property   System.Management.Automation.Host.Size B...
CursorPosition         Property   System.Management.Automation.Host.Coordi...
CursorSize             Property   System.Int32 CursorSize {get;set;}
ForegroundColor        Property   System.ConsoleColor ForegroundColor {get...
KeyAvailable           Property   System.Boolean KeyAvailable {get;}
MaxPhysicalWindowSize  Property   System.Management.Automation.Host.Size M...
MaxWindowSize          Property   System.Management.Automation.Host.Size M...
WindowPosition         Property   System.Management.Automation.Host.Coordi...
WindowSize             Property   System.Management.Automation.Host.Size W...
WindowTitle            Property   System.String WindowTitle {get;set;}
```

Itt már minden van, ami hasznos lehet. Gyakorlatilag az ablak legtöbb tulajdonsága ezen objektumon keresztül lekérdezhető és beállítható. Például cseréljük le az ablak fejlécének szövegét:



42. ábra Megváltoztatott ablak-fejléc

Billentyűleütésre váró programok készíthetők a `ReadKey()` metódus segítségével:

```
[13] PS I:\>$host.ui.rawui.ReadKey()
a

    VirtualKeyCode          Character    ControlKeyState            KeyDown
    --------------          ---------    ---------------            -------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                65                  a           NumLockOn                True
```

Kimeneteként láthatjuk a karakterkódot és a leütött karaktert magát is. Ha nem akarjuk látni a leütött karaktert, akkor használhatjuk a `ReadKey()` különböző opciói közül a `NoEcho`-t, amelyet vagy az `IncludeKeyDown`, vagy az `IncludeKeyDown` opcióval együtt kell használni:

```
[14] PS C:\>$host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")

    VirtualKeyCode          Character    ControlKeyState            KeyDown
    --------------          ---------    ---------------            -------
                74                  j           NumLockOn               True
```

Az `IncludeKeyUp` használatával csak a billentyű felengedésekor ad kimenetet a kifejezés. Ez akkor hasznos, ha a Shift vagy egyéb kiegészítő billentyűt is akarjuk használni, hiszen ha a lenyomásra élesedne, akkor már a Shifthez való hozzáéréskor lefutna a metódus és nem lenne idő az „igazi" billentyű lenyomására. Például egy Shift+Ctrl+Alt+w megnyomása esetén a metódus futásának eredménye így néz ki:

```
[15] PS C:\>$host.UI.RawUI.ReadKey("NoEcho,IncludeKeyUp") | fl *


VirtualKeyCode  : 87
Character       :
ControlKeyState : LeftAltPressed, LeftCtrlPressed, ShiftPressed, NumLockOn
KeyDown         : False
```

Ezzel, és még egy opció, az „AllowCtrlC" megadásával akár a Ctrl+C is megfigyeltethető:

```
[16] PS C:\>$host.UI.RawUI.ReadKey("NoEcho,AllowCtrlC,IncludeKeyUp")

    VirtualKeyCode          Character    ControlKeyState            KeyDown
    --------------          ---------    ---------------            -------
                67                  ♥    LeftCtrlPressed              False
```

2.1.3 Környezeti változók (env:)

A DOS/Windows-os, és néhány PowerShellel kapcsolatos környezeti változót is elérhetünk az `env`: PSDrive-on keresztül:

```
[1] PS C:\> cd env:
[2] PS Env:\> dir

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\Administrator\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   DC
ComSpec                        C:\Windows\system32\cmd.exe
FP NO HOST CHECK               NO
HOMEDRIVE                      C:
HOMEPATH                       \Users\Administrator
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
LOCALAPPDATA                    C:\Users\Administrator\AppData\Local
LOGONSERVER                     \\DC
NUMBER_OF_PROCESSORS            1
OS                              Windows_NT
Path                            %SystemRoot%\system32\WindowsPowerShell...
PATHEXT                         .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;...
PROCESSOR_ARCHITECTURE          AMD64
PROCESSOR_IDENTIFIER            Intel64 Family 6 Model 15 Stepping 10, ...
PROCESSOR_LEVEL                 6
PROCESSOR_REVISION              0f0a
ProgramData                     C:\ProgramData
ProgramFiles                    C:\Program Files
ProgramFiles(x86)               C:\Program Files (x86)
ProgramW6432                    C:\Program Files
PSModulePath                    C:\Users\Administrator\Documents\Window...
PUBLIC                          C:\Users\Public
SESSIONNAME                     Console
SystemDrive                     C:
SystemRoot                      C:\Windows
TEMP                            C:\Users\ADMINI~1\AppData\Local\Temp\2
TMP                             C:\Users\ADMINI~1\AppData\Local\Temp\2
USERDNSDOMAIN                   R2.DOM
USERDOMAIN                      R2
USERNAME                        Administrator
USERPROFILE                     C:\Users\Administrator
windir                          C:\Windows
```

Ezeket a változókat felhasználhatjuk szkriptjeinkben. Például keressük az összes `SystemRoot`-ban található *„log”* kiterjesztésű fájlt:

```
[6] PS C:\> $env:systemroot -split ";" | foreach-object {Get-ChildItem -Path "$
_\*" -Include *.log}


    Directory: C:\Windows


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---      2009. 11. 09.     21:14       1774 DtcInstall.log
-a---      2010. 02. 21.     13:01      13484 PFRO.log
-a---      2009. 11. 09.     21:14      14312 setupact.log
-a---      2009. 07. 14.      6:56          0 setuperr.log
-a---      2009. 11. 09.     21:14       1313 TSSysprep.log
-a---      2010. 03. 04.     10:47    1786861 WindowsUpdate.log
```

Vagy például ki szeretném törölni az összes tmp kiterjesztésű átmeneti állományt:

```
[47] PS C:\> dir ($($env:temp)+"\*") -Include *.tmp | Remove-Item
```

Itt előfordulhat, hogy ha vannak zárolt állományaink, akkor törlésük nem sikerül, hibát kapunk, de a többi állományt törli a kifejezés.

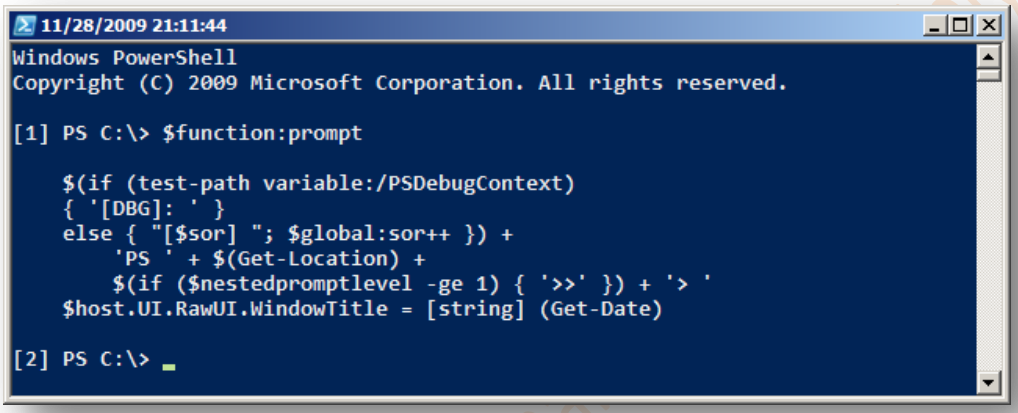
A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.1.4 Prompt beállítása

A PowerShellben a prompt testre szabható. Mint ahogy a *1.7.11 Gyári függvények* fejezetben láttuk, a promptot egy automatikusan, minden beviteli sor megnyílásakor meghívódó `prompt` nevű függvény generálja. Ez a függvény átdefiniálható, testre szabható. Én ebben a könyvben a normál „`[PS] C:\>`” jellegű prompt elé biggyesztettem egy folytonosan növekvő sorszámot, hogy jobban lehessen hivatkozni a bemásolt parancssorokra. De természetesen a `prompt` függvénnyel nem csak olyan tevékenységeket végeztethetünk, ami kizárólag a megjelenő promptot szabja testre, hanem például a PowerShell ablakunk fejlécét is folyamatosan aktualizálhatjuk az aktuális időre.

Nézzünk erre egy példát:



43. ábra A prompt testre szabása és annak eredménye

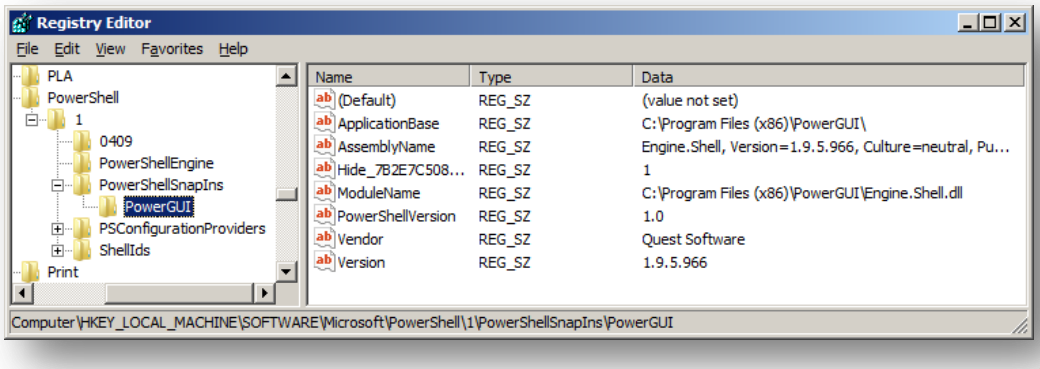
Látjuk, hogy amint a függvény definíciója megtörtént, már életbe is lép az új prompt. Azonban vigyázzunk, ha becsukjuk az ablakot és újra megnyitjuk, akkor visszakapjuk az eredeti promptot, a mi testre szabásunkat elfelejti. Ezen majd a nemsokára bemutatásra kerülő profilok segítenek majd.

2.1.5 Snapin-ek

A PowerShell moduláris felépítésű. A PowerShell 1.0 verzióban az egyes cmdletek és providerek különböző snapin-eknek voltak köszönhetők. A snapinek általában dll fájlok, és a registry-ben kell őket regisztrálni a PowerShell számára. Egy ilyen snapin regisztrációjának lenyomata valahogy így néz ki:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



44. ábra Snapin regisztrációja

Ezt természetesen nem nekünk, manuálisan kell bejegyezni, hanem a snapin gyártója általában biztosít valamilyen telepítőt, ami ezt megteszi. Ha van ilyen szépen előkészített snapinünk, akkor – ha a telepítője valamilyen egyéb módon nem integrálja be a PowerShell konzolba – a `Add-PSSnapin` cmdlettel be tudjuk építeni a PowerShell konzolba.

Snapinekből már eleve több van a PowerShellben, és ezeket a „gyári” snapineket nem kell külön regisztrálni. A snapinek lekérdezhetők a `get-pssnapin` cmdlettel:

```
[2] PS C:\> Get-PSSnapin


Name        : Microsoft.PowerShell.Diagnostics
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains Windows Eventing and
              Performance Counter cmdlets.

Name        : Microsoft.WSMan.Management
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains cmdlets (such as Get-
              WSManInstance and Set-WSManInstance) that are used by the Wind
              ows PowerShell host to manage WSMan operations.

Name        : Microsoft.PowerShell.Core
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains cmdlets used to manag
              e components of Windows PowerShell.

Name        : Microsoft.PowerShell.Utility
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains utility Cmdlets used
              to manipulate data.

Name        : Microsoft.PowerShell.Host
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains cmdlets (such as Star
              t-Transcript and Stop-Transcript) that are provided for use wi
              th the Windows PowerShell console host.

Name        : Microsoft.PowerShell.Management
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains management cmdlets us
              ed to manage Windows components.
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name        : Microsoft.PowerShell.Security
PSVersion   : 2.0
Description : This Windows PowerShell snap-in contains cmdlets to manage Win
              dows PowerShell security.
```

Snap-inekkel kapcsolatos egyéb cmdletek:

```
[3] PS I:\>Get-Command -noun pssnapin

CommandType     Name                              Definition
-----------     ----                              ----------
Cmdlet          Add-PSSnapin                      Add-PSSnapin [-Name] <Stri...
Cmdlet          Get-PSSnapin                      Get-PSSnapin [[-Name] <Str...
Cmdlet          Remove-PSSnapin                   Remove-PSSnapin [-Name] <S...
```

Azaz hozzá lehet adni (`Add-PSSnapin`) ezeket, le lehet kérdezni (`Get-PSSnapin`), és el lehet távolítani (`Remove-PSSnapin`).

Még azt is megnézhetjük, hogy az egyes cmdletek melyik snapinben tanyáznak:

```
[17] PS C:\> get-command -commandtype cmdlet | Group-Object pssnapin | ft -wr
ap

Count Name                      Group
----- ----                      -----
   79 Microsoft.PowerShell.Mana {Add-Computer, Add-Content, Checkpoint-Compu
      gement                    ter, Clear-Content...}
   41 Microsoft.PowerShell.Core {Add-History, Add-PSSnapin, Clear-History, D
                                isable-PSSessionConfiguration...}
   87 Microsoft.PowerShell.Util {Add-Member, Add-Type, Clear-Variable, Compa
      ity                       re-Object...}
   13 Microsoft.WSMan.Managemen {Connect-WSMan, Disable-WSManCredSSP, Discon
      t                         nect-WSMan, Enable-WSManCredSSP...}
   10 Microsoft.PowerShell.Secu {ConvertFrom-SecureString, ConvertTo-SecureS
      rity                      tring, Get-Acl, Get-AuthenticodeSignature...
                                }
    4 Microsoft.PowerShell.Diag {Export-Counter, Get-Counter, Get-WinEvent,
      nostics                   Import-Counter}
    2 Microsoft.PowerShell.Host {Start-Transcript, Stop-Transcript}
```

Mikor lehet erre szükség? Az alap snap-ineket nem lehet eltávolítani, de találkozhatunk olyan kiegészítésekkel különböző szoftverek által, amelyeket esetleg ki lehet kapcsolni, vagy éppen az alap PowerShell alkalmazáshoz hozzá lehet adni.

Snap-inekkel kapcsolatos változtatások sem időt állók, azaz ha becsukjuk a PowerShell ablakot és újra megnyitjuk, akkor megint az alaphelyzet szerinti snap-inek köszönnek vissza. Ezek automatikus visszatöltéséhez majd a PS konzolfájlok segítenek (ld. *2.1.6 Konzolfájl*)

Az „alap” PSSnapineken kívüli egyéb bővítményeket a következő módon tudjuk kilistázni:

```
PS C:\> Get-PSSnapin -Registered


Name        : PowerGUI
PSVersion   : 1.0
Description :
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.1.6 Konzolfájl

Mind a szkriptkönyvtáraknál, mind a promptnál, mind pedig a snapineknél megjegyeztem, hogy a saját bővítményeinket, testre szabásainkat a PowerShell ablak elfelejti, ha becsukjuk. Ezért vannak olyan megoldások, amelyek a bővítményeink definiálását, beemelését végző szkriptjeinket automatikusan minden PowerShell ablak nyitáskor lefuttatják.

Az egyik ilyen lehetőséggel magát a konzolt tudjuk testre szabni a snapinek tekintetében függetlenül attól, hogy ki indítja el a PowerShell környezetet. Ezt konzolfájl segítségével tudjuk elérni, ilyen fájlt az `Export-Console` cmdlet segítségével lehet legegyszerűbben létrehozni. Nézzük, hogy most milyen snapinek vannak a gépemen a Start menüből elindított PowerShell ikon után megjelenő konzolban:

```
PS C:\Users\tibi> Get-PSSnapin | Format-Table name

Name
----
Microsoft.PowerShell.Diagnostics
Microsoft.WSMan.Management
Microsoft.PowerShell.Core
Microsoft.PowerShell.Utility
Microsoft.PowerShell.Host
Microsoft.PowerShell.Management
Microsoft.PowerShell.Security
Pscx
```

Látszik, hogy én már telepítettem egy bővítményt: PowerShell Community Extentions (`Pscx`), ami egy PowerShell közösség által fejlesztett bővítménycsomag.

Nézzük ezek után, mit eredményez az `export-console` cmdlet:

```
PS C:\Users\tibi> Export-Console -Path .\console
PS C:\Users\tibi> Get-Content .\console.psc1
<?xml version="1.0" encoding="utf-8"?>
<PSConsoleFile ConsoleSchemaVersion="1.0">
  <PSVersion>2.0</PSVersion>
  <PSSnapIns>
    <PSSnapIn Name="Pscx" />
  </PSSnapIns>
</PSConsoleFile>
```

Ez a cmdlet egy `psc1` kiterjesztésű XML fájlt generál, aminek a mélyén ott található azon snapineknek a listája, amelyek nem „gyáriak”. Ha egy ilyen `psc1` fájlra duplán kattintunk, vagy a powershell.exe paramétereként szerepeltetjük, akkor az itt felsorolt snapineket automatikusan betölti a konzol. Nézzük, hogy hogyan nézne ki ezzel a powershell.exe felparaméterezése:

```
powershell.exe -PSConsoleFile .\console.psc1
```

2.1.7 Modulok

A PowerShell 2.0-ban egyszerűsítették a bővíthetőséget. Most már nem a snap-inek az elsődleges építőkövek, hanem a modulok. Egy-egy modul a PowerShell különböző objektumainak - cmdleteknek, providereknek, álneveknek, függvényeknek, súgó témáknak, formátumoknak, típusoknak és változóknak a

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

gyűjteménye, melyek a fájlrendszer egy könyvtárában helyezkednek el. Nem kell telepíteni, nem kell regisztrálni ezeket a registry-ben. Egyszerűen hordozhatók és életre kelthetők.

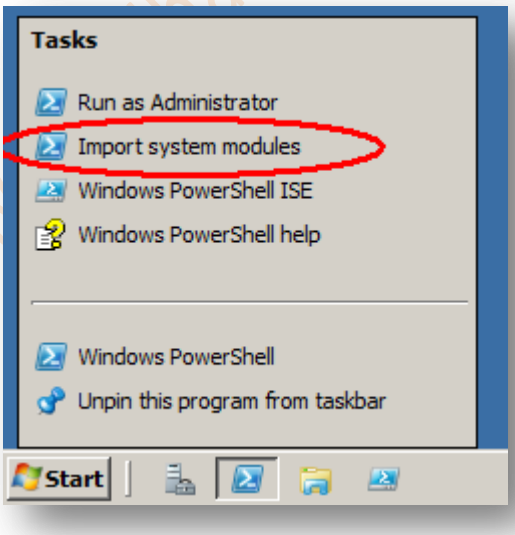
2.1.7.1 Modulok importálása és eltávolítása

A modulokról tehát egyelőre annyit tudunk, hogy egy-egy könyvtár tartalma alkot egy modult, de vajon hol vannak ezek? Egy modulkönyvtár állhat egy adott felhasználó rendelkezésére, ilyenkor a `$home\Documents\WindowsPowerShell\Modules` (Windows XP esetében: `$home\My Documents\WindowsPowerShell\Modules`) hely alkönyvtáraként célszerű létrehozni vagy oda másolni. Ha minden felhasználó részére szeretnénk elérhetővé tenni a modult, akkor a `$PSHOME\modules` könyvtár alkönyvtáraként érdemes létrehozni. Ez utóbbi helyen található modulokat hívjuk rendszermoduloknak. Ezen két elérési úton található modulokat a PowerShell a teljes elérési út kiírása nélkül is megtalálja. Természetesen ettől eltérő helyre is lehet tenni a modulokat, de ilyenkor a teljes elérési út megadásával kell rájuk hivatkozni, vagy a `$PSModulePath` környezeti változó tartalmát kell módosítani:

```
[12] PS C:\> $env:PSModulePath
C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Windows\system
32\WindowsPowerShell\v1.0\Modules\
[13] PS C:\> $env:PSModulePath += ";c:\sajátmodulok"
```

A [12]-es sorban lekérdeztem ezt a változót, a [13]-as sorban kibővítette egy egyedi elérési úttal.

A modulkönyvtárak létrehozása még önmagában nem teszi elérhetővé a modul elemeit, ezután be kell importálni a modult. A rendszermodulokat a tálcán található PowerShell ikon jobb egérgomb-nyomás után megjelenő menüvel egy mozdulattal is be tudjuk importálni:



45. ábra Az összes rendszermodul importálása

A modulokat természetesen egyesével is lehet importálni. Elsőként kérdezzük le a rendelkezésünkre álló modulokat a `get-module` cmdlet segítségével:

```
[1] PS C:\> Get-Module -ListAvailable
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Manifest   ActiveDirectory           {}
Manifest   ADRMS                     {}
Manifest   AppLocker                 {}
Manifest   BestPractices             {}
Manifest   BitsTransfer              {}
Manifest   GroupPolicy               {}
Manifest   PSDiagnostics             {}
Manifest   ServerManager             {}
Manifest   TroubleshootingPack       {}
```

Itt a `ListAvailable` kapcsolót kellett használni, hiszen enélkül csak a már beimportált modulokat kapjuk meg. Az előző paranccsal csak a már korábban említett elérési utakon található modulokat találja meg a `get-module`. Ezek közül beimportálni a modulokat az `import-module` cmdlettel lehet:

```
[2] PS C:\> Import-Module psdiagnostics
```

Ez nem sok kimenetet ad. Azt kideríteni, hogy mit is hoztunk be ezzel a rendszerünkbe a `get-module` cmdlettel lehet:

```
[4] PS C:\> Get-Module psdiagnostics

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Script     psdiagnostics             {Enable-PSTrace, Enable-WSManTrace, ...
```

Hogy jobban látható legyen a modul tartalma, nézzük kicsit részletesebb, olvashatóbb formában:

```
[5] PS C:\> Get-Module psdiagnostics | fl *


ExportedCommands    : {Enable-PSTrace, Enable-WSManTrace, Start-Trace, Disab
                      le-PSWSManCombinedTrace...}
Name                : psdiagnostics
Path                : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\psd
                      iagnostics\PSDiagnostics.psm1
Description         :
Guid                : c61d6278-02a3-4618-ae37-a524d40a7f44
ModuleBase          : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\psd
                      iagnostics
PrivateData         :
Version             : 1.0.0.0
ModuleType          : Script
AccessMode          : ReadWrite
ExportedFunctions   : {[Disable-PSTrace, Disable-PSTrace], [Disable-PSWSManC
                      ombinedTrace, Disable-PSWSManCombinedTrace], [Disable-
                      WSManTrace, Disable-WSManTrace], [Enable-PSTrace, Enab
                      le-PSTrace]...}
ExportedCmdlets     : {}
NestedModules       : {}
RequiredModules     : {}
ExportedVariables   : {}
ExportedAliases     : {}
SessionState        : System.Management.Automation.SessionState
OnRemove            :
ExportedFormatFiles : {}
```

```
ExportedTypeFiles   : {}
```

Egy modulból számunkra legfontosabbak az elérhető parancsok, ezek listáját a következő módon érjük el:

```
[6] PS C:\> (Get-Module psdiagnostics).exportedcommands

Name                            Value
----                            -----
Enable-PSTrace                  Enable-PSTrace
Enable-WSManTrace               Enable-WSManTrace
Start-Trace                     Start-Trace
Disable-PSWSManCombinedTrace    Disable-PSWSManCombinedTrace
Disable-PSTrace                 Disable-PSTrace
Disable-WSManTrace              Disable-WSManTrace
Get-LogProperties               Get-LogProperties
Stop-Trace                      Stop-Trace
Enable-PSWSManCombinedTrace     Enable-PSWSManCombinedTrace
Set-LogProperties               Set-LogProperties
```

Ha egy parancs részleteire vagyunk kíváncsiak, akkor azt a következő kifejezéssel érhetjük el:

```
[7] PS C:\> (Get-Module psdiagnostics).exportedcommands."Enable-PSTrace" | fl
 *


HelpUri             :
ScriptBlock         :
                          $Properties = Get-LogProperties ($script:psprovide
                      rname + $script:analyticlog)
                          $Properties.Enabled = $true
                          Set-LogProperties $Properties

CmdletBinding       : False
DefaultParameterSet :
Definition          :
                          $Properties = Get-LogProperties ($script:psprovide
                      rname + $script:analyticlog)
                          $Properties.Enabled = $true
                          Set-LogProperties $Properties

Options             : None
Description         :
OutputType          : {}
Name                : Enable-PSTrace
CommandType         : Function
Visibility          : Public
ModuleName          : psdiagnostics
Module              : psdiagnostics
Parameters          : {}
ParameterSets       : {}
```

Látható, hogy egy-egy parancs (jelen esetben egy függvény) definíciója egy hashtáblában található, és mivel a parancsok, függvények neveiben általában van kötőjel, ezért idézőjellel kell hivatkozni a hashtábla „kulcsára”.

Modult eltávolítani a `remove-module` cmdlettel lehet:

```
[17] PS C:\> Remove-Module psdiagnostics
```

2.1.7.2 Szkriptmodulok

Eddig a „gyári”, mások által létrehozott modulokat láttuk. De – ellentétben a snapinekkel – modult mi is könnyen tudunk készíteni.

A PowerShell 2.0-ban többfajta modult lehet létrehozni, importálni. Ezek közül számunkra, rendszergazdáknak a legfontosabbak a szkriptmodulok. A szkriptmodul egy `psm1` kiterjesztésű fájl, ami PowerShell szkriptet tartalmaz. Ezt tudjuk legegszerűbben létrehozni, hiszen akár egy már meglevő, bevált szkriptünket egyszerű fájlátnevezéssel modullá tehetjük. Mivel tud többet egy szkriptmodul, mint a szkriptfájl? Elsősorban a szkriptben található függvények, változók és egyéb elemek láthatóságát, hozzáférhetőségét tudjuk kényelmesebben szabályozni az `export-modulemember` cmdlet segítségével. A másik előny, hogy nem kell „dotsourcing” segítségével átemelni az szkript függvényeit és egyéb elemeit. Nézzünk erre egy példát! A következőkben létrehozok egy szkriptmodult, ami egy olyan függvényt definiál (`Read-Popup`), amivel egy felugró ablakot lehet kirakni a képernyőre:

```
$ButtonTypes = @{
    OK = 0;
    OKCancel = 1;
    AbortRetryIgnore = 2;
    YesNoCancel = 3;
    YesNo = 4;
    RetryCancel = 5
}

$IconTypes = @{
    Stop = 16;
    Question = 32;
    Exclamation = 48;
    Information = 64
}

$ReturnButtons = @{
    1 = "OK";
    2 = "Cancel";
    3 = "Abort";
    4 = "Retry";
    5 = "Ignore";
    6 = "Yes";
    7 = "No"
    -1 = "TimeOut"
}

function Read-Popup
{
    param (
        [string] $str = "",
        [int] $wait = 0,
        [string] $title = "Message",
        [string] $buttontype = "OK",
        [string] $icontype = "Information"
    )
    $com = New-Object -ComObject WScript.Shell

    $button = if(!$buttontypes.$buttontype){0} else {$buttontypes.$buttontype}
```

```
    $icon = if(!$icontypes.$icontype){64} else {$icontypes.$icontype}

    $ret = $com.popup($str,$wait,$title,$button+$icon)
    return $returnbuttons.$ret
}
New-Alias -Name popup -Value Read-Popup
Export-ModuleMember -Function Read-Popup -Variable ReturnButtons -Alias popup
```

Ezt a fájlt elmentettem `popup.psm1` néven egy `Popup` nevű könyvtárba, amit a `C:\Windows\System32\WindowsPowerShell\v1.0\Modules\` helyre tettem.

Megjegyzés

Fontos, hogy a modulfájl neve és a könyvtár neve egyforma legyen. Ha különböző, akkor nem lehet importálni.

A szkriptfájl elején definiáltam néhány hashtábla változót, hogy a függvényemet könnyebben lehessen paraméterezni. Ezután definiáltam magát a függvényt, majd létrehoztam egy becenevet a függvényemhez, majd az `Export-ModuleMember` cmdlet segítségével meghatároztam, hogy ebből a szkriptmodulból mit akarok láttatni a külvilággal. Például nem akarom láttatni a `$ButtonTypes` és a `$IconTypes` változóimat, mert erre nem valószínű, hogy az én függvényemen kívül bárkinek is szüksége lehet. De a többi elemet, magát a függvényt, a `$ReturnButtons` változót és a becenevet exportáltam, mert ezek fontosak a későbbi felhasználást tekintve.

Nézzük, hogy hogyan lehet felhasználni ezt a modult:

```
[1] PS C:\> Get-Module -ListAvailable

ModuleType Name                        ExportedCommands
---------- ----                        ----------------
Manifest   ActiveDirectory             {}
Manifest   ADRMS                       {}
Manifest   AppLocker                   {}
Manifest   BestPractices               {}
Manifest   BitsTransfer                {}
Manifest   GroupPolicy                 {}
Script     PopUp                       {}
Manifest   psdiagnostics               {Enable-PSTrace, Enable-WSManTrace, ...
Manifest   ServerManager               {}
Manifest   TroubleshootingPack         {}
```

Látható, hogy megtalálta a `PopUp` modulomat a `get-module` cmdlet. Importálom a modult, és nézzük meg, hogy mit hozott be a PowerShell környezetbe:

```
[2] PS C:\> Import-Module popup
[3] PS C:\> Get-Module popup | fl


Name              : popup
Path              : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\popup
                    \popup.psm1
Description       :
ModuleType        : Script
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Version           : 0.0
NestedModules     : {}
ExportedFunctions : Read-Popup
ExportedCmdlets   : {}
ExportedVariables : ReturnButtons
ExportedAliases   : popup
```

A `Get-Module` megadta, hogy pont azok az elemek érhetők el, amit engedélyeztem az `export-modulemember` cmdlet segítségével.

Amúgy az `export-modulemember` cmdlet elhagyható a modulból, csakhogy a PowerShell ilyenkor automatikusan exportálja az összes függvényt, viszont nem exportál egy változó és becenevet sem. Nézzük ezt meg a következő példában, itt kivettem a szkriptmodul fájlból az utolsó sort:

```
[4] PS C:\> Remove-Module popup
[5] PS C:\> Import-Module popup
[6] PS C:\> Get-Module popup | fl


Name              : popup
Path              : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\popup
                    \popup.psm1
Description       :
ModuleType        : Script
Version           : 0.0
NestedModules     : {}
ExportedFunctions : Read-Popup
ExportedCmdlets   : {}
ExportedVariables : {}
ExportedAliases   : {}
```

A [4]-es sorban eltávolítottam a korábban importált `popup` modult, majd a módosított fájlt újra importáltam az [5]-ös sorban. Ha megnézzük így a modul jellemzőit a [6]-os sorban, akkor látható, hogy csak a függvényem érhető el.

2.1.7.3 Moduljegyzék készítése (Module Manifest)

Korábban láthattuk a `get-module` futtatása során, hogy egy modulnak lehetne verziója, leírása és sok egyéb metaadata, de vajon hogyan lehet ezeket megadni? Erre egy újabb fájltípus, a moduljegyzék, vagy eredeti nevén *module manifest* szolgál. Ez egy szövegfájl, amiben egy hashtábla formátumban vannak megadva a modulhoz tartozó leíró adatok. Hogyan lehet ilyen fájlt létrehozni? Erre a `New-ModuleManifest` cmdletet használhatjuk:

```
[16] PS C:\> New-ModuleManifest -Path $pshome\modules\popup\popup.psd1 -Autho
r "Soós Tibor" -ModuleToProcess popup -ModuleVersion 1.0.0.0 -Description "Gr
afikus Popup ablak"

cmdlet New-ModuleManifest at command pipeline position 1
Supply values for the following parameters:
NestedModules[0]:
CompanyName:
Copyright:
TypesToProcess[0]:
FormatsToProcess[0]:
RequiredAssemblies[0]:
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
FileList[0]:
[17] PS C:\>
[18] PS C:\> Remove-Module popup
[19] PS C:\> Import-Module popup
[20] PS C:\> Get-Module popup | fl


Name              : popup
Path              : C:\Windows\system32\WindowsPowerShell\v1.0\Modules\popup
                    \popup.psm1
Description       : Grafikus Popup ablak
ModuleType        : Script
Version           : 1.0.0.0
NestedModules     : {}
ExportedFunctions : Read-Popup
ExportedCmdlets   : {}
ExportedVariables : ReturnButtons
ExportedAliases   : popup
```

A moduljegyzék létrehozása során én csak a legfontosabb adatokat adtam meg a [16]-os sorban, de a PowerShell szíve szerint minden további adatokat is bekérne, amelyekre rá is kérdez, de nem szükséges ezekre megadni a választ. Ezután eltávolítottam a modul korábban jegyzék nélkül importált változatát [18], majd az újat beimportáltam [19], majd látható, hogy az így lekérdezett modulinformációk [20] már tartalmazzák a jegyzékben meghatározott adatokat is.

Nézzük meg magát a jegyzékfájlt:

```
[23] PS C:\> Get-Content $pshome\modules\popup\popup.psd1
#
# Module manifest for module 'popup'
#
# Generated by: Soós Tibor
#
# Generated on: 2009. 12. 06.
#

@{

# Script module or binary module file associated with this manifest
ModuleToProcess = 'popup'

# Version number of this module.
ModuleVersion = '1.0.0.0'

# ID used to uniquely identify this module
GUID = '200a1d9f-ae8f-4bca-8e70-cd44b23dc294'

# Author of this module
Author = 'Soós Tibor'

# Company or vendor of this module
CompanyName = 'Unknown'

# Copyright statement for this module
Copyright = '(c) 2009 Soós Tibor. All rights reserved.'

# Description of the functionality provided by this module
Description = 'Grafikus Popup ablak'
```

```
# Minimum version of the Windows PowerShell engine required by this module
PowerShellVersion = ''

# Name of the Windows PowerShell host required by this module
PowerShellHostName = ''

# Minimum version of the Windows PowerShell host required by this module
PowerShellHostVersion = ''

# Minimum version of the .NET Framework required by this module
DotNetFrameworkVersion = ''

# Minimum version of the common language runtime (CLR) required by this modu
le
CLRVersion = ''

# Processor architecture (None, X86, Amd64, IA64) required by this module
ProcessorArchitecture = ''

# Modules that must be imported into the global environment prior to importi
ng this module
RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to impo
rting this module
ScriptsToProcess = @()

# Type files (.ps1xml) to be loaded when importing this module
TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
FormatsToProcess = @()

# Modules to import as nested modules of the module specified in ModuleToPro
cess
NestedModules = @()

# Functions to export from this module
FunctionsToExport = '*'

# Cmdlets to export from this module
CmdletsToExport = '*'

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module
AliasesToExport = '*'

# List of all modules packaged with this module
ModuleList = @()

# List of all files packaged with this module
FileList = @()

# Private data to pass to the module specified in ModuleToProcess
PrivateData = ''
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
}
```

Ebben látható, hogy milyen adatok kitöltése lehetséges, ráadásul minden adatmező kis magyarázattal is el van látva.

2.1.7.4 Jegyzékmodul (Manifest Module)

Lehetőség van olyan modul létrehozására is, amiben nincsen modulfájl, csak jegyzékfájl és esetleg .NET építőelemek.

Az előző `Popup` példához készítettem egy Popup2 alkönyvtárba egy jegyzékfájlt:

```
#
# Module manifest for module 'popup2'
#
# Generated by: Soós Tibor
#
# Generated on: 2009. 12. 06.
#

@{

# Script module or binary module file associated with this manifest
ModuleToProcess = ''

# Version number of this module.
ModuleVersion = '1.0'

# ID used to uniquely identify this module
GUID = '3f235c27-503d-46a5-99b9-58ae300eafea'

# Author of this module
Author = 'Soós Tibor'

# Company or vendor of this module
CompanyName = 'IQSOFT - John Bryce Oktatóközpont'

# Copyright statement for this module
Copyright = '(c) 2009 Soós Tibor. All rights reserved.'

# Description of the functionality provided by this module
Description = 'Teszt ModuleManifest'

# Minimum version of the Windows PowerShell engine required by this module
PowerShellVersion = ''

# Name of the Windows PowerShell host required by this module
PowerShellHostName = ''

# Minimum version of the Windows PowerShell host required by this module
PowerShellHostVersion = ''

# Minimum version of the .NET Framework required by this module
DotNetFrameworkVersion = ''

# Minimum version of the common language runtime (CLR) required by this module
CLRVersion = ''

# Processor architecture (None, X86, Amd64, IA64) required by this module
```

```
ProcessorArchitecture = ''

# Modules that must be imported into the global environment prior to importing
this module
RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to
importing this module
ScriptsToProcess = @()

# Type files (.ps1xml) to be loaded when importing this module
TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
FormatsToProcess = @()

# Modules to import as nested modules of the module specified in
ModuleToProcess
NestedModules = 'popup'

# Functions to export from this module
FunctionsToExport = '*'

# Cmdlets to export from this module
CmdletsToExport = '*'

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module
AliasesToExport = '*'

# List of all modules packaged with this module
ModuleList = @()

# List of all files packaged with this module
FileList = @()

# Private data to pass to the module specified in ModuleToProcess
PrivateData = ''

}
```

A lényeges rész itt a `NestedModules` rész, ahol hivatkozom az „igazi” modulra. Ezzel a lehetőséggel egy „igazi” modulhoz akár többfajta jegyzékfájllal más és más exportált változókat, függvényeket és cmdleteket lehet definiálni, így egy nagyon kicsi fájllal lehet több fajta interfészt készíteni egy modulhoz az aktuális igényeknek megfelelően.

2.1.7.5 Bináris és dinamikus modulok

A bináris és a dinamikus modulokkal ebben a könyvben most nem foglalkozom, mert ezek létrehozása már túlmutat egy rendszergazda szerepkörén és inkább fejlesztők tevékenységi körébe tartozik.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A bináris modul egy .NET-es építőelem (.dll), ami Visual Studio segítségével lefordított kódot tartalmaz. Cmdlet-fejlesztők ennek segítségével hozhatnak létre olyan modulokat, amelyek cmdleteket, providereket és egyéb elemeket tartalmaznak.

A dinamikus modulok olyan modulok, amelyek nincsenek elmentve diszkre, hanem futásidőben jönnek létre. Létrehozni ilyeneket a `New-Module` cmdlet segítségével lehet.

2.1.8 Profilok

Láttuk, hogy konzolfájlokkal a snapineket tudjuk automatikusan testre szabni. Ennél sokkal több lehetőséget biztosítanak a profilok.

A profilok egyszerű fájlban tárolt szkriptek, amelyek a PowerShell indításakor automatikusan lefutnak. Kicsit hasonlatosak a Windows StartUp mappáihoz. A PowerShellben összesen négy különböző hely van, ahova létrehozhatjuk a profilunkat:

Elérési út	Magyarázat
%windir%\system32\WindowsPowerShell\v1.0\ profile.ps1	Az adott gép összes felhasználójának, mindenfajta PowerShelles ügyködésére hatással van.
%windir%\system32\WindowsPowerShell\v1.0\ Microsoft.PowerShell_profile.ps1	Az adott gép összes felhasználójának, de csak az alaphelyzet szerinti Microsoft PowerShell konzol ügyködésére van hatással.
%UserProfile%\My Documents\ WindowsPowerShell\profile.ps1	Az adott gép adott felhasználójának, mindenfajta PowerShelles ügyködésére hatással van.
%UserProfile%\My Documents\ WindowsPowerShell\ Microsoft.PowerShell_profile.ps1	Az adott gép adott felhasználójának csak az alaphelyzet szerinti Microsoft PowerShell ügyködésére van hatással.

Mit jelent az, hogy "Microsoft PowerShell ügyködés"? Azt, hogy nem feltétlenül csak a hagyományos "DOS" ablakos kinézetet veheti fel a PowerShell, egyéb gyártók más, akár grafikus felületeket is csinálhatnak. Na, ilyen esetekben nekik lehet külön profil fájljuk is a Microsoftoshoz képest.

Ilyen profilban meghívhatok „include” jellegű, függvénykönyvtárak és modulok betöltését végző szkripteket, testre szabhatjuk a promptot, globális változókat definiálhatok, de akár még snapineket is betölthetek. Én a sorszámozott promptomhoz egy globális `$sor` változót és a prompt testre szabott függvénydefinícióját tettem bele egy ilyen profilba.

2.1.9 Örökítsük meg munkánkat (start-transcript)

A PowerShell magnóként is tud működni, azaz az összes parancssort, amit mi begépelünk, és az összes kimenetet, amit kapunk a konzolra képes elmenteni automatikusan egy fájlba. Ezt az üzemmódot a `start-transcript` cmdlettel tudjuk elindítani, és a `stop-transcript`-tel megállítani:

```
[6] PS C:\> Start-Transcript
Transcript started, output file is C:\Documents and Settings\Administrator\
My Documents\PowerShell_transcript.20080421212835.txt
[7] PS C:\> $a = "Valaki figyel!"
[8] PS C:\> $a.Split()
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Valaki
figyel!
[9] PS C:\> Stop-Transcript
Transcript stopped, output file is C:\Documents and Settings\Administrator\
My Documents\PowerShell_transcript.20080421212835.txt
```

Ha nem adjuk meg, hogy milyen fájlba rögzítsen, akkor az aktuális felhasználó dokumentumkönyvtárába ment, dátummal, idővel ellátott nevű szöveges fájlba. Nézzük, hogy hogyan néz ki egy ilyen fájl:

```
[11] PS C:\> Get-Content 'C:\Documents and Settings\Administrator\My Documen
ts\PowerShell_transcript.20080421212835.txt'
**********************
Windows PowerShell Transcript Start
Start time: 20080421212835
Username  : ASUS\Administrator
Machine   : ASUS (Microsoft Windows NT 5.2.3790 Service Pack 2)
**********************
Transcript started, output file is C:\Documents and Settings\Administrator\
My Documents\PowerShell_transcript.20080421212835.txt
[7] PS C:\> $a = "Valaki figyel!"
[8] PS C:\> $a.Split()
Valaki
figyel!
[9] PS C:\> Stop-Transcript
**********************
Windows PowerShell Transcript End
End time: 20080421212912
**********************
```

Látszik, hogy mindent, még a promptokat is szóról-szóra rögzítette fejléc és lábléc információk között. Ennek a fájlnak egyfajta feldolgozására látunk példát a *2.5.5 Sortörés kezelése szövegfájlokban* fejezetben.

2.1.10 Stopperoljuk a futási időt és várakozzunk (measure-command, start-sleep, get-history)

Bizonyos esetekben fontos lehet a szkriptünk futásának hatékonysága, melynek egyik ismérve, hogy milyen gyorsan fut le. Szerencsére nem kell stopperórával a kezünkben figyelni a konzolt, mert a PowerShell rendelkezik beépített stopperrel, melyet a `measure-command` cmdlettel tudunk üzembe helyezni.

Például stopperoljuk le, hogy mennyi ideig tart végiglistázni a c:\ meghajtó összes alkönyvtárát és fájlját:

```
[5] PS C:\> Measure-Command {get-childitem c:\ -recurse}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 33
Milliseconds      : 306
Ticks             : 333067130
TotalDays         : 0,000385494363425926
TotalHours        : 0,00925186472222222
TotalMinutes      : 0,555111883333333
TotalSeconds      : 33,306713
TotalMilliseconds : 33306,713
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Láthatjuk, hogy a `measure-command` paramétere egy szkriptblokk, kimenete egy `timespan` típusú objektum, amelyben mindenféle mértékegységben láthatjuk az eltelt időt.

Ha egy szkriptünknél pont az lenne a feladat, hogy ne legyen túl gyors, mert például percenként kellene, hogy valamilyen paramétert kiolvassunk, akkor a `start-sleep` cmdlettel tudunk várakozni:

```
[12] PS C:\> get-date; start-sleep 10; get-date

2008. április 24. 22:40:59
2008. április 24. 22:41:09
```

A fenti példában 10 másodpercet várakoztam, ha rövidebb várakozási időre van szükség, akkor `–milliseconds` paraméterrel lehet ezt megadni.

A `measure-command` és a `start-sleep` kombinálásával lehet olyan ciklust írni, ami például pontosan minden egész percben fut le. Azaz mérhetjük, hogy mennyi ideig tart a futása egy ciklusnak, majd kiegészítésként annyit várunk, hogy a következő perc elején induljon újra a ciklusunk.

Ha utólag vagyunk kíváncsiak egy kifejezés futtatására, akkor azt a `get-history` cmdlet segítségével lekérdezhetjük. Alaphelyzetben a `get-history` csak egy előzménylistát mutat:

```
[14] PS C:\munka> Get-History

  Id CommandLine
  -- -----------
…
  11 Get-Command
  12 get-help
  13 dir .\
```

De ha részletesebb nézetre váltunk, akkor már az időadatokat is láthatjuk:

```
[15] PS C:\munka> Get-History | Format-List
…

Id                 : 11
CommandLine        : Get-Command
ExecutionStatus    : Completed
StartExecutionTime : 2010. 01. 12. 19:48:46
EndExecutionTime   : 2010. 01. 12. 19:48:50

Id                 : 12
CommandLine        : get-help
ExecutionStatus    : Completed
StartExecutionTime : 2010. 01. 12. 19:48:54
EndExecutionTime   : 2010. 01. 12. 19:48:54

Id                 : 13
CommandLine        : dir .\
ExecutionStatus    : Completed
StartExecutionTime : 2010. 01. 12. 19:48:59
EndExecutionTime   : 2010. 01. 12. 19:49:00
```

Ennek birtokában keressük meg a három leghosszabb ideig futó kifejezést:

```
[19] PS C:\munka> Get-History | Select-Object -Property id, commandline, @{n =
"Futásidő"; e={$_.EndExecutionTime - $_.StartExecutionTime}} | Sort-Object -Pro
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
perty Futásidő -Descending | Select-Object -First 3

                        Id CommandLine               Futásidő
                        -- -----------               --------
                        11 Get-Command               00:00:04.7500000
                        15 Get-History | Format-L... 00:00:00.6250000
                        12 get-help                  00:00:00.4062500
```

2.1.11 Előrehaladás jelzése (write-progress)

Ha már időmérés és várakozás, akkor érdemes megemlíteni, hogy szép „progress bar”-t, azaz előrehaladás-jelzőt is megjeleníthetünk PowerShellben a `write-progress` cmdlet segítségével. Ez főleg akkor hasznos, ha időigényesebb valamelyik parancssorunk, szkriptünk és nyomon szeretnénk követni, hogy ténylegesen hol is tart a futás. Példaként nézzünk egy „analóg” órát:

```
 Óra
    12
    [ooooooooooooooooooooooooooooooo                                    ]
   Perc
      53
      [oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo      ]
     MP
        53
        [ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo     ]

[91] PS C:\>while ($true) {
>> $t = Get-Date
>> $óra = $t.hour
>> $perc = $t.Minute
>> $mp = $t.Second
>> Write-Progress -Activity "Óra" -status "$óra" -Id 1 -PercentComplete ($ó
a/24*100)
>> Write-Progress -Activity "Perc" -status "$perc" -Id 2 -ParentId 1 -Perce
tComplete ($perc/60*100)
>> Write-Progress -Activity "MP" -status "$mp" -ParentId 2 -PercentComplete
($mp/60*100)
>> Start-Sleep -Seconds 1
>> }
>>
```

Sajnos nyomtatásban nem annyira „szép” a kép, nem látszik az ablak tetején a türkizes háttérszínem, de a kis karikákból álló előrehaladás-jelzőt azért látható. Maga a szkript, ami ezt produkálja a karikás sorok alatt található. Egy `while` ciklusban folyamatosan kiolvasom az aktuális rendszeridőt, szétszedem külön óra, perc, másodpercre. Ezután jön a lényegi rész. Az egyes „csíkoknak” adhatok címkét (`Activity` paraméter), státusz címkét (`Status`), azonosító számot (`Id` paraméter), az előrehaladás értékét (0 és 100 közötti szám, `PercentComplete` paraméter). Látszik, hogy a kijelzők egymásba is ágyazhatók, ha definiálom a `ParentID` paraméterrel a „szülő” csíkot. Ekkor a „gyerek” a szülő alatt fog megjelenni, kicsit beljebb.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.1.12 Levélküldés

Szkriptekben gyakori feladat a levélküldés, hiszen egy ütemezett szkript eredményéről legkényelmesebben e-mailben kaphatunk visszajelzést. A PowerShell 2.0 már rendelkezik e-mail küldésére képes cmdlettel, ez a `Send-MailMessage`. Ezen kívül van egy környezeti változónk, a `$PSEmailServer`, ahol megadhatjuk az alaphelyzet szerinti SMTP kiszolgálónk nevét, így a levélküldéskor ezt használhatjuk és nem kell újra és újra megadni a levelező szerverünket.

A parancs használata nagyon egyszerű és magától értetődő:

```
[59] PS C:\> $variable:PSEmailServer = "mail.iqjb.hu"
[60] PS C:\> Send-MailMessage -From "soost@iqjb.hu" -Body "kakukk" -Subject "Le
vel" -To "soost@iqjb.hu" -Credential iqjb\soost
```

Ennek hatására a megadott SMTP kiszolgáló „relay”-t hajt végre a levelünkkel.

Megjegyzés

Ha nem kellően körültekintően állítjuk be a levelezésünket, akkor könnyen levélszemétként értékelődhet levelünk, így ha nem történik meg a kézbesítés, akkor gyanakodjunk erre.

Ha még precízebbek szeretnénk lenni, akkor küldhetünk HTML formátumú levelet, ilyenkor alkalmazzuk a `-BodyAsHTML` kapcsolót is. A levelünknek több címzettje is lehet, ilyenkor a `-To` paraméternek több címzettet tömbként adhatunk át.

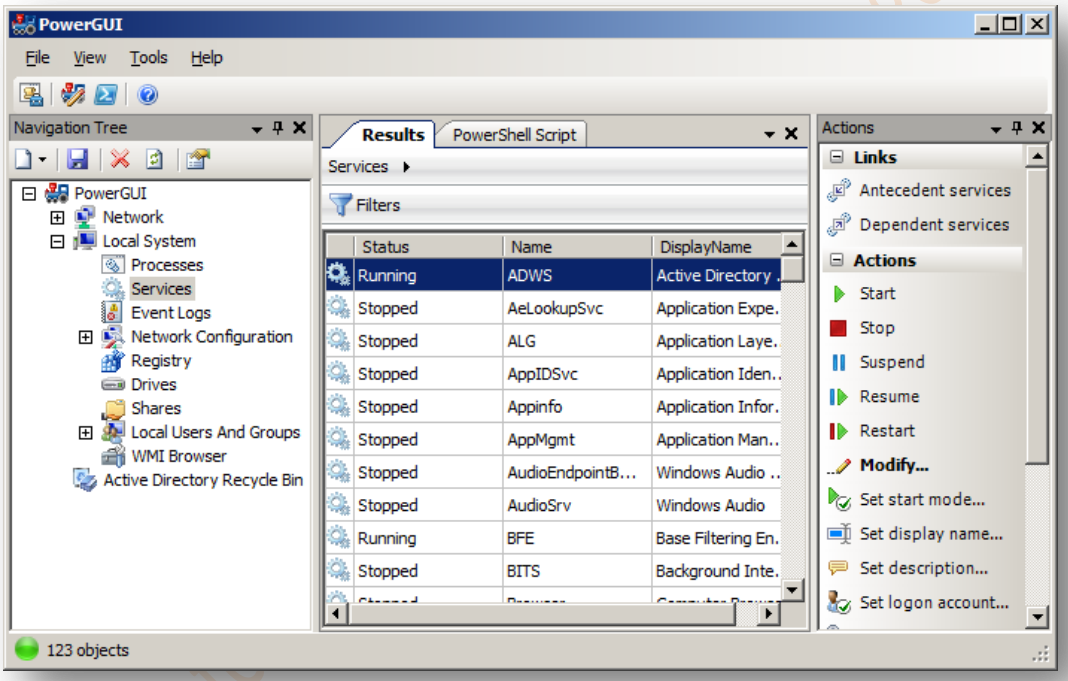
A könyv tanfolyami felhasználása nem engedélyezett!

2.2 Segédprogramok

Ha kezdők vagyunk PowerShellben, akkor mindenképpen érdemes felszerelkeznünk néhány hasznos segédprogrammal, amelyek megkönnyítik a munkánkat és a tanulási folyamatot. Ebben a fejezetben néhány ilyen programot mutatok be, amelyek többsége ingyenesen letölthető, használható, vagy legalábbis van ingyenes próbaváltozata.

2.2.1 PowerGUI, PowerGUI Script Editor

A PowerShellhez az egyik legpraktikusabb segédprogram a Quest Software PowerGUI csomagja. Ez két fő programból áll. Az első maga a PowerGUI, ami jó néhány előre elkészített PowerShell parancsot, kifejezést tartalmaz egy fastruktúrába fűzve:



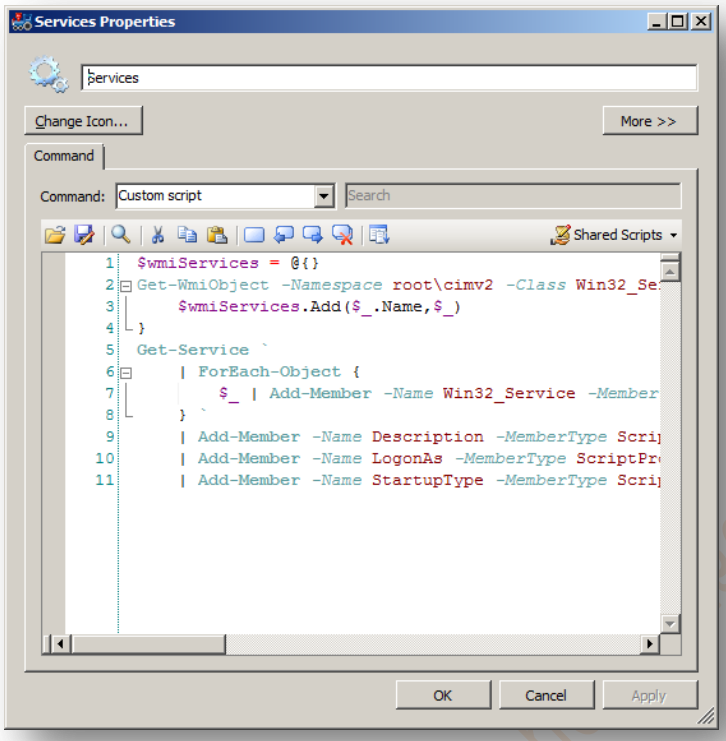
46. ábra A PowerGUI felülete

A fastruktúra egyébként bővíthető, például Active Directory elemekkel, attól függően, hogy milyen bővítményeket telepítettünk a gépünkre, és mi magunk is készíthetünk hozzá faelemeket.

A programban az ikonnal jelzett faelemek mögött PowerShell szkriptek vannak:

A könyv tanfolyami felhasználása nem engedélyezett!

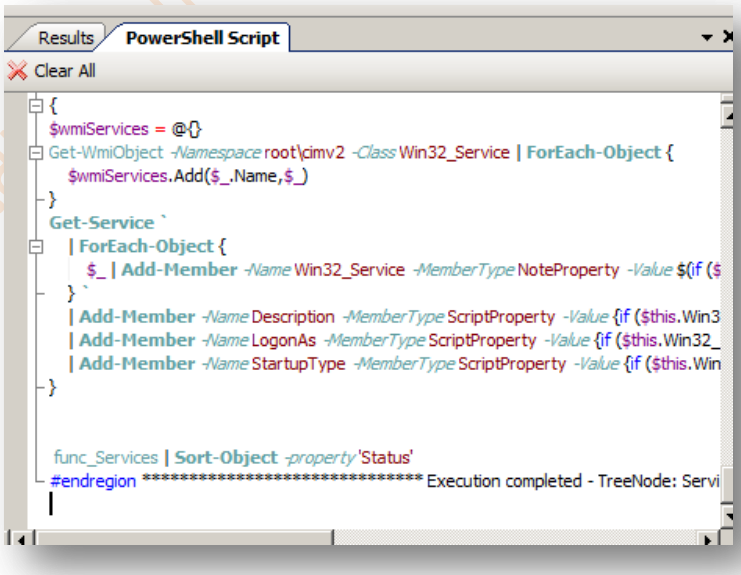
© Soós Tibor – soost@ IQJB.hu



47. ábra A PowerGUI faelemek mögötti PowerShell parancsok

Ez lehet egy egyszerű PowerShell parancs, ún. cmdlet, mint a Processes-nél, vagy bonyolultabb szkript, mint a Services-nél.

Az ablak tetején, ha bekapcsoljuk a View menüben „PowerShell Script” opciót, akkor a „*PowerShell Script*” fülre kattintva meg lehet figyelni, hogy pontosan mit hajt végre a PowerGUI amikor kattintgatunk a felületén:



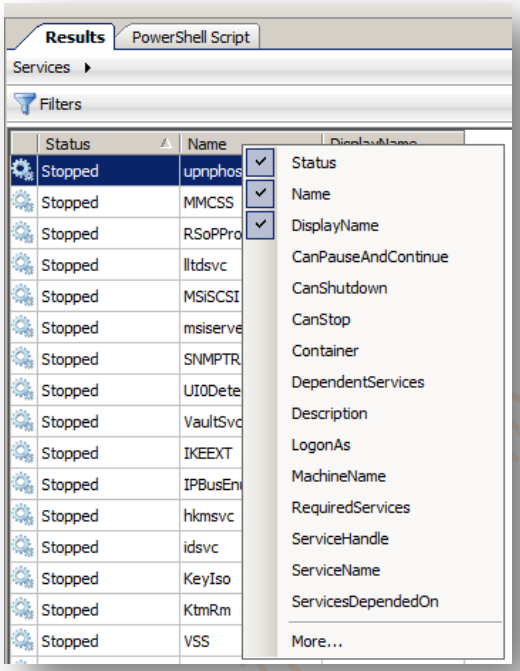
48. ábra A PowerGUI által végrehajtott parancsok

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Például a fenti képen, középtájon látszik, hogy amikor a szolgáltatások listáján a „*Status*” oszlopra kattintottam, akkor erre egy `func_Services | Sort-Object -property 'Status'` parancs hajtódott végre. Kezdőként, elemezgetve az általa végrehajtott kifejezéseket, sokat lehet tanulni.

Illetve a kimenetet mutató rács oszlopainak fejlécén jobb egérgombbal kattintva lehet kérni egy listát, ami a megjelenített objektumok tagjellemzői közül a tulajdonságokat mutatja meg:

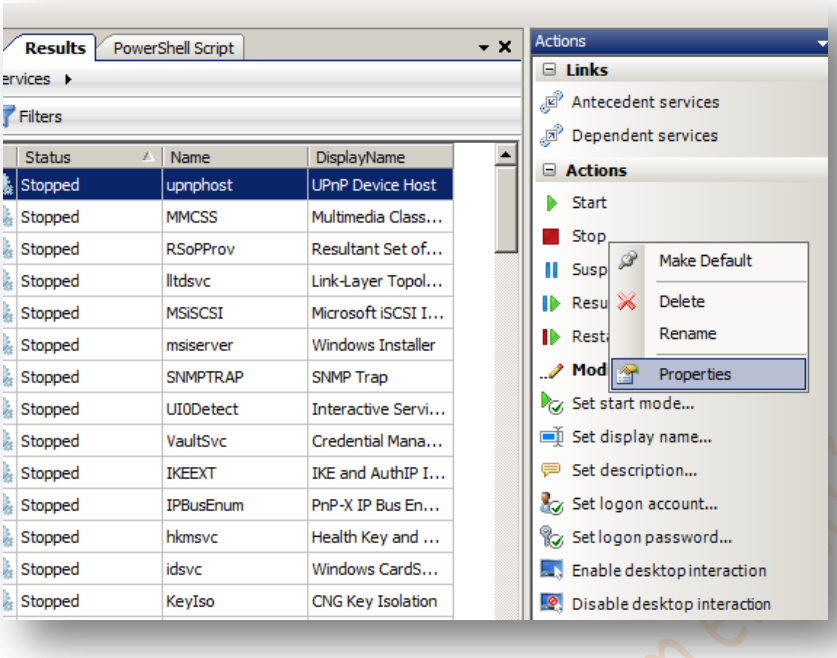


49. ábra Oszlopok, azaz tulajdonságok kiválasztása

Ezek közül lehet kijelölni azokat, amelyeket a rácsban meg akarunk jeleníteni. Ilyen szempontból ez az eszköz „okosabb”, mint a PowerShell GridView eszköze.

Ezen kívül ez az eszköz képes arra is, hogy a „Results” rácsban megjelenő elemekkel kapcsolatos tevékenységeket is létrehozzunk. A szolgáltatások esetében például ott a „Start”, „Stop”, „Resume”, stb. Ezek mögött mind-mind PowerShell szkriptek találhatók, amiket meg is nézhetünk, ha jobbegérgombbal kattintunk a tevékenység nevére és a megjelenő menüben kattintunk a „Properties”-re.

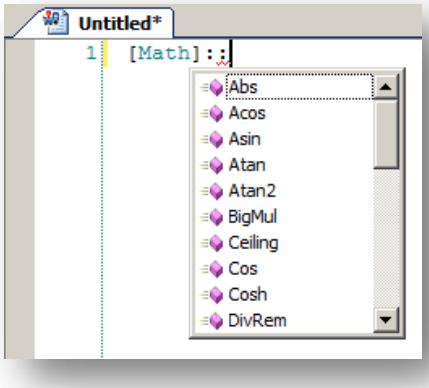
A könyv tanfolyami felhasználása nem engedélyezett!



50. ábra Eredmény-objektumokkal kapcsolatot tevékenységek

Az egész PowerGUI testre szabható, mi magunk is tehetünk bele faelemeket és tevékenységeket. Ezzel gyakorlatilag egy jó kis szkripttárként is tud működni a PowerGUI, nem kell a fájlrendszerben keresgélni a szkriptjeinket, mindegyik ott van a PowerGUI-ban. Ráadásul az ilyen testreszabásainkat ki is rakhatjuk egy un. PowerPack fájlba, amit más PowerGUI felhasználók egyszerűen be tudnak importálni.

A másik eszköz a PowerGUI csomagban a Script Editor. Ez hasonló szkriptszerkesztő, mint a PowerShell ISE, de annál bizonyos vonatkozásaiban még többet is tud. Azon kívül, hogy gépelés során a parancsokon rögtön szintaxisellenőrzést végez, különböző színnel jelöli a különböző fajtájú szövegelemeket (parancs, kifejezés, változó, sztring, stb.), még a .NET osztályok tulajdonságait is felderíti, azaz a Reflector program szolgáltatásait is részben nyújtja:

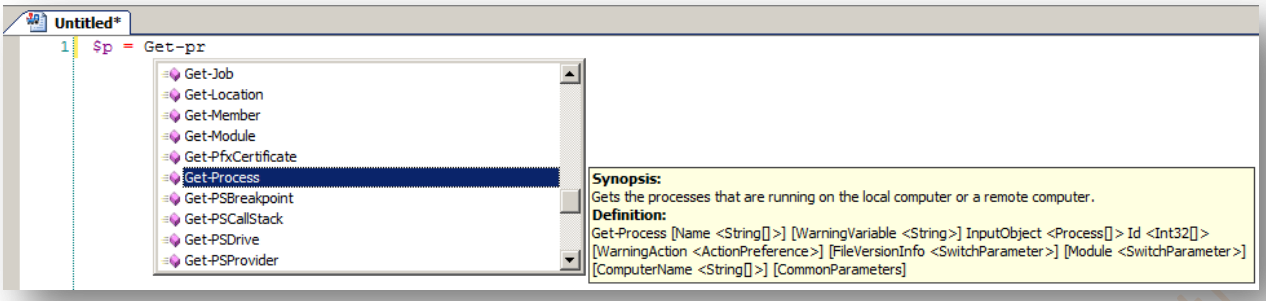


51. ábra A .NET osztályok tulajdonságainak felderítése

Gépelés közben a lehetséges parancsokat egy listában azonnal megjeleníti, a parancsok argumentumlistáját kis felugró ablakban kiírja. F1-re az adott parancs helpjét is megjeleníti.

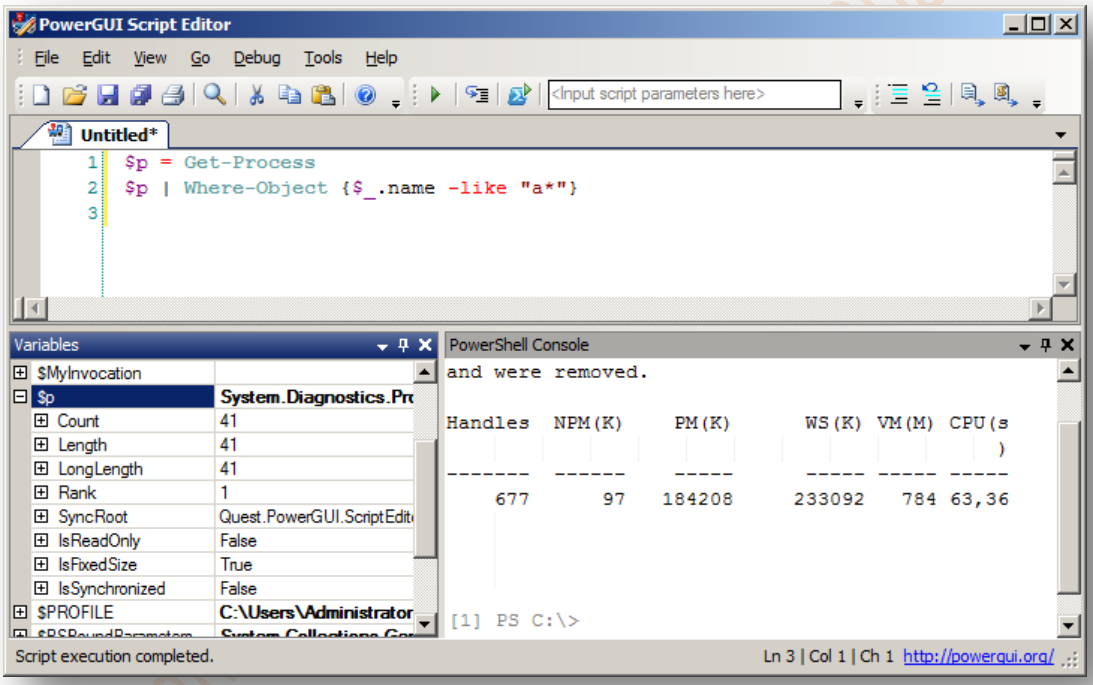
A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



52. ábra Parancskiválasztó a PowerGUI Script Editorban

Az ablak felső részében szerkeszthetjük a kódot, bal alsó részben láthatjuk változóinkat és azok értékét, a jobb alsó részben a futtatott kód kimenetét olvashatjuk:



53. ábra A PowerGUI szkriptszerkesztője

A jobb alsó rész legalján egy interaktív prompt is helyet foglal, ad-hoc parancsokat oda is be tudunk gépelni és végre tudjuk azokat hajtani.

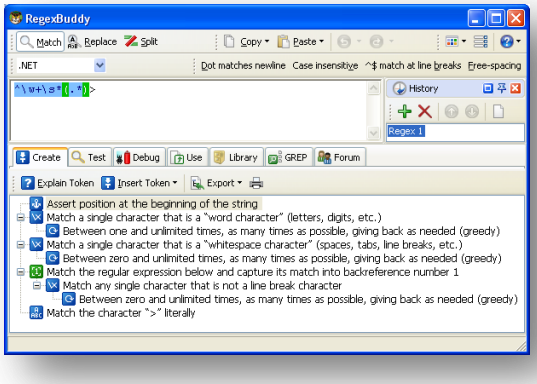
2.2.2 RegexBuddy

Korábban foglalkoztam a reguláris kifejezésekkel, amelyek a sztringek összehasonlításában, vizsgálatában segítenek. Például meg lehet ilyen kifejezések segítségével állapítani, hogy egy szöveg vajon e-mailcímet tartalmaz-e, vagy pl. telefonszámot. Regex kifejezésekkel ki lehet nyerni szövegekből adott formátumú, adott mintára illeszkedő karaktersorozatokat.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Regex kifejezéseket írni nem egyszerű, ebben segít a *RegexBuddy*.



54. ábra Barátunk a RegexBuddy

Az ablak felső részén látható, hogy kiválaszthatjuk a regex kifejezésünk „tájszólását”, ami a PowerShell esetében .NET. Ez alatt van a szerkesztő felület, legalul meg egy fülrendszeren sok minden. A *Create* fül értelmezését adja a regex kifejezésemnek, és regex elemeket, un. tokeneket is hozzá tudok fűzni. Ami még érdekes számunkra a *Test* fül, ahol minta sztringeket tudok begépelni és ellenőrizni tudom, hogy a regex mintám illeszkedik-e arra, amire kellene, illetve nem illeszkedik-e arra, amire nem kell.

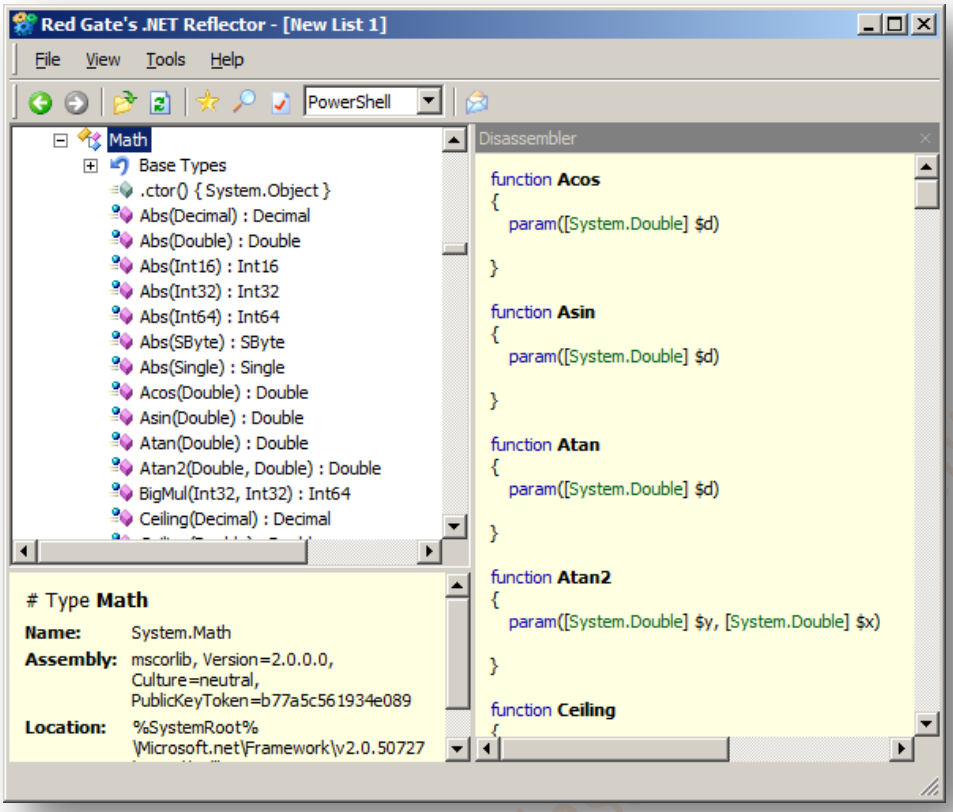
Hasznos segítséget nyújt a *Library* fül, ahol sok probléma megoldásához vannak előre elkészített regex kifejezések, amiket segítségül tudunk hívni és testre tudjuk szabni a saját igényeinknek megfelelően.

2.2.3 Reflector

A PowerShell a .NET Frameworkre épül, a keretrendszer osztályaival, objektumaival dolgozik, így fontos tudni, hogy a keretrendszerben mi található. A Visual Studio természetesen az Object Browser-ével segít a .NET osztályok felderítésében, de a PowerShell felhasználók zöme nem fejlesztő, nincs szükségük általában a Visual Studiora, és csak az Object Browser miatt telepíteni azt elég nagy luxus.

Az interneten Lutz Röder jóvoltából elérhető egy Reflector nevű kis ingyenes program (http://www.red-gate.com/products/reflector), amellyel helyettesíteni lehet az Object Browsert. Ráadásul ez tartalmaz egy PowerShell add-in-t, amivel PowerShell szintaxissal lehet megjeleníteni a .NET-es objektumokat.

A könyv tanfolyami felhasználása nem engedélyezett!



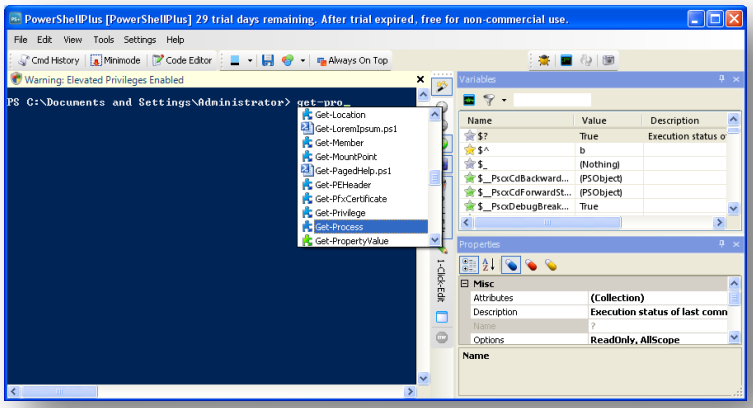
55. ábra Rendszergazdák Object Browsere: .NET Reflector

Ha találtunk olyan .NET osztályt, vagy annak valamilyen tagját, ami érdekel minket, akkor Ctrl+M billentyűkombinációval közvetlenül elő tudjuk hívni az MSDN website megfelelő lapját, ahol részletes leírást kapunk a kiválasztott elemünkről.

2.2.4 PSPlus

Ez egy másik, jóval összetettebb, több szolgáltatást nyújtó szkript editor.

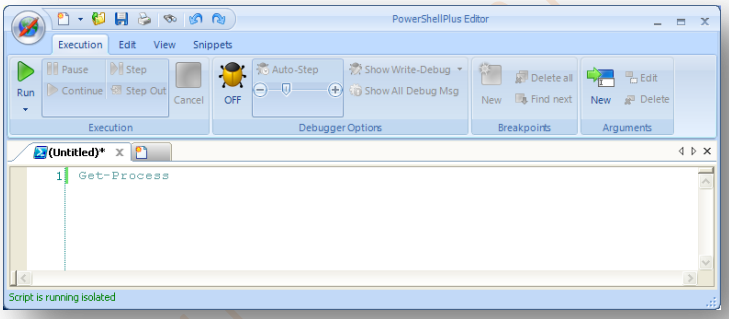
A könyv tanfolyami felhasználása nem engedélyezett!



56. ábra PowerShell Plus felülete

Ez az eszköz integrál magában az eredeti PowerShell konzolablakhoz nagyon hasonló ablakrészt, láthatjuk a változókat és egyéb paramétereket.

Külön szkriptszerkesztője is van:



57. ábra PowerShell Plus Script Editora

Ez az Office 2007 stílusú szerkesztőfelület hasonló szolgáltatásokat nyújt, mint a PowerGUI Script Editora. Igazából én ezt nem használom, rendszergazda tevékenységekhez a PowerGUI ISE vagy a PowerGUI bőven elég.

A könyv tanfolyami felhasználása nem engedélyezett!

2.3 Hibakezelés

Az elméleti részben nem foglalkoztam részletesen a hibákkal, hiszen elméletileg minden tökéletes ☺, na de a gyakorlat! Nézzük meg, hogy a PowerShell milyen lehetőségeket nyújt a hibák detektálására, elhárítására, illetve milyen lehetőségek vannak arra, ha a felmerülő hibákat én akarom a szkriptemben lekezelni.

2.3.1 Megszakító és nem megszakító hibák

Elsőként nézzünk pár fogalmat. A PowerShellben két hibafajta van, a „terminating error”, azaz a futást mindenképpen megszakító, és a „nonterminating error”, azaz a futást nem feltétlenül megszakító hiba.

Megszakító hibák például a szintaktikai hibák, amikor elgépelünk valamit. Vagy például amikor nullával szeretnénk osztani. Mi magunk is generálhatunk ilyen hibákat a korábban már látott `throw` kulcsszóval, amikor például egy függvényünknek nem ad át a felhasználó minden fontos paramétert.

Előfordulhatnak olyan hibák, amelyek előállásakor nem kívánjuk, hogy a szkript futása megszakadjon, de azért szeretnénk értesülni ezekről. Ez főleg olyan cmdleteknél és függvényeknél jöhet jól, amelyek csőelemeket dolgoznak fel, és egy-két csőelem esetében megengedjük, hogy ne fusson le a szkript, de azért a többi elemre nyugodtan próbálkozzon.

Nézzünk példát a kétfajta hibára, elsőként a megszakító hibára, rögtön két példát is:

```
[7] PS C:\old> "Eleje"; 15/0; "Vége"
Attempted to divide by zero.
At line:1 char:13
+ "Eleje"; 15/ <<<< 0; "Vége"
    + CategoryInfo          : NotSpecified: (:) [], ParentContainsErrorRecord
   Exception
    + FullyQualifiedErrorId : RuntimeException
```

A [7]-es sorban 0-val osztok. Ez olyannyira „terminating”, hogy már a parancssor ellenőrzése során kiszúrja ezt a hibát, és már az „Eleje” sem fut le.

```
[8] PS C:\old> "Eleje"; throw "kakukk"; "Vége"
Eleje
kakukk
At line:1 char:15
+ "Eleje"; throw <<<<  "kakukk"; "Vége"
    + CategoryInfo          : OperationStopped: (kakukk:String) [], RuntimeEx
   ception
    + FullyQualifiedErrorId : kakukk
```

A [8]-as sorban én magam okoztam egy megszakító hibát a `throw` kulcsszó segítségével.

Most nézzünk nem megszakító hibára példát:

```
[9] PS C:\old> "Eleje"; Remove-Item c:\old\nincs.txt; "Vége"
Eleje
Remove-Item : Cannot find path 'C:\old\nincs.txt' because it does not exist.
At line:1 char:21
+ "Eleje"; Remove-Item <<<<  c:\old\nincs.txt; "Vége"
    + CategoryInfo          : ObjectNotFound: (C:\old\nincs.txt:String) [Remo
   ve-Item], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.Remo
   veItemCommand
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Vége
```

A fenti példában nem megszakító hiba történt, amikor törölni akartam egy nem létező fájlt, ugyan hibát kaptam, de kiírásra került a „Vége” is.

Mi magunk is befolyásolhatjuk a hibák lefolyását. Akár parancsonként vagy akár az adott konzolra vonatkozóan is. Először nézzük, hogy a cmdletek esetében mit tehetünk. A parancsokhoz tartozó súgókban legtöbbször valami hasonlót látunk:

```
[18] PS C:\old> (get-help remove-item).syntax

Remove-Item [-path] <string[]> [-recurse] [-force] [-include <string[]>] [-
exclude <string[]>] [-filter <string>] [-credential <PSCredential>] [-whatI
f] [-confirm] [<CommonParameters>]
Remove-Item [-literalPath] <string[]> [-recurse] [-force] [-include <string
[]>] [-exclude <string[]>] [-filter <string>] [-credential <PSCredential>]
[-whatIf] [-confirm] [<CommonParameters>]
```

Nézzük meg, a „`CommonParameters`” mi lehet?

Paraméter	Magyarázat
Verbose	Bőbeszédes kimenetet ad a művelet lefolyásáról.
Debug	Hibakereső információkat ad, és interaktív módon lekezelhetők a hibák.
ErrorAction	Az előzőhöz hasonló, de nem csak interaktívan, hanem fixen beállítható hibakezelési mód: `Continue` [default] - folytat, `Stop` - megáll, `SilentlyContinue` – figyelmeztetés nélkül továbbmegy, `Inquire` - rákérdez.
ErrorVariable	Saját hibaváltozónk neve ($ jel nélkül!). Az `$error` változó mellett ide is betöltődik a hibát leíró objektum.
OutVariable	Kimenetet ide tölti be.
OutBuffer	Az objektum-puffer mérete, ennyi elemet „magában” tart, mielőtt továbbítja az outputot a következő csőszakasznak.
WarningAction	Hasonló, mint az ErrorAction, csak épp a figyelmeztetések bekövetkezését kezeli.
WarningVariable	Figyelmeztetések betöltése alternatív változóba.

A `-debug` paraméter pont a mostani témánkba vág, nézzük, milyen opciókat kínál:

```
[23] PS C:\old> "Eleje"; get-content blabla.txt -debug; "Vége"
Eleje

Confirm
Cannot find path 'C:\old\blabla.txt' because it does not exist.
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help
(default is "Y"):h
Get-Content : Command execution stopped because the user selected the Halt opt
ion.
At line:1 char:21
+ "Eleje"; get-content <<<<  blabla.txt -debug; "Vége"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    + CategoryInfo          : OperationStopped: (:) [Get-Content], ParentCont
   ainsErrorRecordException
    + FullyQualifiedErrorId : ActionPreferenceStop,Microsoft.PowerShell.Comma
   nds.GetContentCommand
```

Láthatjuk, hogy ezzel a paraméterrel a nem megszakító hibák lefolyásáról dönthetünk interaktív módon. Ha a „Halt” opciót választom, akkor kiírta a megállás okát, és a „Vége” nem futott le. Nézzünk, hogy hogyan viselkedik más válasz esetén:

```
[24] PS C:\old> "Eleje"; get-content blabla.txt -debug; "Vége"
Eleje

Confirm
Cannot find path 'C:\old\blabla.txt' because it does not exist.
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help
(default is "Y"):a
Get-Content : Cannot find path 'C:\blabla.txt' because it does not exist.
At line:1 char:21
+ "Eleje"; get-content <<<<  blabla.txt -debug; "Vége"
    + CategoryInfo          : ObjectNotFound: (C:\blabla.txt:String) [Get-Con
   tent], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetC
   ontentCommand

Vége
```

Ha „Yes” vagy „Yes to All” opciót választom, akkor a cmdlet úgy fut le, ahogy eddig láttuk a nem megszakító hibáknál, azaz kiírta a hibát, de tovább futott. Nézzük a további választási lehetőségünket:

```
[25] PS C:\old> "Eleje"; get-content blabla.txt -debug; "Vége"
Eleje

Confirm
Cannot find path 'C:\old\blabla.txt' because it does not exist.
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help
(default is "Y"):s
[26] PS C:\old >>> 1+1
2
[27] PS C:\old>>> exit

Confirm
Cannot find path 'C:\Users\tibi\blabla.txt' because it does not exist.
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help
(default is "Y"):h
```

Ha a „Suspend” opciót választjuk, akkor a szkript futása megszakad, egy alprompt nyílik és ott bármit begépelhetünk, ellenőrizhetünk, majd az exit kulcsszót beírva visszatérhetünk a fenti állapothoz, azaz újra rákérdez, hogy mi legyen.

Nézzük a másik témánkba vágó „CommonParameter”-t, az –ErrorAction-t:

```
[44] PS C:\old> "Eleje"; get-content blabla.txt -ErrorAction Stop; "Vége"
Eleje
Get-Content : Cannot find path 'C:\blabla.txt' because it does not exist.
At line:1 char:21
+ "Eleje"; get-content <<<<  blabla.txt -ErrorAction Stop; "Vége"
    + CategoryInfo          : ObjectNotFound: (C:\blabla.txt:String) [Get-Con
```

```
   tent], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetC
   ontentCommand
```

Ha a `Stop` opciót választjuk, akkor a szkript a nem megszakító hibáknál is megszakítódik.

```
[45] PS C:\old> "Eleje"; get-content blabla.txt -ErrorAction SilentlyContinu
e; "Vége"
Eleje
Vége
```

A `SilentlyContinue` opcióval folytatja a szkriptet és nem is ad semmilyen hibajelzést.

```
[46] PS C:\old> "Eleje"; get-content blabla.txt -ErrorAction Inquire; "Vége"

Eleje

Confirm
Cannot find path 'C:\old\blabla.txt' because it does not exist.
[Y] Yes  [A] Yes to All  [H] Halt Command  [S] Suspend  [?] Help
(default is "Y"):a
Get-Content : Cannot find path 'C:\blabla.txt' because it does not exist.
At line:1 char:21
+ "Eleje"; get-content <<<<  blabla.txt -ErrorAction Inquire; "Vége"
    + CategoryInfo          : ObjectNotFound: (C:\blabla.txt:String) [Get-Con
   tent], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetC
   ontentCommand

Vége
```

Ha az `Inquire` opciót választjuk, akkor - hasonlóan a `-debug` paraméter használatakor látottakhoz - rákérdez a folytatás miként jére.

2.3.2 Hibajelzés kiolvasása ($error)

Az előzőekben a hibajelzéseket a képernyőn olvastuk el, de ezeket a PowerShell automatikusan egy `$error` tömbbe gyűjti. A [0]-ás indexű elem mindig a legfrissebb hibajelzést tartalmazza, és így tolja mindig el eggyel a hibákat. Ez azért jó, mert ha ki is kapcsoljuk a hibajelzést (`SilentlyContinue`), az `$error` változó ennek ellenére őrizni fogja a fellépő hibát:

```
[54] PS C:\old> "54Eleje"; get-content blabla.txt -ErrorAction SilentlyConti
nue; "54Vége"
54Eleje
54Vége
[55] PS C:\old> $error[0]
Get-Content : Cannot find path 'C:\blabla.txt' because it does not exist.
At line:1 char:23
+ "54Eleje"; get-content <<<<  blabla.txt -ErrorAction SilentlyContinue; "54Vé
ge"
    + CategoryInfo          : ObjectNotFound: (C:\blabla.txt:String) [Get-Con
   tent], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetC
   ontentCommand
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ráadásul – most már nem is csodálkozunk – a `$error` elemei is objektumok! Nézzük meg a tagjellemzőit:

```
[59] PS C:\old> $error[0] | gm


   TypeName: System.Management.Automation.ErrorRecord

Name                  MemberType     Definition
----                  ----------     ----------
Equals                Method         bool Equals(System.Object obj)
GetHashCode           Method         int GetHashCode()
GetObjectData         Method         System.Void GetObjectData(System.Runti...
GetType               Method         type GetType()
ToString              Method         string ToString()
CategoryInfo          Property       System.Management.Automation.ErrorCate...
ErrorDetails          Property       System.Management.Automation.ErrorDeta...
Exception             Property       System.Exception Exception {get;}
FullyQualifiedErrorId Property       System.String FullyQualifiedErrorId {g...
InvocationInfo        Property       System.Management.Automation.Invocatio...
PipelineIterationInfo Property       System.Collections.ObjectModel.ReadOnl...
TargetObject          Property       System.Object TargetObject {get;}
PSMessageDetails      ScriptProperty System.Object PSMessageDetails {get=& ...
```

Itt igazából a tulajdonságok az érdekesek számunkra:

```
 [62] PS C:\old> $error[0].categoryinfo


Category   : ObjectNotFound
Activity   : Get-Content
Reason     : ItemNotFoundException
TargetName : C:\old\blabla.txt
TargetType : String



[63] PS C:\old> $error[0].errordetails
[64] PS C:\old> $error[0].fullyqualifiederrorid
PathNotFound,Microsoft.PowerShell.Commands.GetContentCommand
[65] PS C:\old> $error[0].invocationinfo


MyCommand        : Get-Content
ScriptLineNumber : 1
OffsetInLine     : -2147483648
ScriptName       :
Line             : "54Eleje"; get-content blabla.txt -ErrorAction SilentlyC
                   ontinue; "54Vége"
PositionMessage  :
                   At line:1 char:23
                   + "54Eleje"; get-content  <<<< blabla.txt -ErrorAction S
                   ilentlyContinue; "54Vége"
InvocationName   : get-content
PipelineLength   : 1
PipelinePosition : 1



[66] PS C:\old> $error[0].targetobject
C:\old\blabla.txt
```

nem

A `CategoryInfo` tulajdonság önmagában nagyon sok mindent elárul a hibáról, az `InvocationInfo` tulajdonságban még további részleteket láthatunk. Ezen információk birtokában tényleg nem jelenthet gondot a hibák felderítése és elhárítása.

Ha nem annyira lényeges, hogy sok információ jelenjen meg a hibákról, akkor átválthatjuk a hibajelentés nézetét egy egyszerűsített nézetre az `$ErrorView` változó `NormalView`-ról való átállításával `CategoryView`-ra:

```
[26] PS C:\> get-item c:\aaa\a.txt
Get-Item : Cannot find path 'C:\aaa\a.txt' because it does not exist.
At line:1 char:9
+ get-item <<<<  c:\aaa\a.txt
    + CategoryInfo          : ObjectNotFound: (C:\aaa\a.txt:String) [Get-Item
   ], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetI
   temCommand

[27] PS C:\> $ErrorView = "CategoryView"
[28] PS C:\> get-item c:\aaa\a.txt
ObjectNotFound: (C:\aaa\a.txt:String) [Get-Item], ItemNotFoundException
```

Így jóval egyszerűbb hibajelzést kapun, ami ott, ahol esetleg laikusok futtatják a szkriptjeinket, jól jöhet, mert kevésbé rémisztő.

2.3.3 Hibakezelés globális paraméterei

Az előzőekben látott hibakezelési módszerek úgy működtek, hogy egy adott cmdletnél állítottam be a „`CommonParameter`-ek” segítségével, hogy az hogyan reagáljon egy hibára. Azonban ez elég nehézkes egy hosszabb szkript esetében. Szerencsére ezt a fajta működést globálisan is beállíthatjuk. Nézzük meg, hogy milyen „error”-ral kapcsolatos változóink vannak:

```
[69] PS C:\old> Get-ChildItem variable:\*error*

Name                           Value
----                           -----
Error                          {PathNotFound,Microsoft.PowerShell.Comman...
ReportErrorShowSource          1
ReportErrorShowStackTrace      0
ReportErrorShowExceptionClass  0
ErrorActionPreference          Continue
MaximumErrorCount              256
ReportErrorShowInnerException  0
ErrorView                      NormalView
```

Nézzük ezek közül azoknak a magyarázatát, amelyek a mindennapi használatkor érdekesek:

Változó	Magyarázat
$Error	A korábban már látott hibajelzések tömbje.
$ErrorActionPreference	Globális hibakezelési mód: `Continue` [default] - folytat, `Stop` - megáll, `SilentlyContinue` – figyelmeztetés nélkül

	továbbmegy, `Inquire` - rákérdez.
$MaximumErrorCount	Az `$error` tömb maximális mérete. Az ennél régebbi (nagyobb sorszámú) hibajelzések kihullnak a tömbből.
$ErrorView	A hibajelzések nézete: `Normal` vagy `CategoryView`

Ezekkel tehát a hibák globális kezelését oldhatjuk meg, leginkább ugye az `$ErrorActionPreference` változóval.

2.3.4 Hibakezelés saját függvényeinkben (trap, try, catch, finally)

Az eddigi sok okosságot a hibakezeléssel kapcsolatban mind a cmdletek fejlesztőinek köszönhetjük. Azaz hogy egy cmdlet rendelkezik az `-ErrorAction` paraméterrel, és ennek különböző értékeire hogyan reagál, az mind leprogramozandó. Ha mi készítünk egy függvényt vagy szkriptet, és abban mi is figyelembe akarjuk venni a globális `$ErrorActionPreference` változó értékét, vagy mi is implementálni akarjuk az `-ErrorAction` paramétert, akkor ezt a függvényünk, szkriptünk belsejében nekünk le kell programozni.

Ehhez a PowerShell nyelvi szinten a `trap` kulcsszót biztosítja az 1.0 verziótól kezdve. Ennek szintaxisa így néz ki:

```
trap [ExceptionType] {code; keyword}
```

A `trap`-pel tehát át lehet venni a vezérlést a hibák felmerülésekor. Nézzünk erre pár példát:

```
trap
{
     "Hibajelenség: $_"
}

Remove-Item c:\nemlétezőfile.txt
```

A fenti példa egy szkript, nézzük mit ad ez kimenetként, ha lefuttatom:

```
PS C:\> .\_munka\powershell2\scripts\trap1.ps1
Remove-Item : Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
At C:\_munka\powershell2\scripts\trap1.ps1:6 char:12
+ Remove-Item <<<<  c:\nemlétezőfile.txt
    + CategoryInfo          : ObjectNotFound: (C:\nemlétezőfile.txt:String) [
   Remove-Item], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.Remo
   veItemCommand
```

Nem sok mindent látunk a „gyári” hibajelzéshez képest. Mivelhogy a `trap` csak megszakító hibákra éled fel, azaz a fenti nem megszakító hibát át kell alakítani megszakító hibává. Nézzük a módosított szkriptet:

```
trap
{
     "Hibajelenség: $_"
}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Remove-Item c:\nemlétezőfile.txt -ErrorAction Stop
```

Itt már mindenképpen megállíttatom a cmdlet futását bármilyen hibajelenségre. És ennek kimenete:

```
PS C:\> .\_munka\powershell2\scripts\trap2.ps1
Hibajelenség: Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
Remove-Item : Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
At C:\_munka\powershell2\scripts\trap2.ps1:6 char:12
+ Remove-Item <<<<  c:\nemlétezőfile.txt -ErrorAction Stop
    + CategoryInfo          : ObjectNotFound: (C:\nemlétezőfile.txt:String) [
   Remove-Item], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.Remo
   veItemCommand
```

Itt már megjelenik az én `trap` kódom is. A `trap` megkapja a hibát jelző objektumot a `$_` változóban, amit ki is írtam. Viszont nem rendelkeztem arról, hogy a saját hibakezelő rutinom lefutása után mi történjen, így utána visszakapja a vezérlést a PowerShell, ami szintén kiírja a hibajelzést. Ez felesleges, így módosítsuk a hibakezelést:

```
trap
{
    "Hibajelenség: $_"
    continue
}

Remove-Item c:\nemlétezőfile.txt -ErrorAction Stop
```

Ebben a változatban a `trap` szkriptblokkjába elhelyeztem egy `continue` kulcsszót, ami azt jelzi a PowerShellnek, hogy már minden rendben, neki már nem kell foglalkozni a hibával. Nézzük ekkor a kimenetet:

```
PS C:\> .\_munka\powershell2\scripts\trap3.ps1
Hibajelenség: Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
```

Fejlesszük tovább a kódot, jó lenne kicsit hasznosabb információkat is kiírni a hibáról:

```
trap
{
    "Hibajelenség: $_"
    "-------- Részletek --------------"
    $_.Exception.GetType().FullName
    $_.InvocationInfo
    "-------- Részletek vége ---------"
    continue
}

Remove-Item c:\nemlétezőfile.txt -ErrorAction Stop
```

Nézzük ennek a kimenetét:

```
PS C:\> .\_munka\powershell2\scripts\trap4.ps1
Hibajelenség: Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
```

```
-------- Részletek --------------
System.Management.Automation.ItemNotFoundException


MyCommand         : Remove-Item
BoundParameters   : {[ErrorAction, Stop], [Path, System.String[]]}
UnboundArguments  : {}
ScriptLineNumber  : 11
OffsetInLine      : 12
HistoryId         : 13
ScriptName        : C:\ munka\powershell2\scripts\trap4.ps1
Line              : Remove-Item c:\nemlétezőfile.txt -ErrorAction Stop
PositionMessage   :
                    At C:\_munka\powershell2\scripts\trap4.ps1:11 char:12
                    + Remove-Item <<<<  c:\nemlétezőfile.txt -ErrorAction Stop
InvocationName    : Remove-Item
PipelineLength    : 1
PipelinePosition  : 1
ExpectingInput    : False
CommandOrigin     : Internal

-------- Részletek vége ---------
```

A „Részletek” első eleme a hibajelenség típusa, azaz az `ExceptionType`. Jelen esetben ez egy:

`System.Management.Automation.ActionPreferenceStopException` típusú hiba. Ha csak bizonyos típusú hibákra szeretném, hogy a `trap`-em reagáljon, akkor ezt a típuselnevezést kell a `trap` paramétereként beírni (lásd nemsokára).

Ezután kiírattam a hibaobjektum `InvocationInfo` paraméterét, ami természetesen önmaga is egy objektum, így annak is számos tulajdonsága van. Ezen tulajdonságok között láthatjuk, hogy milyen parancs végrehajtása során lépett fel a hiba, milyen szkriptben, mi volt a teljes parancssor, stb. Ezek az információk hasznos segítséget nyújtanak a hiba felderítésében.

Nézzük, mit kapunk, ha egy másik hibajelenséget kap a `trap`-em, mondjuk egy nullával való osztást:

```
trap
{
     "Hibajelenség: $_"
     "-------- Részletek --------------"
     $_.Exception.GetType().FullName
     $_.InvocationInfo
     "-------- Részletek vége ---------"
     continue
}

$nulla = 0
8/$nulla
```

Kicsit trükközni kellett, hiszen korábban láttuk, hogy ha közvetlenül ezt írnám. „8/0”, akkor azt már maga a parancsértelmező kiszúrná, és un. „nontrappable”, azaz nem elkapható hibaként kezelné. Ezért elrejtem egy változóba a nullát, és így osztok vele. Nézzük a kimenetet:

```
PS C:\> .\_munka\powershell2\scripts\trap5.ps1
Hibajelenség: Attempted to divide by zero.
-------- Részletek --------------
System.DivideByZeroException
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
MyCommand        :
BoundParameters  : {}
UnboundArguments : {}
ScriptLineNumber : 12
OffsetInLine     : 3
HistoryId        : 14
ScriptName       : C:\_munka\powershell2\scripts\trap5.ps1
Line             : 8/$nulla
PositionMessage  :
                   At C:\ munka\powershell2\scripts\trap5.ps1:12 char:3
                   + 8/ <<<< $nulla
InvocationName   : /
PipelineLength   : 0
PipelinePosition : 0
ExpectingInput   : False
CommandOrigin    : Internal

-------- Részletek vége ---------
```

Látjuk a részletek első sorában, hogy ez egy másfajta hiba, egy `System.DivideByZeroException` típusú. Ezek után, ha én csak az ilyen fajta hibákat akarom lekezelni, akkor a `trap` definícióját kell módosítani:

```
trap [System.DivideByZeroException]
{
     "Hibajelenség: $_"
     "-------- Részletek --------------"
     $_.Exception.GetType().FullName
     $_.InvocationInfo
     "-------- Részletek vége ---------"
     continue
}

$nulla = 0
8/$nulla

Remove-Item nemlétezőfile.txt
```

Ebben a szkriptben a `trap` mellett látható annak a hibatípusnak a neve, amire szeretnénk, ha a `trap`-ünk reagálna. Az összes többi hibafajtát a PowerShell fogja lekezelni. A szkript végén kétfajta hibát okozó műveletet hajtok végre. Nézzük, hogyan reagál ezekre a `trap`:

```
PS C:\> .\_munka\powershell2\scripts\trap6.ps1
Hibajelenség: Attempted to divide by zero.
-------- Részletek --------------
System.DivideByZeroException


MyCommand        :
BoundParameters  : {}
UnboundArguments : {}
ScriptLineNumber : 12
OffsetInLine     : 3
HistoryId        : 15
ScriptName       : C:\_munka\powershell2\scripts\trap6.ps1
Line             : 8/$nulla
```

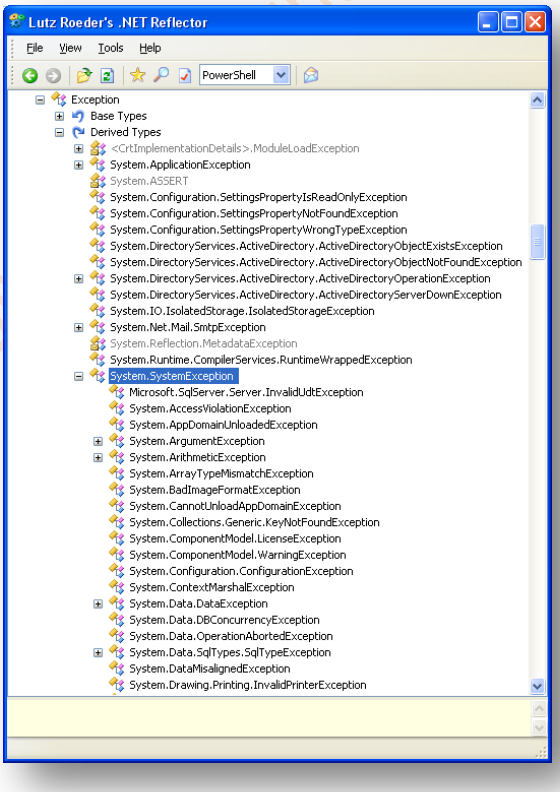
A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PositionMessage  :
                   At C:\_munka\powershell2\scripts\trap6.ps1:12 char:3
                   + 8/ <<<< $nulla
InvocationName   : /
PipelineLength   : 0
PipelinePosition : 0
ExpectingInput   : False
CommandOrigin    : Internal

-------- Részletek vége ---------
Remove-Item : Cannot find path 'C:\nemlétezőfile.txt' because it does not exis
t.
At C:\_munka\powershell2\scripts\trap6.ps1:14 char:12
+ Remove-Item <<<<  nemlétezőfile.txt
    + CategoryInfo          : ObjectNotFound: (C:\nemlétezőfile.txt:String) [
   Remove-Item], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.Remo
   veItemCommand
```

Láthatjuk, hogy csak egyszer fut le az én hibakezelő kódom, a nullával való osztásra. A nem létező fájl törlésekor a PowerShell eredeti hibakezelője futott le.

Az ilyen hibatípusokat legegyszerűbben „megtapasztalás” által lehet felderíteni vagy egy általános trapkezelő rutinnal, vagy az `$error` tömb elemeinek vizsgálatával. Ha valaki előre fel akar készülni a lehetséges hibákra, akkor a .NET Framework `System.Exception` leszármaztatott objektumait kell nézni, például a már említett Reflector programmal:



58. ábra Exception típusok a .NET Frameworkben a Reflector programmal szemlélve

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nagyon sok ilyen hibatípus van, így valószínű a megtapasztalás által könnyebben eredményre jutunk.

Megjegyzés

A `System.Exception` hibaosztályt lehetőleg ne kezeljük, mert nagyon sok hiba tartozik ebbe.

2.3.4.1 Többszintű csapda

Az előbb látott `trap`-eket (csapdákat) a szkriptünk bármelyik szintjén használhatjuk: a globális scope-ban, szkriptek vagy függvények al-scope-jaiban is. Ha egy mélyebb szinten `trap` által lekezelt hibaesemény lép fel, akkor az – beállítható módon – a magasabb szinteken is jelezhet hibát.

Nézzük azt az esetet, amikor van egy szkriptem, benne egy „fő” trap, egy függvény és abban is egy trap, a végén meg a függvény meghívásával nullával osztok:

```
trap
{
     Write-Error "Külső trap"
}

function belső ($osztó)
{
     trap
     {
          Write-Error "Belső trap"
     }

     20/$osztó
}

belső 0
```

Nézzük, hogy mi történik, ha futtatom ezt:

```
PS C:\> .\_munka\powershell2\scripts\nestedtrap.ps1
belső : Belső trap
At C:\ munka\powershell2\scripts\nestedtrap.ps1:18 char:6
+ belső <<<<  0
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorExce
   ption
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorExcepti
   on,belső

Attempted to divide by zero.
At C:\_munka\powershell2\scripts\nestedtrap.ps1:15 char:5
+     20/ <<<< $osztó
    + CategoryInfo          : NotSpecified: (:) [], RuntimeException
    + FullyQualifiedErrorId : RuntimeException
```

Látszik, hogy a belső trap éledt fel, és mivel nem mondtuk, hogy folytathatja, ezért ki is száll a futtatásból a program, megkapjuk még a PowerShell saját hibajelzését is.

Módosítsuk a belső trap-et, mondjuk neki, hogy folytathatja:

```
trap
{
```

```
     Write-Error "Külső trap"
}

function belső ($osztó)
{
     trap
     {
          Write-Error "Belső trap"
          continue
     }

     20/$osztó
}

belső 0
```

Nézzük ennek is a kimenetét:

```
PS C:\> .\_munka\powershell2\scripts\nestedtrap.ps1
belső : Belső trap
At C:\_munka\powershell2\scripts\nestedtrap.ps1:18 char:6
+ belső <<<<  0
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorExce
   ption
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorExcepti
   on,belső
```

Itt ugye az történt, hogy a belső trap egy folytatással (`continue`) zárult le, így a külső környezet már erről a hibáról nem is értesült, azt hiszi, minden rendben.

Zárjuk le akkor a belső trap-et egy `break`-kel:

```
trap
{
     Write-Error "Külső trap"
}

function belső ($osztó)
{
     trap
     {
          Write-Error "Belső trap"
          break
     }

     20/$osztó
}

belső 0
```

Ennek kimenete:

```
PS C:\> .\_munka\powershell2\scripts\nestedtrap.ps1
belső : Belső trap
At C:\_munka\powershell2\scripts\nestedtrap.ps1:17 char:6
+ belső <<<<  0
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorExce
   ption
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorExcepti
   on,belső

C:\_munka\powershell2\scripts\nestedtrap.ps1 : Külső trap
At line:1 char:44
+ .\_munka\powershell2\scripts\nestedtrap.ps1 <<<<
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorExce
   ption
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorExcepti
   on,nestedtrap.ps1

Attempted to divide by zero.
At C:\_munka\powershell2\scripts\nestedtrap.ps1:14 char:5
+     20/ <<<< $osztó
    + CategoryInfo          : NotSpecified: (:) [], RuntimeException
    + FullyQualifiedErrorId : RuntimeException
```

Itt már mindkét szint trapje megjelenik, hiszen a belső trap rendhagyó módon, egy megtöréssel (`break`) ért véget, ami kívülről nézve is hibát jelez. Ezért a külső trap is felélesedik és az is jelzi a hibát.

Mindezek figyelembevételével el lehet dönteni, hogy hol érdemes, és a működés szempontjából hol kell trapet elhelyezni.

2.3.4.2 Dobom és elkapom

Gyakori hibakezelési feladat, hogy a függvényünk, szkriptünk használata során nem ad meg valaki valamilyen kötelező paramétert. Ennek kezelésére már láttuk a `throw` kulcsszót az *1.7 Függvények* fejezet *1.7.2.3 Hibajelzés* alfejezetében, de ennek hatására megjelenő hibajelzés nem annyira szép, mert nem csak az általunk megadott szöveg kerül kiírásra, hanem egyéb szövegek is. Adódik az ötlet, hogy akkor kapjuk el egy `trap`-pel az általunk `throw`-val dobott hibajelzést, és szabjuk testre a `trap` kódjában a hibajelzést.

Készítettem egy újabb szkriptet az előzőek alapján:

```
trap
{
     "Hibajelenség: $_"
     "-------- Részletek --------------"
     $_.Exception.GetType().FullName
     $_.InvocationInfo
     "-------- Részletek vége ---------"
     continue
}

function dupla ($v = $(throw "Adj meg paramétert!"))
{
     $v*2
}

dupla
```

A `trap` része nem nagyon változott, és ott van mellette a korábban már látott duplázó függvényem, hibajelzéssel a paraméterhiány esetére. A szkript legvégén meghívom magát a dupla függvényt mindenféle paraméter nélkül. Mindennek ez lesz az eredménye:

```
PS C:\> .\_munka\powershell2\scripts\trap7.ps1
Hibajelenség: Adj meg paramétert!
```

```
-------- Részletek --------------
System.Management.Automation.RuntimeException


MyCommand         :
BoundParameters   : {}
UnboundArguments  : {}
ScriptLineNumber  : 11
OffsetInLine      : 29
HistoryId         : 24
ScriptName        : C:\ munka\powershell2\scripts\trap7.ps1
Line              : function dupla ($v = $(throw "Adj meg paramétert!"))
PositionMessage   :
                    At C:\_munka\powershell2\scripts\trap7.ps1:11 char:29
                    + function dupla ($v = $(throw <<<<  "Adj meg paramétert!")
                    )
InvocationName    : throw
PipelineLength    : 0
PipelinePosition  : 0
ExpectingInput    : False
CommandOrigin     : Internal

-------- Részletek vége ---------
```

Tényleg működik a `throw` elkapása, csak az én általam megadott hibajelzés jelent meg, viszont elég általánosnak tűnik a hiba típusa:

```
System.Management.Automation.RuntimeException.
```

Ez nem túl specifikus, hanem általános futási hiba. Viszont lehetünk specifikusabbak, adjunk a `throw`-val egy értelmesebb hibaobjektumot a `trap` számára, azaz ne csak egy egyszerű szöveget adjunk neki, hanem egy megfelelő `Exception` objektumot!

```
trap
{
     "Hibajelenség: $_"
     "-------- Részletek --------------"
     $_.Exception.GetType().FullName
     $_.InvocationInfo
     "-------- Részletek vége ---------"
     continue
}

function dupla ($v = $(throw `
     (New-Object System.ArgumentException `
          -arg "Adjál meg valamit, amit duplázni lehet!")))
{
     $v*2
}

dupla
```

En úgy gondoltam, hogy az ilyen jellegű hiba leírására a `System.ArgumentException` típus lesz a legjobb, így ennek egy objektumát hozom létre a `throw` paramétereként. Ráadásul ennek az objektumnak a konstruktora argumentumot is képes fogadni, az általam megadott hibaleírást szöveggel. Így már nagyon elegáns és hiba specifikus lesz a kimenete is:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> .\_munka\powershell2\scripts\trap8.ps1
Hibajelenség: Adjál meg valamit, amit duplázni lehet!
-------- Részletek --------------
System.ArgumentException


MyCommand        :
BoundParameters  : {}
UnboundArguments : {}
ScriptLineNumber : 11
OffsetInLine     : 29
HistoryId        : 25
ScriptName       : C:\_munka\powershell2\scripts\trap8.ps1
Line             : function dupla ($v = $(throw `
PositionMessage  :
                   At C:\_munka\powershell2\scripts\trap8.ps1:11 char:29
                   + function dupla ($v = $(throw <<<<  `
InvocationName   : throw
PipelineLength   : 0
PipelinePosition : 0
ExpectingInput   : False
CommandOrigin    : Internal

-------- Részletek vége ---------
```

Így már specifikusabb `trap`-et is lehet készíteni, ami csak erre a hibatípusra éled fel.

2.3.4.3 Try, Catch, Finally

A PowerShell 2.0-ban új hibakezelési lehetőség is megjelent, amit a `Try`, `Catch` és `Finally` kulcsszavakkal tudunk munkára bírni. A `Try` szekcióba tegyük azt a szkriptrészt, ahol előfordulhat az a hiba, amit kezelni akarunk. Fontos, hogy csak megszakító hibákat kezel le ez a lehetőség is, márpedig alaphelyzetben a PowerShell cmdletek nem megszakító hibát eredményeznek. Ezt mindjárt egy példán is megnézzük. Legyen egy olyan egyszerű függvényünk, ami töröl egy paraméterként megadott elérési úton található fájlt:

```
function deletefile ($path)
{
    try {remove-item -Path $path}
    catch {"Nincs ilyen fájl!"}
    finally {"Itt a vég!"}
}
```

Nézzük, mit ad ez eredményül:

```
[93] PS C:\> deletefile c:\nincs.txt
Remove-Item : Cannot find path 'C:\nincs.txt' because it does not exist.
At line:3 char:21
+     try {remove-item <<<<  -Path $path}
    + CategoryInfo          : ObjectNotFound: (C:\nincs.txt:String) [Remove-I
   tem], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.Remo
   veItemCommand

Itt a vég!
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ez a standard PowerShelles hibajelzés, és nem az, amit vártunk. Ennek oka az, hogy amint említettem, a PowerShell cmdletek hibái legtöbbször nem megszakító jellegűek, és a `Try-Catch` csak megszakító hibára működik. Hogyan lehet ezt megszakítóvá alakítani? Állítsuk át az `$ErrorActionPreference` változót `Stop` értékűre, vagy még inkább az adott cmdletnél az `-ErrorAction` paramétert:

```
function deletefile ($path)
{
    try {remove-item -Path $path -ErrorAction Stop}
    catch {"Nincs ilyen fájl!"}
    finally {"Itt a vég!"}
}
```

Nézzük, most hogyan fut:

```
[95] PS C:\> deletefile c:\nincs.txt
Nincs ilyen fájl!
Itt a vég!
```

Na ez az, amire számítottunk. Látható, hogy a „`Finally`” rész mindig lefut, még akkor is, ha nincs hiba:

```
[97] PS C:\> deletefile c:\a.txt
Itt a vég!
```

Természetesen ez többet is tud ennél. A `Catch` részt csak bizonyos hibákra is érzékennyé tehetjük:

```
function dl ($url, $cél)
{
    try
    {
        $wc = new-object System.Net.WebClient
        $wc.DownloadString($url)  > $cél
    }
    catch [System.Management.Automation.MethodException],
        [System.Management.Automation.DriveNotFoundException]
    {
        $error[0].Exception.GetType().FullName
        "Nem tudok letölteni $url helyről $cél helyre."
    }
    catch
    {
        $error[0].Exception.GetType().FullName
        "Valami egyéb hiba van."
    }
}
```

Ebben az egyszerű példában a `dl` függvény az `$url` helyről letölti a weboldalt a `$cél` helyre. Nézzük hogyan reagál ez a különböző hibákra:

```
PS C:\> dl http://www.iqjb.hu c:\nincs\yyy.txt
System.IO.DirectoryNotFoundException
Valami egyéb hiba van.
```

A fenti esetben a célkönyvtár nem létezik. Ennek hibatípusa:

`System.IO.DirectoryNotFoundException`,

amit az első `Catch` részben nem szűrtem, így az „bezuhant” a második `Catch`-be, azaz az ottani hibaszöveget írta ki.

```
PS C:\> dl http://www.iqjb.hu x:\yyy.txt
System.Management.Automation.DriveNotFoundException
Nem tudok letölteni http://www.iqjb.hu helyről x:\yyy.txt helyre.
```

Ebben az esetben a cél meghajtó nem létezik, ez az első `Catch` blokkban van lekezelve, ahogy a következő eset is:

```
PS C:\> dl http://www.iqjb.xx c:\_mobil\yyy.txt
System.Management.Automation.MethodInvocationException
Nem tudok letölteni http://www.iqjb.xx helyről c:\_mobil\yyy.txt helyre.
```

Látható tehát, hogy átláthatóan, követhetőbben lehet lekezelni a különböző hibajelenségeket.

Megjegyzés

Sajnos a cmdletek Stop módon való megállítása egy kicsit megkavarja a Try-Catch működését. Nézzünk erre egy példát:

```
function deletefile ($path)
{
    try {remove-item -Path $path -ErrorAction Stop}
    catch {$error[0].Exception.GetType().FullName; "Nincs ilyen!"}
}
```

Ha ezt futtatom egy nem létező fájlra, akkor a már korábbal látotthoz hasonló kimenetet kapunk:

```
PS C:\> Deletefile c:\yyyy.txt
System.Management.Automation.ItemNotFoundException
Nincs ilyen!
```

Látható, hogy milyen a hiba típusa. Ha erre akarok szűrni, akkor nézzük, hogy mi történik:

```
function deletefile ($path)
{
    try {remove-item -Path $path -ErrorAction Stop}
    catch [System.Management.Automation.ItemNotFoundException]
     {$error[0].Exception.GetType().FullName; "Nincs ilyen!"}
}
```

Ennek ugyanazt kellene adnia kimenetként, de nem:

```
PS C:\> Deletefile c:\yyyy.txt
Cannot find path 'C:\yyyy.txt' because it does not exist.
At :line:4 char:21
+     try {remove-item  <<<< -Path $path -ErrorAction Stop}
```

Na ezt nem értem teljesen, illetve valami olyasmi történik, mintha egy burkolt `Throw` történne valahol a `Catch` belsejében… Ennek kezelésére a következő fejezetben teszek javaslatot.

Végezetül pár szempont, hogy mikor érdemes `trap`-et, és mikor `Try-Catch`-et alkalmazni:

- `Trap`: saját scope-ja van, viszont globális a hatása, azaz bárhol is van a hiba, azt a `trap` elkapja. Nem lehet a hibát „továbbdobni”, azaz egy üres `Throw` kifejezés egy speciális, új hibát generál, aminek a típusa „`ScriptHalted`”.
- `Try/Catch`: nincs külön scope-ja, szelektíven, csak a `Try` „testében” levő hibákra reagál. A hiba továbbdobható egy üres `Throw` kifejezéssel.

2.3.5 Nem megszakító hibák kezelése függvényeinkben

A `trap` tárgyalásának elején már láthattuk, hogy azzal hibakezelést csak megszakító hibákra tudunk készíteni, legfeljebb a `trap` végére tett `continue` kulcsszóval átalakítjuk a hibát nem megszakítóra. De vajon hogyan tudunk nem megszakító hibákat kezelni? Erre nincs külön kulcsszó, a többi PowerShelles nyelvi eszközzel kell ezt megoldani.

Az egyik ilyen helyzet, hogy a nem megszakító hibát nem akarom csak azért megszakítóvá alakítani, hogy aztán egy `continue`-val elnyomhassam a hibajelzést, hanem eleve nem akarok látni semmilyen hibainformációt a konzolon. Ehhez azt kell tudni, hogy külön outputja van a hibajelzéseknek, így lehet olyat is csinálni, hogy csak ezeket a hibajelzéseket küldjük át egy fájlba, vagy akár a semmibe. Nézzünk erre példát:

```
[43] PS C:\> Remove-Item nemlétezőfájl.txt 2> $null
```

Ez tehát a „`2>`” operátor, ami a 2-es számú outputot, a hibajelzések outputját irányítja a semmibe. (Amúgy „`1>`” nem létezik, csak a kettessel kezdődő változat.)

Ha pont ellentétes lenne a feladat, azaz valami egyedi, saját hibajelzést szeretnénk adni a „gyári” jelzés helyett, arra külön `write-...` kezdetű cmdletek állnak rendelkezésünkre. Ezekkel szintén nem a „normál” outputra, hanem erre a hibajelzések számára elkülönített outputra küldhetjük a jelzést.

```
[67] PS C:\powershell2\scripts> filter bulk-remove
>> {
>>     if (-not (Test-Path $_))
>>         { Write-Warning "Nincs ilyen file: $_!" }
>>     else
>>         { Remove-Item $_ }
>> }
>>
```

A fenti `filter` kifejezéssel definiálok egy olyan fájltörlő függvényt, amely először ellenőrzi, hagy van-e ott ténylegesen fájl, amit megadott a felhasználó, és ha nincs, akkor nem csúnya hibajelzést ad, hanem csak egy sárga színű figyelmeztetést a `write-warning` cmdlet segítségével.

```
[68] PS C:\powershell2\scripts> "nincs.txt", "file.txt" | bulk-remove
WARNING: Nincs ilyen file: nincs.txt!
```

Ez azért is jó, mert ez nem kerül be a `$error` tömbbe, így nem „terheli” azt. Természetesen lehetne piros feliratú `write-error`-t is használni, ekkor olyan hibajelzést kapunk, ami bekerül az `$error` tömbbe.

Nem csak ilyen következményei vannak a `write-error` és `write-warning` használatának. Ha átállítjuk a következő automatikus változókat, akkor szabályozhatjuk függvényünk, szkriptünk futását is:

```
[79] PS C:\powershell2\scripts> $WarningPreference
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Continue
[80] PS C:\powershell2\scripts> $ErrorActionPreference
Continue
```

Természetesen a `write-error` még profibbá tehető az `Exception` objektumok használatával:

```
filter bulk-remove
{
     if (-not (Test-Path $_))
           { Write-Error -Exception `
                 (New-Object System.IO.FileNotFoundException `
                       -arg "Nincs ilyen file!",$_) }
     else
           {
                 Remove-Item $_
           }
}
```

És ennek kimenete hiba esetén:

```
[2] PS C:\> "semmi.txt" | bulk-remove
bulk-remove : Nincs ilyen file!
At line:1 char:26
+ "semmi.txt" | bulk-remove <<<<
    + CategoryInfo          : NotSpecified: (:) [Write-Error], FileNotFoundEx
   ception
    + FullyQualifiedErrorId : System.IO.FileNotFoundException,bulk-remove
[3] PS C:\> $error[0].exception.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     FileNotFoundException                    System.IO.IOExce...
```

A [83]-as sorban láthatjuk, hogy ez a hiba már „igazi” `FileNotFoundException` lett.

A másik lehetőség a `$?` automatikus változó vizsgálatával történő hibakezelés. Ez a változó az előző kifejezés végrehajtásának eredményét tartalmazza.

```
filter bulk-remove
{
    remove-item $_ -ErrorAction silentlycontinue
     if (!$?)
           { $Error[0].categoryinfo.reason}
}
```

A fenti, módosított filterben maga a `remove-item` hibajelzése el van nyomva (`silentlycontinue`), ennek ellenére azért a hibakód belekerül az `$error` változóba, illetve a hiba ténye miatt `$false` lesz a `$?` változó, így reagálhatunk rá. Az `if` szerkezet szkriptblokkjára akkor adódik rá a vezérlés, ha hibát adott az aktuális elem eltávolítása.

2.3.6 Hibakeresés

Akármennyire sok lehetőségünk van a hibák jelzésére (`write-warning`, `write-error` és `trap`), kezelésére mégsem minden esetben elég ez. Főleg akkor, ha a szkriptünk a maga módján jól működik, csak

éppen mi rossz programot írtunk. Ilyenkor van szükség a „dibággolásra”, „bogártalanításra”, azaz a hibakeresésre. Erre is lehetőséget biztosít a PowerShell, bár valószínű egy bizonyos bonyolultság felett már inkább használunk erre a célra valamilyen grafikus szkriptszerkesztőt, mint például a PowerShell Integrated Scripting Environmentjét vagy a PowerGUI Script Editorát.

2.3.6.1 Státusjelzés (write-verbose, write-debug)

Az egyik „bogártalanítási” módszer, hogy jól teletűzdeljük a programunkat kiíratási parancsokkal, amelyek segítségével a változók aktuális állapotát írhatjuk ki, illetve jelezhetjük, hogy éppen milyen ágon fut a programon. Erre használhatjuk akár a `write-host` cmdletet, csak az a baj ezzel, hogy a végleges, kijavított, tesztelt programból elég nehéz kiszedegetni ezeket. Ezért találtak ki másfajta kiíratási lehetőségeket is: a `write-verbose` és a `write-debug` cmdleteket.

Nézzünk erre példát. Van egy faktoriálist számoló szkriptem, amit jól teletűzdelek `write-verbose` helyzetjelentésekkel:

```
Write-Verbose "Eleje"
$a = 10; $result = 1
Write-Verbose "Közepe $a"
for($i=1; $i -le $a; $i++)
     {
           $result = $result * $i
           Write-Verbose "`$i = $i, `$result = $result"
     }
Write-Verbose "Vége"
$result
```

Nézzük, mi történik, ha futtatom:

```
[7] PS C:\powershell2\scripts> .\verbose.ps1
3628800
```

Nem sok mindent látunk ezen, mintha ott sem lennének a `write-verbose` parancsok. Merthogy a `write-verbose` hatását egy globális változóval, a `$VerbosePreference`-szel lehet ki- és bekapcsolni:

```
[9] PS C:\powershell2\scripts> $verbosepreference = "continue"
[10] PS C:\powershell2\scripts> .\verbose.ps1
VERBOSE: Eleje
VERBOSE: Közepe 10
VERBOSE: $i = 1, $result = 1
VERBOSE: $i = 2, $result = 2
VERBOSE: $i = 3, $result = 6
VERBOSE: $i = 4, $result = 24
VERBOSE: $i = 5, $result = 120
VERBOSE: $i = 6, $result = 720
VERBOSE: $i = 7, $result = 5040
VERBOSE: $i = 8, $result = 40320
VERBOSE: $i = 9, $result = 362880
VERBOSE: $i = 10, $result = 3628800
VERBOSE: Vége
3628800
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nyomtatásban esetleg nem látszik, de a `VERBOSE:` kimenetek szép sárgák, hasonlóan a „Warning” jelzésekhez. Ennek további értékei lehetnek – hasonlóan az `$ErrorActionPreference` változóhoz – `SilentlyContinue` (alapérték), `Inquire`, `Stop`.

Ugyanígy működik a `write-debug` cmdlet is, csak az ő hatását a `$DebugPreference` változó szabályozza.

Azaz van két, egymástól függetlenül használható kiíratható cmdletünk, amelyekkel a szkriptjeink futásának státusát írathatjuk ki a fejlesztés során, majd ha készen vagyunk és meggyőződtünk, hogy minden jól működik, akkor anélkül, hogy a szkriptből ki kellene szedegetni ezeket a státusjelzéseket, egy globális változó beállításával ezek hatását ki tudjuk kapcsolni.

2.3.6.2 Lépésenkénti végrehajtás és szigorú változókezelés (set-psdebug)

A másik hibaűző módszer, ha lépésenként hajtatjuk végre a szkriptünket és így jobban mg tudjuk figyelni, hogy merre jár. Azaz nem kell minden sor után egy `write-debug` sort beiktatni. Erre is lehetőséget biztosít a PowerShell a `Set-PSDebug` cmdlettel. Nézzük meg a szintaxisát:

```
[22] PS C:\powershell2\scripts> (get-help set-psdebug).syntax

Set-PSDebug [-Off] [<CommonParameters>]
Set-PSDebug [-Step] [-Strict] [-Trace <int>] [<CommonParameters>]
```

A második változatban kapcsolhatjuk be a hibakeresési üzemmódot. A `-trace` paraméter lehetséges értékei:

- 0: kikapcsoljuk a nyomkövetést
- 1: csak a végrehajtás során az aktuális sor számát jelzi
- 2: a sorok száma mellett a változók értékadását is jelzi

Az előző szkriptet fogom használni, kicsit átalakítva:

```
$a = 3; $result = 1
for($i=1; $i -le $a; $i++)
     {
           $result = $result * $i
     }
$result
```

És akkor nézzük a `-Trace 1` opcióval a futását:

```
PS C:\> Set-PSDebug -Trace 1
DEBUG:     2+          $foundSuggestion = <<<<  $false
DEBUG:     4+          if <<<< ($lastError -and
DEBUG:    15+          $foundSuggestion <<<<
```

Érdekes módon, saját magát is „megnyomkövette”!

```
PS C:\> .\_munka\powershell2\scripts\psdebug.ps1
DEBUG:     1+  <<<< .\_munka\powershell2\scripts\psdebug.ps1
DEBUG:     1+ $a = <<<<  3; $result = 1
DEBUG:     1+ $a = 3; $result = <<<<  1
DEBUG:     2+ for <<<< ($i=1; $i -le $a; $i++)
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:     6+ $result <<<<
6
DEBUG:     2+          $foundSuggestion = <<<<  $false
DEBUG:     4+          if <<<< ($lastError -and
DEBUG:    15+          $foundSuggestion <<<<
```

Láthatjuk, hogy szépen kiírja az éppen végrehajtott sorok számát, meg az ottani parancssor tartalmát is. Nézzük a `-Trace 2` opció hatását:

```
PS C:\> Set-PSDebug -Trace 2
DEBUG:     1+  <<<< Set-PSDebug -Trace 2
DEBUG:     2+          $foundSuggestion = <<<<  $false
DEBUG:      ! SET $foundSuggestion = 'False'.
DEBUG:     4+          if <<<< ($lastError -and
DEBUG:    15+          $foundSuggestion <<<<
PS C:\> .\_munka\powershell2\scripts\psdebug.ps1
DEBUG:     1+  <<<< .\_munka\powershell2\scripts\psdebug.ps1
DEBUG:      ! CALL function 'psdebug.ps1'  (defined in file
'C:\_munka\powershell2\scripts\psdebug.ps1')
DEBUG:     1+ $a = <<<<  3; $result = 1
DEBUG:      ! SET $a = '3'.
DEBUG:     1+ $a = 3; $result = <<<<  1
DEBUG:      ! SET $result = '1'.
DEBUG:     2+ for <<<< ($i=1; $i -le $a; $i++)
DEBUG:      ! SET $i = '1'.
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:      ! SET $result = '1'.
DEBUG:      ! SET $i = '2'.
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:      ! SET $result = '2'.
DEBUG:      ! SET $i = '3'.
DEBUG:     4+    $result = <<<<  $result * $i
DEBUG:      ! SET $result = '6'.
DEBUG:      ! SET $i = '4'.
DEBUG:     6+ $result <<<<
6
DEBUG:     2+          $foundSuggestion = <<<<  $false
DEBUG:      ! SET $foundSuggestion = 'False'.
DEBUG:     4+          if <<<< ($lastError -and
DEBUG:    15+          $foundSuggestion <<<<
```

Láthatjuk, hogy az előzőek mellett még a szkripthívást is látjuk, és minden egyes változó értékadását is az új értékekkel együtt. Ha még a `-step` paramétert is használjuk, akkor minden sor végrehajtására külön rákérdez:

```
[37] PS C:\powershell2\scripts> Set-PSDebug -Trace 1 -step
[38] PS C:\powershell2\scripts> .\psdebug.ps1

Continue with this operation?
   1+ .\psdebug.ps1
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
DEBUG:    1+ .\psdebug.ps1

Continue with this operation?
```

```
   1+ $a = 3; $result = 1
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
DEBUG:    1+ $a = 3; $result = 1

Continue with this operation?
   1+ $a = 3; $result = 1
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):s
```

Ekkor választhatunk, hogy tovább lépünk egy sorral (`Yes`), folytatjuk a futtatást végig (`Yes to All`), megszakítjuk a futtatást (`No` és `No to All`). A `Suspend` „némán” állítja meg a futtatást, nem lesz hibajelzés a rendkívüli programmegszakításról.

A `-strict` kapcsoló használata után csak olyan változókra hivatkozhatunk, amelynek korábban adtunk kezdőértéket:

```
[41] PS C:\powershell2\scripts> Set-PSDebug -strict
[42] PS C:\powershell2\scripts> $c=1
[43] PS C:\powershell2\scripts> $c
1
[44] PS C:\powershell2\scripts> $d
The variable '$d' cannot be retrieved because it has not been set.
At line:1 char:3
+ $d <<<<
    + CategoryInfo          : InvalidOperation: (d:Token) [], RuntimeExceptio
   n
    + FullyQualifiedErrorId : VariableIsUndefined
```

A fenti példában a `$c` értékadás után lekérdezhető volt, szemben a `$d`-vel, amire hibajelzést kaptunk, hiszen nem adtam neki kezdőértéket.

Ha már nincs szükségünk a `Set-PSDebug`-gal beállított hibafelderítő üzemmódokra, akkor az `-Off` kapcsolóval kapcsolhatjuk ki ezekez:

```
[45] PS C:\powershell2\scripts> Set-PSDebug -off
[46] PS C:\powershell2\scripts> $d
[47] PS C:\powershell2\scripts>
```

Itt látható, hogy már nem okozott hibajelzést az értékadás nélküli `$d`-re való hivatkozás.

A `Set-PSDebug` egy globális hatású cmdlet, azaz beállítása után minden futtatási környezet (scope) azonos módon „szigorú” lesz. A PowerShell 2.0-ban egy új cmdletünk is van, a `Set-StrictMode`. Ez már csak az adott futtatási környezetre és alkörnyezetekre hat, így sokkal könnyebb az éppen hibafelderítés alatt álló részekre beállítani a szigorúságot. A `-Version 1` paraméterrel hasonlóan működik, mint a korábban látott szigorú üzemmód. De a `-Version 2`-vel még szigorúbb lesz:

```
PS C:\Users\tibi> Set-StrictMode -Version 2
PS C:\Users\tibi> $a="hkh"
PS C:\Users\tibi> $a.yyy
Property 'yyy' cannot be found on this object. Make sure that it exists.
At line:1 char:4
+ $a. <<<< yyy
    + CategoryInfo          : InvalidOperation: (.:OperatorToken) [], Runtime
   Exception
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    + FullyQualifiedErrorId : PropertyNotFoundStrict
```

Az első sorban beállítottam ezt a fajta szigorúságot a PowerShell 2.0 kompatibilis szintre. Ezután az `$a` sztringemnek nem létező `yyy` tulajdonságára történő hivatkozás hibát okozott. Ez nagyon nagy szolgálat a hibakeresésben az 1.0 verzióhoz képest!

Ezen kívül van még egy szolgáltatása ennek a `Set-StrictMode` beállításnak. Ez pedig a hagyományos, Visual Basic-szerű függvényhívás szintaxisra való figyelmeztetés:

```
PS C:\Users\tibi> function valami ($x,$y){$x+$y}
PS C:\Users\tibi> valami(1,2)
The function or command was called as if it were a method. Parameters should b
e separated by spaces. For information about parameters, see the about_Paramet
ers Help topic.
At line:1 char:7
+ valami <<<< (1,2)
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : StrictModeFunctionCallWithParens
```

Itt a nagyon egyszerű függvényemet úgy próbáltam hívni, mintha például egy Excel függvény, vagy egy metódus lenne: nincs szóköz a függvény neve és a zárójel között, és eleve zárójel van. Ez valószínűsíthetően hiba, így megkapjuk a hibajelzést.

2.3.6.3 Ássunk még mélyebbre (Trace-Command)

Ha még ez sem lenne elég, akkor még mélyebbre áshatunk a `Trace-Command` cmdlet segítségével. Igazából ez nem biztos, hogy a rendszergazdák mindennapos eszköze lesz, inkább azon fejlesztők tudnak profitálni használatából, akik cmdleteket fejlesztenek.

Nézzük a szintaxisát:

```
[5] PS I:\>(get-help trace-command).syntax

Trace-Command [-Command] <string> [-ArgumentList <Object[]>] [-Name] <string[]
> [[-Option] {None | Constructor | Dispose | Finalizer | Method | Property | D
elegates | Events | Exception | Lock | Error | Errors | Warning | Verbose | Wr
iteLine | Data | Scope | ExecutionFlow | Assert | All}] [-Debugger] [-FilePath
 <string>] [-Force] [-InputObject <psobject>] [-ListenerOption {None | Logical
OperationStack | DateTime | Timestamp | ProcessId | ThreadId | Callstack}] [-P
SHost] [<CommonParameters>]
Trace-Command [-Expression] <scriptblock> [-Name] <string[]> [[-Option] {None
| Constructor | Dispose | Finalizer | Method | Property | Delegates | Events |
 Exception | Lock | Error | Errors | Warning | Verbose | WriteLine | Data | Sc
ope | ExecutionFlow | Assert | All}] [-Debugger] [-FilePath <string>] [-Force]
 [-InputObject <psobject>] [-ListenerOption {None | LogicalOperationStack | Da
teTime | Timestamp | ProcessId | ThreadId | Callstack}] [-PSHost] [<CommonPara
meters>]
```

Huh, nem túl egyszerű! Az látható, hogy alapvetően kétfajta dolgot lehet vele vizsgálni: parancsokat és kifejezéseket. Alapvetően mi (rendszergazdák) két esetben használhatjuk: az első eset, amikor a parancsok, kifejezések meghívásának körülményeit szeretnénk tisztázni, a másik, amikor a paraméterek átadásának körülményeit. Nézzünk példát az elsőre:

```
PS C:\> Trace-Command commanddiscovery {dir c:\} -pshost
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
DEBUG: CommandDiscovery Information: 0 : Looking up command: dir
DEBUG: CommandDiscovery Information: 0 : Alias found: dir  Get-ChildItem
DEBUG: CommandDiscovery Information: 0 : Cmdlet found: Get-ChildItem
Microsoft.PowerShell.Commands.GetChildItemCommand,
Microsoft.PowerShell.Commands.Management, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35


    Directory: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       2009.07.14.      5:20            PerfLogs
d-r--       2010.02.13.     15:39            Program Files
d-r--       2010.02.25.     14:31            Program Files (x86)
…
```

Itt a `trace-command` cmdletet a „`commanddiscovery`” névvel hívtam meg. Mögötte ott van az a kifejezés szkriptblokk formában, amit elemeztetni szeretnék, majd megadom, hogy az eredményt a konzolra írja ki.

Az elemzés felderíti, hogy a PowerShell a „`dir`” álnevet hogyan párosítja a `Get-ChildItem` cmdlettel, megnézi, hogy nem tréfáltuk-e meg, és nem készítettünk-e ilyen névvel valamilyen függvényt vagy szűrőt (filter). Ha nem, akkor kiírja, hogy melyik PSSnapIn-ben van ez a cmdlet definiálva és utána már csak a futásának eredményét látjuk. A `trace-command` ezzel az üzemmódjával tehát akkor jön jól nekünk, ha mindenféle aliast, függvényt, szkriptet és egyebeket definiálunk, gyanítjuk, hogy esetleg többször is ugyanolyan névvel, és amikor futtatunk valamit, nem értjük, hogy mi is fut le valójában.

Nézzük a paraméterátadás felderítését:

```
PS C:\> Trace-Command parameterbinding {get-process powershell} -pshost
DEBUG: ParameterBinding Information: 0 : BIND NAMED cmd line args
[Get-Process]
DEBUG: ParameterBinding Information: 0 : BIND POSITIONAL cmd line args
[Get-Process]
DEBUG: ParameterBinding Information: 0 :     BIND arg [powershell] to
parameter [Name]
DEBUG: ParameterBinding Information: 0 :         Binding collection parameter
Name: argument type [String], parameter type [System.String[]], collection
type Array, element type [System.String], no coerceElementType
DEBUG: ParameterBinding Information: 0 :         Creating array with element
type [System.String] and 1 elements
DEBUG: ParameterBinding Information: 0 :         Argument type String is not
IList, treating this as scalar
DEBUG: ParameterBinding Information: 0 :         Adding scalar element of type
 String to array position 0
DEBUG: ParameterBinding Information: 0 :         Executing VALIDATION
metadata: [System.Management.Automation.ValidateNotNullOrEmptyAttribute]
DEBUG: ParameterBinding Information: 0 :         BIND arg [System.String[]] to
 param [Name] SUCCESSFUL
DEBUG: ParameterBinding Information: 0 : MANDATORY PARAMETER CHECK on cmdlet
[Get-Process]
DEBUG: ParameterBinding Information: 0 : CALLING BeginProcessing
DEBUG: ParameterBinding Information: 0 : CALLING EndProcessing

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    382      24    85184      90136   595     4,52   2912 powershell
```

Itt a `trace-command` név paramétere „`parameterbinding`”, mögötte megint a vizsgálandó kifejezés, majd az, hogy a konzolra küldje az eredményt. Látjuk, hogy a „PowerShell” karaktersorozatot a „`Name`” paraméterhez fűzte. Ez nekünk természetesnek tűnik, de azért ez nem ilyen triviális. Nézzük, vajon mit is vár a `get-process` „`Name`” paraméter gyanánt:

```
[10] PS I:\>(get-help get-process).syntax

Get-Process [[-name] <string[]>] [<CommonParameters>]
```

Láthatjuk a súgóban, hogy egy sztringtömböt vár a `get-process`. Az elemzés további része annak lépéseit mutatja, hogy hogyan csinál a PowerShell a sima sztringből sztringtömböt, és hogy ezzel a paraméterrel sikeresen vette a cmdlet mindhárom feldolgozási fázisát is, a `begin` és az `end` részt is.

A másik eset, hogy nem értjük, hogy egy cmdletnek vagy szkriptnek átadott paraméter miért nem jó, miért nem úgy értelmezi a kifejezés, mint ahogy mi szeretnénk. Ez utóbbi demonstrálására nézzünk egy példát, amelyben annak nézünk utána, hogy ha egy gyűjteményt adunk át a `get-member`-nek, akkor az vajon miért a gyűjtemény elemeinek tagjellemzőit adja vissza, miért nem a gyűjteményét:

```
PS C:\> Trace-Command parameterbinding {1, "kettő" | gm} -pshost
DEBUG: ParameterBinding Information: 0 : BIND NAMED cmd line args [Get-Member]
DEBUG: ParameterBinding Information: 0 : BIND POSITIONAL cmd line args
[Get-Member]
DEBUG: ParameterBinding Information: 0 : MANDATORY PARAMETER CHECK on cmdlet
[Get-Member]
DEBUG: ParameterBinding Information: 0 : CALLING BeginProcessing
DEBUG: ParameterBinding Information: 0 : BIND PIPELINE object to parameters:
[Get-Member]
DEBUG: ParameterBinding Information: 0 :     PIPELINE object TYPE =
[System.Int32]
DEBUG: ParameterBinding Information: 0 :     RESTORING pipeline parameter's
original values
DEBUG: ParameterBinding Information: 0 :     Parameter [InputObject] PIPELINE
INPUT ValueFromPipeline NO COERCION
DEBUG: ParameterBinding Information: 0 :     BIND arg [1] to parameter
[InputObject]
DEBUG: ParameterBinding Information: 0 :         BIND arg [1] to param
[InputObject] SUCCESSFUL
DEBUG: ParameterBinding Information: 0 : MANDATORY PARAMETER CHECK on cmdlet
[Get-Member]
DEBUG: ParameterBinding Information: 0 : BIND PIPELINE object to parameters:
[Get-Member]
DEBUG: ParameterBinding Information: 0 :     PIPELINE object TYPE =
[System.String]
DEBUG: ParameterBinding Information: 0 :     RESTORING pipeline parameter's
original values
DEBUG: ParameterBinding Information: 0 :     Parameter [InputObject] PIPELINE
INPUT ValueFromPipeline NO COERCION
DEBUG: ParameterBinding Information: 0 :     BIND arg [kettő] to parameter
[InputObject]
DEBUG: ParameterBinding Information: 0 :         BIND arg [kettő] to param
[InputObject] SUCCESSFUL
DEBUG: ParameterBinding Information: 0 : MANDATORY PARAMETER CHECK on cmdlet
[Get-Member]
DEBUG: ParameterBinding Information: 0 : CALLING EndProcessing
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
…
```

Láthatjuk, hogy a `get-member` hogyan dolgozza fel a csővezetéken érkező különböző objektumokat, hogyan emelődnek át az argumentumok `InputObject`-té. Elsőként a parancsértelmező megnézi, hogy vannak-e név szerinti paraméterek a parancssorban, majd ellenőrzi a pozíció alapján történő paraméterátadást. Ezután ellenőrzi, hogy a kötelező paraméterek meg vannak-e adva, a `get-member`-nek nincs ilyenje. Mindezek után veszi észre, hogy csővezetékben adom át a paramétereket, így annak kezelése történik meg, először az „`1`” lesz behelyettesítve „`InputObject`” paraméterként, majd a „`kettő`”.

2.3.6.4 Megszakítási pontok kezelése a konzolon

A PowerShell 2.0-ban jelent meg beépített, nyelvi szinten definiált (azaz nem a szerkesztő eszköz vagy a konzol szintjén definiált) megszakítási pontok létrehozásának lehetősége. Ilyen megszakítási pontokat a `PSBreakPoint` főnévvel rendelkező cmdletekkel lehet kezelni:

```
[16] PS C:\> Get-Command -Noun psbreakpoint

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Disable-PSBreakpoint               Disable-PSBreakpoint [-Brea...
Cmdlet          Enable-PSBreakpoint                Enable-PSBreakpoint [-Id] <...
Cmdlet          Get-PSBreakpoint                   Get-PSBreakpoint [[-Script]...
Cmdlet          Remove-PSBreakpoint                Remove-PSBreakpoint [-Break...
Cmdlet          Set-PSBreakpoint                   Set-PSBreakpoint [-Script] ...
```

Ezeket vagy szkriptek szintjén lehet kezelni, ha pozícióhoz (sor, oszlop) rendeljük a megszakítást, vagy a konzol ablakban is használhatjuk ad-hoc módon, ha parancsokhoz vagy változóhoz rendeljük ezeket.

Az első lépés a megszakítási pont létrehozása, ehhez a `Set-PSBreakPoint` cmdletet használhatjuk. Nézzük ennek szintaxisait:

```
[17] PS C:\> (get-help Set-PSBreakpoint).syntax

Set-PSBreakpoint -Command <string[]> [[-Script] <string[]>] [-Action <scriptbl
ock>] [<CommonParameters>]
Set-PSBreakpoint [-Script] <string[]> [-Line] <Int32[]> [[-Column] <int>] [-Ac
tion <scriptblock>] [<CommonParameters>]
Set-PSBreakpoint -Variable <string[]> [[-Script] <string[]>] [-Mode {Read | Wr
ite | ReadWrite}] [-Action <scriptblock>] [<CommonParameters>]
```

Nézzük az utolsó opciót, amikor is változót figyelünk. Látható, hogy három üzemmód van: olvasást, írást vagy mindkettőt figyeljük. Az alapérték az írás figyelése:

```
PS C:\> Set-PSBreakpoint -Variable a

  ID Script            Line Command          Variable         Action
  -- ------            ---- -------          --------         ------
   0                                         a


PS C:\> $a = 8
Entering debug mode. Use h or ? for help.

Hit Variable breakpoint on '$a' (Write access)
```

```
$a = 8
[DBG]: PS C:\>>> ?

 s, stepInto         Single step (step into functions, scripts, etc.)
 v, stepOver         Step to next statement (step over functions, scripts, etc.
)
 o, stepOut          Step out of the current function, script, etc.

 c, continue         Continue execution
 q, quit             Stop execution and exit the debugger

 k, Get-PSCallStack  Display call stack

 l, list             List source code for the current script.
                     Use "list" to start from the current line, "list <m>"
                     to start from line <m>, and "list <m> <n>" to list <n>
                     lines starting from line <m>

 <enter>             Repeat last command if it was stepInto, stepOver or list

 ?, h                Displays this help message


For instructions about how to customize your debugger prompt, type "help about_
prompt".

[DBG]: PS C:\>>>
```

A fenti példában az „a” változó manipulálására kérek megszakítást a [13]-as sorban. Majd értéket adok `$a`-nak, erre rögtön debug, azaz hibakereső üzemmódba lép a prompt. Itt számos parancs áll rendelkezésünkre, amelyeket a kérdőjellel ki is listáztam. Látható, hogy megváltozott a prompt is, a sor elején a `[DBG]` előtag látható. Ebben a promptban bármit csinálhatunk, futtathatunk parancsokat vagy a fenti, speciális hibakezelő parancsokat alkalmazhatjuk.

Ha most kiolvasom a `$a` értékét, akkor a következőket látom:

```
[DBG]: PS C:\>>> $a
1
```

A [14]-es sorral ellentétben az `$a` értéke még a korábbi, azaz a megszakítási pontok mindig a feltétel megjelenése előtti ponton állítják le a futást. Kilépni a `C`, azaz `Continue`-val lehet ebből az üzemmódból. A megszakítási pontokat megszűntetni a következő kifejezéssel lehet:

```
[23] PS C:\> Get-PSBreakpoint | Remove-PSBreakpoint
```

A PSBreakPointok is „scope”-pal rendelkeznek, azaz ha egy olyan függvényt vagy szkriptet futtatok, amelyben szintén van `$a` változó, akkor a felsőbb scope-okban definiált megszakítási pontok erre is hatni fognak. Ennek bemutatására egy kis előkészítést végeztem:

```
[38] PS C:\> $a = 123
[39] PS C:\> function belső ($a){"Ez egy belső a: $a"}
[40] PS C:\> Set-PSBreakpoint -Variable a -Mode readwrite

  ID Script            Line Command          Variable         Action
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
-- ------           ---- -------          --------         ------
 6                                        a
```

Van tehát egy `$a` külső változóm, egy „belső” nevű függvényem, amiben van egy belső `$a` változó, majd definiálok egy megszakítási pontot az „a” változóhoz mind írásra, mind olvasásra. Nézzük, mi történik, ha meghívom a `belső` függvényt:

```
[41] PS C:\> belső 8
Hit Variable breakpoint on '$a' (ReadWrite access)

function belső ($a){"Ez egy belső a: $a"}
[DBG]: PS C:\>>> $a
8
[DBG]: PS C:\>>> $global:a
123
```

A megszakítási pont feléledt, beléptem a hibakeresési üzemmódba. Az `$a`-t kiolvasva megkaptam a paraméterátadás uráni 8-at, azaz a hibakeresés scope-ja a függvény. Viszont elérem akár a külső `$a`-t is a `$global:a` formátum segítségével.

Azt, hogy pontosan hol is vagyok, a „`k`”, azaz `Get-PSCallStack` belső paranccsal tudom megjeleníteni:

```
[DBG]: PS C:\>>> k

Command                 Arguments               Location
-------                 ---------               --------
belső                   {a=8}                   prompt
prompt                  {}                      prompt
```

Ezt a listát alulról fölfele kell értelmezni, azaz a prompton (azaz konzolon) belül vagyok, azon belül a `belső` függvényben. A `Location` oszlopban `prompt` szerepel, hiszen a függvényem is a promptban (konzolon) született, nem pedig egy fájlban van.

Nézzünk még érdekesebb megszakítási pontot, azt a fajtát, ahol az parancshoz van rendelve. Ennek kivesézéséhez nézzünk egy rekurzív függvényt, ami faktoriálist számol:

```
[43] PS C:\> function factor ($i)
>> {
>>     if($i -gt 2){
>>         $i* (factor ($i-1))
>>         }
>>     else {$i}
>> }
>>
[44] PS C:\> Set-PSBreakpoint -Command factor

  ID Script            Line Command          Variable         Action
  -- ------            ---- -------          --------         ------
   7                       factor


[45] PS C:\> factor 6
Hit Command breakpoint on 'factor'

factor 6
[DBG]: PS C:\>>> $i
```

```
[DBG]: PS C:\>>>
```

Ugye a faktoriálist úgy lehet számolni, hogy ha a paraméter nagyobb, mint 2, akkor a faktoriális az úgy képezhető, hogy az aktuális szám szorozva az eggyel kisebb szám faktoriálisa. Ha pedig a szám 2 vagy kisebb, akkor maga a szám. Ezután felállítottam a megszakítási pontot, hozzárendelve a „factor” nevű parancshoz, ami a függvényem. Majd meghívtam ezt, 6 faktoriálisát keresem. Rögtön meg is jelent a megszakítás. Kiolvasva a függvényem belső `$i` változóját azonban nem kaptam semmit, hiszen én még „kívül” vagyok. Ezt a hívási lista ellenőrzésével is láthatom:

```
[DBG]: PS C:\>>> k

Command                   Arguments                 Location
-------                   ---------                 --------
prompt                    {}                        prompt
```

Azaz én még mindig a konzolon vagyok, ott meg nincs `$i` változó definiálva. Belelépni a függvénybe az „s”, azaz `stepInto` paranccsal lehet:

```
[DBG]: PS C:\>>> s
    if($i -gt 2){
[DBG]: PS C:\>>> $i
6
```

Itt már van `$i`. Folytatni lehet a „c”, azaz „`Continue`”, folytatással:

```
[DBG]: PS C:\>>> c
Hit Command breakpoint on 'factor'

        $i* (factor ($i-1))
[DBG]: PS C:\>>> k

Command                   Arguments                 Location
-------                   ---------                 --------
factor                    {i=6}                     prompt
prompt                    {}                        prompt
```

Ha hosszabb a szkriptünk és nem teljesen egyértelmű, hogy hol is vagyunk, akkor kérhetjük a forráskód megjelenítését az „l”, azaz `List` paranccsal:

```
[DBG]: PS C:\>>> l

    1:  function factor ($i)
    2:  {
    3:*     if($i -gt 2){
    4:          $i* (factor ($i-1))
    5:          }
    6:      else {$i}
    7:  }
```

Ha elég sokat folytatjuk a felderítést, akkor akár ilyen hívási mélységet is elérünk:

```
[DBG]: PS C:\>>> k
```

```
Command                 Arguments               Location
-------                 ---------               --------
factor                  {i=2}                   prompt
factor                  {i=3}                   prompt
factor                  {i=4}                   prompt
factor                  {i=5}                   prompt
factor                  {i=6}                   prompt
prompt                  {}                      prompt
```

Miután nagyon rövidke a függvényem, így a lépésenkénti végrehajtás: „s” – `stepInto`, a továbblépés az adott szinten: „v”, `stepOver` és a kilépés az adott szintről: „o”, `stepOut` lehetőségek között nincs sok látványbeli különbség.

Nézzünk még egy lehetőséget, a megszakítási pontokhoz valamilyen szkriptblokkot is rendelhetünk a debug prompt helyett:

```
[56] PS C:\> Set-PSBreakpoint -Command factor -Action {write-host "Most az i: $
i"}

  ID Script             Line Command          Variable         Action
  -- ------             ---- -------          --------         ------
   9                        factor                           write-host "M...


[57] PS C:\> factor 5
Most az i:
Most az i: 5
Most az i: 4
Most az i: 3
120
```

Ezzel tehát nem szakad meg a futás, hanem a megszakítási feltétel teljesülésekor lefut az `-Action` paraméternek átadott szkriptblokk.

Utoljára hagytam a szerintem legkevésbé érdekes lehetőséget, amikor is a megszakítási pontot a szkriptünk valamely pozíciójához rendeljük. Íme a szkriptem:

```
1  param ($x)
2  function factor ($i)
3  {
4      if($i -gt 2){
5          $i* (factor ($i-1))
6          }
7      else {$i}
8  }
9
10 factor $x
11
```

59. ábra A fact.ps1 szkript

Ennek a szkriptnek az 5. sorához rendelem a megszakítási pontot:

```
[59] PS C:\> Set-PSBreakpoint -Script C:\munka\fact.ps1 -Line 5
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
  ID Script             Line Command          Variable         Action
  -- ------             ---- -------          --------         ------
  10 fact.ps1              5
[60] PS C:\> C:\munka\fact.ps1 5
Hit Line breakpoint on 'C:\munka\fact.ps1:5'

fact.ps1:5           $i* (factor ($i-1))
[DBG]: PS C:\>>> $i
5
[DBG]: PS C:\>>> v
Hit Line breakpoint on 'C:\munka\fact.ps1:5'

fact.ps1:5           $i* (factor ($i-1))
[DBG]: PS C:\>>> $i
4
[DBG]: PS C:\>>> o
Hit Line breakpoint on 'C:\munka\fact.ps1:5'

fact.ps1:5           $i* (factor ($i-1))
[DBG]: PS C:\>>> c
120
```

Látható, hogy ugyanúgy működik a folyamat, mint a többi esetben.

Egy szkripthez akár több és többfajta megszakítási pontot is rendelhetünk, illetve az `-Action` részbe intelligenciákat helyezhetünk, amivel önműködően tudjuk vezérelni a hibakeresést:

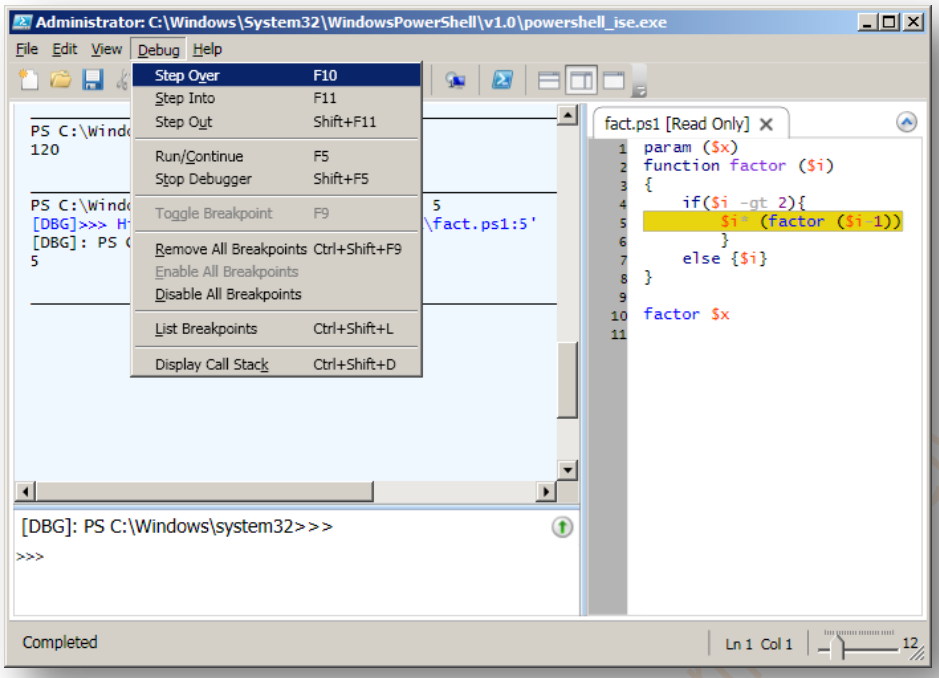
```
[62] PS C:\> Set-PSBreakpoint -Script C:\munka\fact.ps1 -Line 5 -Action {if($i
-gt 3){Write-Host "i: $i"}else{continue} }

  ID Script             Line Command          Variable         Action
  -- ------             ---- -------          --------         ------
  11 fact.ps1              5                                  if($i -gt 3){...


[63] PS C:\> C:\munka\fact.ps1 5
i: 5
i: 4
120
```

Ebben a példában automatikusan továbblép a futás, ha az `$i` értéke kisebb egyenlő, mint 3. A `PSBreakPoint`-okat még hatástalaníthatjuk és újra élesíthetjük a `Disabe-PSBreakpoint` és az `Enable-PSBreakpoint` cmdletekkel.

2.3.6.5 Megszakítási pontok kezelése a grafikus szerkesztőben

Ha már hibakeresésre fanyalodunk, akkor valószínű soksoros szkriptekkel dolgozunk, amelyeket a grafikus felületen könnyebb készíteni. Így érdemes megnézni, hogy itt hogyan lehet ezekkel a lehetőségekkel élni. A PowerShell ISE felületén a fenti lehetőségek egy része a menübe ki van vezetve:

A könyv tanfolyami felhasználása nem

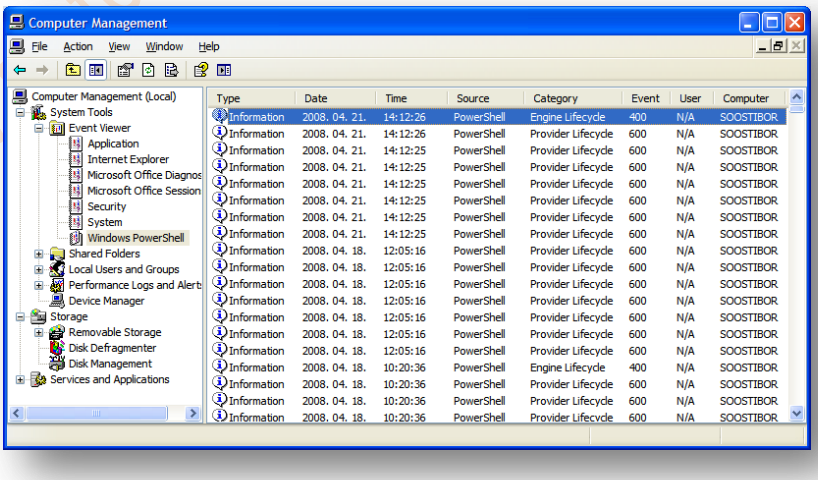


60. ábra A PowerShell ISE hibakeresési lehetőségei

A sorokhoz kötött megszakítási pontot könnyű elhelyezni és kezelni. Parancshoz vagy változóhoz ugyanúgy kell, mint a konzolon. Ha már belekerültünk a hibakeresési üzemmódba, akkor különböző gyorsbillentyűkkel lehet vezérelni a továbblépést.

2.3.7 A PowerShell eseménynaplója

A PowerShell számára külön eseménynapló nyílik telepítése után. Ide nem az egyes parancsok, szkriptek futásának eredménye kerül be, hanem a konzol és a providerek betöltésének státusinformációi:



61. ábra A PowerShell eseménynaplója

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Azaz itt akkor látunk figyelmeztető vagy hibabejegyzéseket, ha valami nagyon súlyos hiba lép fel a PowerShell környezetben.

A könyv tanfolyami felhasználása nem engedélyezett!

2.4 Fejlett függvények – script cmdletek

Az előző fejezetben láttuk, hogy nagyon sok „nyűgös” feladatunk van saját függvényeinkkel vagy szkriptjeinkkel kapcsolatban, ha igazán „profi” megoldást akarunk készíteni. A paraméterek típusának, értékének vagy darabszámának ellenőrzése, valamilyen súgó készítése a függvény vagy szkript használatához, „whatif” jellegű futtatás leprogramozása, hibakezelés, stb. A PowerShell 2.0-ban ezekből a feladatokból jó néhányat rábízhatunk magára a PowerShellre, hiszen ezeket a cmdletek esetében az alkotók már elkészítették, így nekünk csak igénybe kell venni. Ezáltal nagyon fejlett függvényeket és szkripteket tudunk készíteni igazi programozás nélkül. Nézzük, hogy az ilyen fejlett szolgáltatásokat hogyan tudjuk igénybe venni.

2.4.1 Az első fejlett függvényem

A fejlett függvények megadására nincs külön kulcsszó, hanem a függvény definiálásakor különböző metaadatokat kell megadni. Nézzünk erre rögtön egy éles példát! Területszámító fejlett függvényt akarok létrehozni, ami annyira okos, hogy akár téglalap, akár ellipszis területét is kiszámítja, pusztán a paraméterezés alapján kitalálja, hogy a kettő közül melyikre is gondoltam:

```
function adv-terület
{
     [cmdletbinding(DefaultParameterSetName = "téglalap")]

    param(
    [Parameter(
        Mandatory = $true,
        Position = 0,
        ParameterSetName = "téglalap",
        HelpMessage = "Téglalap X oldala"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $x,

    [Parameter(
        Mandatory = $false,
        Position = 1,
        ParameterSetName = "téglalap"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $y = $x,

    [Parameter(
        Mandatory = $true,
        Position = 0,
        ParameterSetName = "ellipszis",
        HelpMessage = "Ellipszis kis tengelye (R)"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $r,

    [Parameter(
        Mandatory = $false,
        Position = 1,
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
        ParameterSetName = "ellipszis"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $p = $r
    )

    if($pscmdlet.ParameterSetName -eq "téglalap"){$x*$y}
        else {$r*$p*[math]::pi}
}
```

Az első fejlett lehetőséget a következő rész határozta meg a fenti függvénydefinícióban:

```
    [cmdletbinding(DefaultParameterSetName = "téglalap")]
```

A `cmdletbinding` kulcsszó azt határozza meg, hogy a függvény úgy viselkedik a paraméterekkel szemben, mint az „igazi” cmdletek, azaz ha több paramétert kap, mint amennyit a függvényben definiáltunk, akkor nem kapja meg a felesleget az `args` változó, hanem hibajelzést kapunk. A mi esetünkben két paramétert vár a függvényem, ráadásul kétfajta két paramétert, de erről majd később. Nézzük, hogyan viselkedik kettőnél több paraméter megadásánál:

```
[11] PS C:\> adv-terület 1 2 3 4 5
adv-terület : A positional parameter cannot be found that accepts argument '3'
.
At line:1 char:12
+ adv-terület <<<<  1 2 3 4 5
    + CategoryInfo          : InvalidArgument: (:) [adv-terület], ParameterBi
   ndingException
    + FullyQualifiedErrorId : PositionalParameterNotFound,adv-terület
```

A fenti hibajelzést a 3-as szám, azaz a 3. paraméter okozta, un. „`ParameterBindingException`” hiba lépett fel. Ezt a hibajelzést leprogramozhattam volna én is az `args` változó vizsgálatával, de a metaadatok használatával maga a PowerShell környezet ezt elvégzi helyettünk.

Nézzük akkor a következő „fejlettséget”, azaz a többfajta paraméterezést! Az előzőleg kiemelt `cmdletbindig` részben meghatároztam egy alaphelyzet szerinti paraméterezést, amelynek neve „téglalap”. Ezt a „címkét” kell majd megjelölni az egyes ide tartozó paramétereknél. Nézzünk meg egy „fejlett” paraméter-meghatározást:

```
[Parameter(
        Mandatory = $true,
        Position = 0,
        ParameterSetName = "téglalap",
        HelpMessage = "Téglalap X oldala"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $x
```

Ez a kiemelés egy darab paramétert definiál nagyon részletes módon. A `parameter` metaadat meghatározásával megadhatjuk, hogy az adott paraméter kötelező-e (`mandatory`) vagy sem. Meghatároztam a paraméter pozícióját (`position`) és hogy melyik peremeterezéshez tartozik

(ParameterSetName). Megadtam egy súgó szöveget (HelpMessage), ami magyarázatot ad a paraméter használatával kapcsolatban:

```
[15] PS C:\> adv-terület

cmdlet adv-terület at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
x: !?
Téglalap X oldala
x: 1
1
```

A fenti példában nem adtam egyetlen paramétert sem a függvényemnek, így PowerShell bekérte az alaphelyzet szerinti paraméterezés kötelező paraméterét. Itt van lehetőségünk a „!?” karakterkombinációval súgót kérni. Igazából csak kötelező paramétereknél érdekes a súgó, hiszen itt van lehetőség segítséget kérni.

Mindezek után megadtam a paraméter típusát ([double]), valamint meghatároztam egy speciális ellenőrző szkriptet ([ValidateScript()]), ami jelen esetben azt ellenőrzi, hogy a paraméter nem negatív-e. Ilyen ellenőrző lehetőségekből számos van, ezeket majd a következő fejezetben tekintem át.

A paraméterdefiníció utolsó részében a paraméter változóját adom meg és az esetleges alaphelyzet szerinti értékét. A kötelező paramétereknél ilyen alapérték megadása felesleges, mert a PowerShell mindenképpen kér paramétert.

Nézzük meg, hogyan néz ki egy másik készletbe tartozó paraméter definíciója:

```
    [Parameter(
        Mandatory = $true,
        Position = 0,
        ParameterSetName = "ellipszis",
        HelpMessage = "Ellipszis kis tengelye (R)"
    )]
    [double]
    [ValidateScript({$_ -ge 0})]
    $r
```

Gyakorlatilag majdnem minden ugyanaz, mint a korábban látott X paraméternél. A különbség csak a ParameterSetName metaadatnál, a súgó szövegénél és a változó nevénél van. Azaz az R paraméter is kötelező, viszont mivel ez már egy másik paraméterezéshez tartozik, az „ellipszishez”, ami nem az alaphelyzet szerinti paraméterezés, így ha paraméter nélkül hívtam meg a függvényt, erre nem kérdez rá a futtatási környezet. Viszont ha a P paramétert használom, ami szintén az ellipszishez tartozik, de az R paramétert nem, akkor már reklamál:

```
[29] PS C:\> adv-terület -p 4

cmdlet adv-terület at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
r: !?
Ellipszis kis tengelye (R)
r: 4
50,2654824574367
```

Nézzük magát a függvénytörzset:

```
    if($pscmdlet.ParameterSetName -eq "téglalap"){$x*$y}
        else {$r*$p*[math]::pi}
```

A fejlett függvények egyik fő ismérve a `$pscmdlet` változó használata. Ez nagyon sok mindent elárul a függvényünk futásának körülményeiről. Ilyen például az, hogy melyik paraméterezést használtuk a függvény hívásakor, amit a `ParameterSetName` tulajdonság ad meg. A függvényem törzsében tehát elég ezt a tulajdonságot megvizsgálnom, maga a PowerShell környezet elemezte ki a paraméterek használata alapján, hogy melyik paraméterezést használtam, nem nekem kellett ezt leprogramoznom. Eszerint a téglalap és az ellipszis területének kiszámítására más-más képletet tudok alkalmazni.

Még egy dolgot nézzünk meg így bevezetésként, kérjük le a függvényem súgóját:

```
[23] PS C:\> get-help adv-terület -full
adv-terület [-x] <Double> [[-y] <Double>] [-Verbose] [-Debug] [-ErrorAction <A
ctionPreference>] [-WarningAction <ActionPreference>] [-ErrorVariable <String>
] [-WarningVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>]
adv-terület [-r] <Double> [[-p] <Double>] [-Verbose] [-Debug] [-ErrorAction <A
ctionPreference>] [-WarningAction <ActionPreference>] [-ErrorVariable <String>
] [-WarningVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>]
```

Bár nem írtam helpet, de már körvonalazódik valami automatikusan. Megkaptam a függvényem paraméterezésének kétfajta lehetőségét és a paramétereket. Ezt majd fokozni fogjuk ebben a fejezetben később. Látható, hogy olyan paraméterek is megjelentek, amelyeket én nem is használtam. Jobban áttekinthetőek ezek a paraméterek a következő módon:

```
[27] PS C:\> (Get-Command adv-terület).parameters | ft -auto

Key             Value
---             -----
x               System.Management.Automation.ParameterMetadata
y               System.Management.Automation.ParameterMetadata
r               System.Management.Automation.ParameterMetadata
p               System.Management.Automation.ParameterMetadata
Verbose         System.Management.Automation.ParameterMetadata
Debug           System.Management.Automation.ParameterMetadata
ErrorAction     System.Management.Automation.ParameterMetadata
WarningAction   System.Management.Automation.ParameterMetadata
ErrorVariable   System.Management.Automation.ParameterMetadata
WarningVariable System.Management.Automation.ParameterMetadata
OutVariable     System.Management.Automation.ParameterMetadata
OutBuffer       System.Management.Automation.ParameterMetadata
```

Csak az első négy paraméter, amit én hoztam létre, a többit a PowerShell rakta hozzá annak köszönhetően, hogy a paraméterezésnél használtam a `[Parameter]` metaadat-címkét. Ezek az automatikus paraméterek az un. „common parameters”, azaz a legtöbb cmdletnél használatos paraméterek, melyekkel a függvényünk viselkedését tudjuk szabályozni a hibákkal, figyelmeztetésekkel szemben, a kimenetet átirányíthatjuk, stb. Ezeknek a használatát már részben láttuk az „igazi” cmdleteknél, részben a fejezet későbbi példáiban előkerülnek.

Azaz megszületett az első fejlett függvényem! Láthatjuk, hogy a fejlett függvényeknek a fő előnyük az, hogy egy csomó dolgot nem kell nekünk leprogramoznunk, hanem a PowerShell által biztosított általános cmdlet-vázat tudjuk felöltöztetni a saját kódunkkal. Így nem kellett külön kezelnünk a függvénynek átadott túl sok paramétert, a paraméterek ellenőrzését, a többfajta paraméterezés használatát, a paraméterek

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

megadásának megkövetelését és a cmdleteknél gyakori paraméterek használatát. Ennél még sokkal többet is igénybe tudunk venni a PowerShell által nyújtott szolgáltatásokból, ezeket nézzük át a következő alfejezetekben.

2.4.2 Paraméterek ellenőrzése

Az első fejlett függvény példámban már láthattunk egy paraméterellenőrzést, amit egy kis szkript hajtott végre. Ez a legnagyobb tudású ellenőrzés, de ennél triviálisabb ellenőrzésekre van jóval egyszerűbb kifejezésünk is, ezek a következők:

Ellenőrző címke	Jelentése
[AllowNull]	Megengedi a paraméternek a `$null` értéket.
[AllowEmptyString]	Megengedi az üres sztring használatát.
[AllowEmptyCollection]	Megengedi az üres gyűjtemény (tömb) megadását.
[ValidateCount(1,5)]	Csak olyan tömböt fogad el, amelynek elemszáma 1 és 5 között van.
[ValidateLength(1,10)]	Csak olyan sztringet enged meg, amely hossza 1 és 10 között van.
[ValidatePattern("\d{2}-\d{3}")]	Csak az adott regex kifejezést kielégítő szöveget enged meg.
[ValidateSet("Vasárnap", "Hétfő", "Kedd")]	Csak a megadott értékek adhatók meg (kis-nagybetű érzéketlen sztringek esetében)

Nézzünk ezekre néhány példát:

```
function adv-validate
{
    param(
    [Parameter(
        Mandatory = $true,
        Position = 0
    )]
    [string]
    [ValidatePattern('^\w{1,4}$')]
    $szöveg,

    [Parameter(Position = 1)]
    [int]
    [ValidateRange(0,3)]
    $szám = 0
    )

$szöveg*$szám
}
```

A fenti példában a `$szöveg` paraméternek csak maximum 4 karakteres sztringet lehet megadni, a `$szám` paraméternek meg csak 0 és 3 közötti egészet. Ettől eltérő paraméter-megadásnál hibákat kapunk:

```
[55] PS C:\> adv-validate abc 2
abcabc
```

használása nem engedélyezett!

```
[56] PS C:\> adv-validate abcde 2
adv-validate : Cannot validate argument on parameter 'szöveg'. The argument "a
bcde" does not match the "^\w{1,4}$" pattern. Supply an argument that matches
"^\w{1,4}$" and try the command again.
At line:1 char:13
+ adv-validate <<<<  abcde 2
    + CategoryInfo          : InvalidData: (:) [adv-validate], ParameterBindi
   ngValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationError,adv-validate

[57] PS C:\> adv-validate abc 20
adv-validate : Cannot validate argument on parameter 'szám'. The 20 argument i
s greater than the maximum allowed range of 3. Supply an argument that is less
 than 3 and then try the command again.
At line:1 char:13
+ adv-validate <<<<  abc 20
    + CategoryInfo          : InvalidData: (:) [adv-validate], ParameterBindi
   ngValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationError,adv-validate
```

Megjegyzés

Ha több paramétert is ellenőriztetünk, akkor az első hibánál megszakad a futtatás, és megkapjuk a vonatkozó hibajelzést, ha további paraméterek sem felelnek meg a különböző feltételeknek, akkor azokra már nem kapjuk meg a figyelmeztetést.

Nézzünk egy példát a paraméter értékkészletének meghatározásához. Ez a függvény csak a hét napjait fogadja el paraméterként és azt adja meg, hogy az adott nap a héten mely dátumra esik:

```
function adv-hétnapja
{
 param(
    [string]
    [ValidateSet("Vasárnap", "Hétfő", "Kedd", "Szerda", "Csütörtök",
        "Péntek", "Szombat")]
    $nap
    )
$napok = [collections.arraylist] ("vasárnap", "hétfő", "kedd", "szerda",
    "csütörtök", "péntek", "szombat")

(get-date).adddays($napok.indexof($nap.tolower()) -
     ([int] (get-date).dayofweek))
}
```

És nézzük hogyan működik egy érvényes és egy érvénytelen paraméter esetében:

```
[67] PS C:\> adv-hétnapja vasárnap

2010. január 3. 13:14:43


[68] PS C:\> adv-hétnapja valami
adv-hétnapja : Cannot validate argument on parameter 'nap'. The argument "vala
mi" does not belong to the set "Vasárnap,Hétfő,Kedd,Szerda,Csütörtök,Péntek,Sz
ombat" specified by the ValidateSet attribute. Supply an argument that is in t
he set and then try the command again.
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
At line:1 char:13
+ adv-hétnapja <<<<  valami
    + CategoryInfo          : InvalidData: (:) [adv-hétnapja], ParameterBindi
   ngValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationError,adv-hétnapja
```

Megjegyzés

Látható, hogy az összehasonlítás érzéketlen a kis-nagy betűkre, ha arra érzékeny megoldást szeretnénk, akkor használjunk szkript alapú ellenőrzést!

2.4.3 Csőelemek kezelése

Nézzük a következő problémát: szeretnék egy olyan függvényt, ami minden fajta objektumnak veszi a nevét és ezt, valamint a nevének a hosszát kiírja. A trükk az lenne még, hogy szeretném, ha ez a függvény csövezhető is lehetne és hagyományos módon is átadható legyen neki bármilyen objektum. Sőt! Ha sztringet adok neki, akkor azt és annak hosszát írja ki.

„Fejletlen” függvénnyel valahogy így indulnék el:

```
function névhossz
{
    param ($name)

    begin {
        if($name) {
            if($name -is [string]) {$name | névhossz}
            else {$name.name | névhossz}
        }
    }
    process {
        if($_) {
            if($_ -is [string]) {"Név: $_, név hossza: $($_.length)" }
            else {"Név: $($_.name), név hossza: $($_.name.length)" }
        }
    }
}
```

Hát… Nem nagyon szép. Hiszen kétszer kell implementálni az egész logikáját, egyszer arra az esetre, ha hagyományosan paraméterezzük, ezt a `begin` szekcióban tettem meg. Másodszor meg a csövezéses használatra, ezt a `process` részben tettem meg. És még ezen belül is kétszer kell megvalósítani azt az esetet, amikor a paraméterként átadott objektumnak sztring és külön, amikor nem az.

Nézzük mindezt fejlett függvénnyel:

```
function adv-név
{
    param(
    [Parameter(
        Mandatory = $true,
        ValueFromPipeline = $true,
        ValueFromPipelineByPropertyName = $true
    )]
    [string]
```

```
    $name
    )
    process {
        "Név: $name, név hossza: $($name.length)"
    }
}
```

Hoppá!!!! Ez sokkal szebb! A lényegi rész a `process` szekcióban található, gyakorlatilag nagyon egyszerű. Mindazt, amit a „fejletlen” változatban nekem kellett megvalósítani azt most megteszi helyettem a PowerShell! Nézzük, hogyan?

A függvény paraméterének neve `$name`. Egyrészt kötelező ez a paraméter (`Mandatory`), ezt már megtárgyaltuk korábban. Másrészt fogad objektumot a csővezetékből (`ValueFromPipeline = $true`). Ez ahhoz kell, hogy ha a csőből sztringek jönnek, akkor így is engedélyezzük a `$name` paraméter feltöltését. Harmadrészt fogad ez a paraméter úgy is értéket a csőből, hogy nem magát az objektumot adjuk át neki, hanem ha az objektumnak is van `name` tulajdonsága, akkor ezt adjuk át a függvényünk `name` paramétereként (`ValueFromPipelineByPropertyName = $true`). Ezért fontos, hogy a függvényem paraméterének neve szintén `name` legyen.

Nézzük, hogyan fut:

```
[21] PS C:\> Get-ChildItem | adv-név
Név: Program Files, név hossza: 13
Név: Program Files (x86), név hossza: 19
Név: Users, név hossza: 5
Név: Windows, név hossza: 7
Név: fájl.txt, név hossza: 8
[22] PS C:\> "kakukk", "csirke" | adv-név
Név: kakukk, név hossza: 6
Név: csirke, név hossza: 6
[23] PS C:\> adv-név "kakukk"
Név: kakukk, név hossza: 6
[24] PS C:\> adv-név (Get-Item C:\fájl.txt)
Név: C:\fájl.txt, név hossza: 11
```

Mind a négy elvárt módon működött a függvényem.

Megjegyzés

A sztring egy nagyon gyakran használt típus, és emiatt (majdnem?) minden objektumnak van `ToString()` metódusa. Elvileg nem feltétlenül kellett volna alkalmazni a `ValueFromPipelineByPropertyName = $true` kitételt, mert enélkül automatikusan ez a `ToString` metódus került volna meghívásra paraméterátadáskor. De mégis érdemes külön ezt a paraméterátadási lehetőséget feltüntetni, mert jobb, kiszámíthatóbb eredményt ad. Nézzük ezt a processzek esetében a függvényem `ValueFromPipelineByPropertyName = $false` változatával:

```
PS C:\Users\Administrator> Get-Process | adv-név
Név: System.Diagnostics.Process (conhost), név hossza: 36
Név: System.Diagnostics.Process (conhost), név hossza: 36
Név: System.Diagnostics.Process (csrss), név hossza: 34
Név: System.Diagnostics.Process (csrss), név hossza: 34
…
```

A könyv tanfolyami felhasználása nem engedélyezett!

Látható, hogy bár működik így is a függvényem, de nem az igazi eredményt adta, mivel a processz objektumoknál a `ToString` metódus nem csak a processz nevét adja vissza, hanem a típusát is. Ezzel szemben a függvényem eredeti változatában jó eredményt kapok:

```
PS C:\Users\Administrator> Get-Process | adv-név
Név: conhost, név hossza: 7
Név: conhost, név hossza: 7
Név: csrss, név hossza: 5
Név: csrss, név hossza: 5
…
```

Ezzel a lehetőséggel tehát nagyon elegáns, a paramétereket csővezetékből is fogadni képes függvényeket tudunk létrehozni. Így most a *1.9.2 Csővezeték feldolgozása (Foreach-Object) – újra* fejezetben látott „`Saját-foreachobject`” függvényt már sokkal profibban, tömörebben is meg tudnánk írni.

Összefoglalásul, a fejlett függvényekben a csővezetékből érkező objektumok „igazi” paraméterhez való rendelését tudjuk nagyon elegánsan rábízni a PowerShell környezetre, ezzel szintén sok munkát spórolhatunk meg és nem utolsó sorban sokkal átláthatóbb, elegánsabb lesz a szkriptünk.

2.4.4 Függvényeink óvatos végrehajtása (-WhatIf)

Láthattuk korábban, hogy az olyan cmdletek esetében, amelyek valamilyen maradandó változást okoznak objektumokon, lehetőség van óvatos, „mi lenne ha” jellegű futtatásra a `-WhatIf` kapcsoló használatával. Ezt természetesen szintén le tudnánk programozni, de a fejlett függvények esetében ennek váza szintén rendelkezésünkre áll. Nézzünk egy olyan függvényt, ami a fájlok kiterjesztését nevezi át:

```
function rename-extention
{
     [cmdletbinding(
         ConfirmImpact = "Medium",
         SupportsShouldProcess = $true
    )]

    param(
    [Parameter(
         Mandatory = $true,
         Position = 1,
         ValueFromPipeline = $true,
         HelpMessage = "Átnevezendő fájl"
    )]
    [System.IO.FileInfo]
    [ValidatePattern('\w+\.\w+$')]
    $file,

    [Parameter(
         Mandatory = $true,
         Position = 0,
         HelpMessage = "Új kiterjesztés"
    )]
    [string]
    $extension
    )

    process {
         if ($PSCmdlet.ShouldProcess("$file" , "Kiterjesztés átnevezése
$extension-re"))
```

```
        { Rename-Item -path $file.fullname -NewName ($file.basename + '.' +
$extension)
        }
    }
}
```

Kezdjük először a paraméterezéssel! Ez már valószínű érthető az eddigiek alapján. Elsőként a `$file` paramétert definiálom kötelezőként, első pozícióba (valójában ez a 2., de a PowerShell 0-tól sorszámoz), csővezetékből is érkezhet a paraméter értéke és készítettem súgót is, valamint meghatároztam a típusát és a mintát (csak olyan fájlnevet fogadok el, amiben van kiterjesztés). Majd az új kiterjesztés megadását szolgáló `$extension` paramétert definiáltam első (0.) paraméterként, szintén értelemszerűen.

Nézzük most, hogy mi van a függvény elején:

```
    ConfirmImpact = "Medium",
    SupportsShouldProcess = $true
```

A `SupportsShouldProcess` kapcsolja be a PowerShell azon „okosságát”, amellyel egyszerűen tudjuk megvalósítani a `–whatif`, `-confirm` és `–verbose` paramétereket, amelyeket megszokhattunk azon cmdleteknél, amelyek valami maradandó változást okoznak valamely provider objektumain. Ez még önmagában nem elég, a függvény tevékenységi részében is kell egy kis átalakítást végezni:

```
        if ($PSCmdlet.ShouldProcess("$file" , "Kiterjesztés átnevezése
$extension-re"))
        { Rename-Item -path $file.fullname -NewName ($file.basename + '.' +
$extension)
        }
```

Azaz a tényleges beavatkozást csak egy `if` vizsgálat után szabad elvégezni. Ebben a `$PSCmdlet ShouldProcess` metódusának igazra értékelése után hajtja csak végre a maradandó változásokat. Ez a metódus az, ami figyelembe veszi a `–whatif`, `-confirm` és `–verbose` switch paramétereket. Ha például van `–whatif` paraméter, akkor ez a metódus `$false`-ra értékelődik ki és nem csinálja meg az átnevezést, viszont kiírja, hogy mit csinálna. Ez a szöveg a `ShouldProcess` paramétereként átadott szövegekből tevődik össze:

```
[30] PS C:\> Get-Item C:\fájl.log | rename-extention txt -WhatIf
What if: Performing operation "Kiterjesztés átnevezése txt-re" on Target "C:\fá
jl.log".
```

Azaz az első paraméterként szereplő szöveget úgy érdemes kialakítani, hogy az utaljon a végrehajtandó művelet áldozatára (target), a második paraméter pedig a tevékenységre. Látható, hogy mindkettőben használhatok változókat, így részletes információkat nyerhetek a feldolgozásról, ami ugye a `–whatif` mellett csak teoretikus.

Majdnem ugyanez történik akkor is, ha a `–verbose` kapcsolót használom, csak éppen a `ShouldProcess` ilyenkor igazra értékelődik ki és így végre is hajtja az átnevezést:

```
[32] PS C:\> Get-Item C:\fájl.log | rename-extention txt -verbose
VERBOSE: Performing operation "Kiterjesztés átnevezése txt-re" on Target
"C:\fájl.log".
```

Arany tanfolyami felhasználás – nem engedélyezett!

Azt, hogy mire értékelődjön ki a `ShouldProcess` elemenként mi magunk is szabályozhatjuk, ha a `-confirm` kapcsolót használjuk. Ilyenkor megjelenik az ismert, „bűvös” kérdés:

```
[33] PS C:\> Get-ChildItem C:\sokfájl | rename-extention log -Confirm

Confirm
Are you sure you want to perform this action?
Performing operation "Kiterjesztés átnevezése log-re" on Target "egy.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Rename File" on Target "Item: C:\sokfájl\egy.txt
Destination: C:\sokfájl\egy.log".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y
…
```

Hoppá! Egy fájlra vajon miért kétszer? Mert a függvényem `-confirm` kapcsolója átadódik a benne található `rename-item` cmdletnek is, jobban mondva az adott futtatási környezet `$ConfirmPreference` változója billen át `Low` értékre. Azaz javítani kell a fejlett függvényemet, a `rename-item` résznél:

```
        { Rename-Item -path $file.fullname -NewName ($file.basename +
            '.' + $extension) -Confirm:$false}
```

Ekkor fájlonként már csak egyszer kapok jóváhagyás-kérést:

```
[37] PS C:\> Get-ChildItem C:\sokfájl | rename-extention txt -Confirm

Confirm
Are you sure you want to perform this action?
Performing operation "Kiterjesztés átnevezése txt-re" on Target "egy.log".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Kiterjesztés átnevezése txt-re" on Target "hat.log".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Kiterjesztés átnevezése txt-re" on Target "három.log".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):a
```

Megint csak rengeteg programozástól kímélt meg minket a PowerShell és mégis nagyon profi függvényt kaptunk.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.4.5 Meglevő cmdletek kiegészítése, átalakítása

A `get-random` cmdletnél láttuk a *1.9.1 Véletlen szám generálás és annál sokkal több (Get-Random)* fejezetben, hogy – bár fantasztikusan jó kis cmdlet – de mégsem túl egyszerű vele 0 és 1 közötti véletlen lebegőpontos számot generáltatni. Merthogy ha 1-et adok meg maximum értéknek, akkor ő csak egy 1-nél kisebb véletlen nem negatív egészet fog generálni, ami véletlenül mindig 0. Jó lenne, átalakítani ezt a cmdletet úgy, hogy ha nem adok neki `–Maximum` paramétert, akkor adjon 0 és 1 közti lebegőpontos számot, azaz használatával a Commodore és ZX Spectrum világába megyünk vissza, hiszen akkoriban működött így az ottani véletlenszám-generáló függvény. És ha már belenyúlunk, akkor szeretnék neki egy újabb paramétert is, mondjuk legyen egy „float” nevű kapcsoló, ami pedig arra jó, hogy ha egészet adok neki maximumként akkor maximum ekkora lebegőpontos véletlen számot adjon. Ráadásul az eredeti funkcionalitást nem akarom leprogramozni, a működésnek más esetekben pontosan ugyanúgy kell folynia, mint eddig. Nem tűnik első hallásra egyszerűnek ez a feladat, de szerencsére elég sok segítséget kapunk a PowerShelltől, illetve a .NET keretrendszertől.

A PowerShellben minden objektum, még a cmdletek is objektumok. Egy speciális osztály, a `System.Management.Automation.CommandMetaData` képes megjeleníteni egy meglevő cmdlet jellemzőit:

```
[6] PS C:\> $metadata = New-Object System.Management.Automation.CommandMetaData
 (Get-Command get-random)
[7] PS C:\> $metadata


Name                    : Get-Random
CommandType             : Microsoft.PowerShell.Commands.GetRandomCommand
DefaultParameterSetName : RandomNumberParameterSet
SupportsShouldProcess   : False
SupportsTransactions    : False
ConfirmImpact           : Medium
Parameters              : {[SetSeed, System.Management.Automation.ParameterMet
                          adata], [Maximum, System.Management.Automation.Param
                          eterMetadata], [Minimum, System.Management.Automatio
                          n.ParameterMetadata], [InputObject, System.Managemen
                          t.Automation.ParameterMetadata]...}
```

Egy másik osztály ezen információk alapján képes legenerálni azt a függvénydefiníciós vázat, amibe belehelyezhetjük a testre szabásainkat. Nézzük tehát, hogyan hívhatjuk elő ezt a vázat:

```
[10] PS C:\> [Management.Automation.ProxyCommand]::Create($metadata)
[CmdletBinding(DefaultParameterSetName='RandomNumberParameterSet')]
param(
    [ValidateNotNull()]
    [System.Nullable``1[[System.Int32, mscorlib, Version=2.0.0.0, Culture=neut
ral, PublicKeyToken=b77a5c561934e089]], mscorlib, Version=2.0.0.0, Culture=neu
tral, PublicKeyToken=b77a5c561934e089]
    ${SetSeed},

    [Parameter(ParameterSetName='RandomNumberParameterSet', Position=0)]
    [System.Object]
    ${Maximum},

    [Parameter(ParameterSetName='RandomNumberParameterSet')]
    [System.Object]
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    ${Minimum},

    [Parameter(ParameterSetName='RandomListItemParameterSet', Mandatory=$true,
 Position=0, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [System.Object[]]
    ${InputObject},

    [Parameter(ParameterSetName='RandomListItemParameterSet')]
    [ValidateRange(1, 2147483647)]
    [System.Int32]
    ${Count})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-Random',
 [System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.Com
mandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {
        throw
    }
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Get-Random
.ForwardHelpCategory Cmdlet

#>
```

Ebben a kimenetben egyetlen karaktert sem én gépeltem be! Maga a `Management.Automation.ProxyCommand` osztály `Create` statikus metódusa írta ezt meg helyettem

az eredeti cmdlet metaadatai alapján. Természetesen ezt nem a képernyőn szeretném látni, hanem be szeretném tenni egy fájlba, amihez egy kis függvénydefiníciós körítést is teszek:

```
[15] PS C:\> "function get-randomnew {`r`n" + [Management.Automation.ProxyComma
nd]::Create($metadata) + "}" > C:\munka\get-randomnew.ps1
```

Nézzük akkor egyben ezt a fájlt! Én már beletettem két megjegyzést, hogy ebbe a vázba majd hol kell módosítani ahhoz, hogy a saját igényeink szerinti működést kapjunk.

```
function get-randomnew {
[CmdletBinding(DefaultParameterSetName='RandomNumberParameterSet')]
param(

<#IDE KELL BEILLESZTENI A SAJÁT EXTRA PARAMÉTEREKET#>

    [ValidateNotNull()]
    [System.Nullable``1[[System.Int32, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]], mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]
    ${SetSeed},

    [Parameter(ParameterSetName='RandomNumberParameterSet', Position=0)]
    [System.Object]
    ${Maximum},

    [Parameter(ParameterSetName='RandomNumberParameterSet')]
    [System.Object]
    ${Minimum},

    [Parameter(ParameterSetName='RandomListItemParameterSet', Mandatory=$true,
Position=0, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [System.Object[]]
    ${InputObject},

    [Parameter(ParameterSetName='RandomListItemParameterSet')]
    [ValidateRange(1, 2147483647)]
    [System.Int32]
    ${Count})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }

<#IDE KELL BEILLESZTENI AZT A SAJÁT KÓDOT, MELLYEL A PARAMÉTEREKET ALAKÍTOM ÁT,
ILLETVE HA A NEM CSŐFELDOLGOZÓ SZAKASZBA SZERETNÉK BEAVATKOZNI#>

        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-Random',
[System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
        $steppablePipeline =
$scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {
```

```
            throw
        }
    }

    process
    {
        try {
    <#IDE LEHET BEILLESZTENI A SAJÁT KÓDOT, HA A CSŐFELDOLGOZÁSBA SZERETNÉNK
    BEAVATKOZNI#>
            $steppablePipeline.Process($_)
        } catch {
            throw
        }
    }

    end
    {
        try {
    <#IDE LEHET BEILLESZTENI A SAJÁT KÓDOT, HA A CSŐFELDOLGOZÁS VÉGÉBE SZERETNÉNK
    BEAVATKOZNI#>
            $steppablePipeline.End()
        } catch {
            throw
        }
    }
    <#

    .ForwardHelpTargetName Get-Random
    .ForwardHelpCategory Cmdlet

    #>
    }
```

Ebbe a vázba, a paraméterdefiníciós rész megjelölt helyére illesztem be egyrészt a saját `-Float` nevű kapcsoló paraméteremet:

```
    [Parameter(ParameterSetName='RandomNumberParameterSet')]
    [switch]
    ${Float},
```

És a `begin` szekcióba a következő kódot:

```
            if($pscmdlet.ParameterSetName -eq 'RandomNumberParameterSet' -and
                    ($PSBoundParameters.ContainsKey('Float') -or
                            -not $PSBoundParameters.ContainsKey('Maximum')))
            {
                    if($PSBoundParameters.ContainsKey('Float')){[Void]
    $PSBoundParameters.Remove('Float')}
                    if($PSBoundParameters.ContainsKey('Maximum'))
    {$PSBoundParameters.Maximum = [float] $PSBoundParameters.Maximum}
                    else{$PSBoundParameters.Maximum = [float] 1}
            }
```

A kód annyit csinál, hogy megnézi elsőként, hogy a `RandomNumberParameterSet` paraméterezést használjuk-e, hiszen ezzel generáljuk a véletlen számokat. A másik paraméterezés működésébe, amellyel egy gyűjteményből választunk ki véletlenül elemeket, nem akarok belenyúlni. Emellett még annak is teljesülnie kell, hogy vagy a `-Float` kapcsoló jelen van, vagy nincsen megadva `-Maximum` paraméter.

Ezután, ha van `-Float`, akkor azt a paraméterek közül kiveszem, hiszen azt az „igazi” `get-random` nem tudja értelmezni. Ha volt `-Maximum` és mellette `-Float`, akkor a maximum értékét átalakítom lebegőpontossá, hogy a generált véletlen szám is az legyen, ha meg nem volt maximum, akkor létrehozok egy lebegőpontos 1-et maximumként. Ez utóbbi eset idézi fel a Commodore és ZX Spektrumok véletlen szám generálását.

Pillantsunk bele a „vázba” is, azaz abba a kódba, amelyet a

`Management.Automation.ProxyCommand`

generált számunkra. A paraméterdefiníciós részben semmi különleges nincsen, egy „sima” fejlett függvénydefinícióknál megszokott részt láthatunk. Majd van egy kicsit trükkösebb rész, amely az `-OutBuffer` paramétert veszi át, és ha ez definiálva van a függvényem hívásánál, akkor ezt átírja 1-re annak érdekében, hogy a csőfeldolgozó szakasz egyesével kapja az elemeket. Ezzel azt érjük el, hogy a saját csőfeldolgozó kódrészünknek könnyebb dolga lesz.

Majd a `$wrappedCmd` változóba berakja az „eredeti” `get-random` cmdletet. Erre azért van szükség, mert én a függvényemnek adhattam volna `get-random` nevet is. Ilyenkor, ha ebben a kódban csak egyszerűen meghívnám a `get-random`-ot, akkor valójában rekurzívan hívnám meg saját magát. Ehhez a `$scriptCmd` változóba hozzábiggyesztjük a `$PSBoundParameters`-ben tárolt paramétereket a passzírozó operátorral. Majd nekikezd a csőfeldolgozásnak, méghozzá un. *steppable* módon, azaz saját kóddal megtűzdelt, elemenként felszakított lépésekkel.

Mindenütt `try` blokkokban történik a műveletvégzés, így ha bármilyen hiba lép fel, akkor a `catch` szekciókban lehetőségünk van saját hibakezelő kód beillesztésére is.

Láthattuk, hogy viszonylag egyszerűen lehet testre szabni a gyári cmdleteket. Ezzel nagyon nagy lökést kapott a PowerShell közösség olyan kis szkriptek, un. *proxy* függvények létrehozására, amelyekkel a tovább lehet gazdagítani a PowerShell cmdletek funkcióit.

2.4.6 Dinamikus paraméterek

Már többször említettem a dinamikus paraméterek fogalmát. Ezek olyan paraméterek, amelyek bizonyos feltételek teljesülésekor jelennek meg a cmdleteknél, függvényeknél. A leggyakoribb ilyen feltétel, hogy a parancs éppen melyik provider környezetében fut. Egy kis rejtvényként próbáljuk meg felderíteni az összes olyan cmdletet, amelynek van dinamikus paramétere, illetve azt, hogy mely providerek esetében milyen paraméterekre kell számítanunk. Első lépésként nézzük meg, hogy hogyan tudunk információkhoz jutni ezekről a dinamikus paraméterekről. A help ebben nem segít. Én tudom, hogy pl. a `get-content` cmdletnek egy `-delimiter` dinamikus paramétere, de az alábbi, súgóból származó információ ebben nem segít nekünk:

```
[17] PS C:\> (Get-Help Get-Content).parameters.parameter | ForEach-Object {$_.n
ame}
Credential
Exclude
Filter
Force
Include
LiteralPath
Path
ReadCount
TotalCount
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
UseTransaction
```

Egy másik cmdlettel, a `get-command`-dal is fel tudjuk deríteni a paramétereket, méghozzá sokkal precízebben, immár a dinamikus paraméterekkel együtt:

```
[18] PS C:\> (Get-Command Get-Content).parameters

Key                                     Value
---                                     -----
ReadCount                               System.Management.Automation.Parame...
TotalCount                              System.Management.Automation.Parame...
Path                                    System.Management.Automation.Parame...
LiteralPath                             System.Management.Automation.Parame...
Filter                                  System.Management.Automation.Parame...
Include                                 System.Management.Automation.Parame...
Exclude                                 System.Management.Automation.Parame...
Force                                   System.Management.Automation.Parame...
Credential                              System.Management.Automation.Parame...
Verbose                                 System.Management.Automation.Parame...
Debug                                   System.Management.Automation.Parame...
ErrorAction                             System.Management.Automation.Parame...
WarningAction                           System.Management.Automation.Parame...
ErrorVariable                           System.Management.Automation.Parame...
WarningVariable                         System.Management.Automation.Parame...
OutVariable                             System.Management.Automation.Parame...
OutBuffer                               System.Management.Automation.Parame...
UseTransaction                          System.Management.Automation.Parame...
Delimiter                               System.Management.Automation.Parame...
Wait                                    System.Management.Automation.Parame...
Encoding                                System.Management.Automation.Parame...
```

Az a baj, hogy ez hashtáblát ad kimenetként, így csőfeldolgozó parancsokkal ezt nem lehet szűrni, feldolgozni, így egy kis átalakítást kell végezni ezzel, hogy megvizsgálhassuk, hogy az egyes paraméterek dinamikusak-e:

```
[19] PS C:\> $h_params = (Get-Command Get-Content).parameters
[20] PS C:\> $h_params.Keys | where-object {$h_params[$_].IsDynamic} | ForEach-
Object {$h_params[$_].name}
Delimiter
Wait
Encoding
```

Annyival többet tud a `get-command` a `get-help`-hez képest, hogy átadhatunk neki egy `-ArgumentList` paraméterben mindenféle paramétert, amelyekkel a cmdlet vizsgálatának körülményeit tudjuk modellezni, így pont elő lehet csiholni a dinamikus paramétereit. Azaz nézzük meg ugyanennek a `get-content`-nek a dinamikus paramétereit akkor, ha épp registry környezetben futtatnánk:

```
[27] PS C:\> $h_params = (Get-Command Get-Content -ArgumentList HKLM:).paramete
rs
[28] PS C:\> $h_params.Keys | where-object {$h_params[$_].IsDynamic} | ForEach-
Object {$h_params[$_].name}
[29] PS C:\>
```

A könyv tanfolyami felhasználása nem engedélyezett!

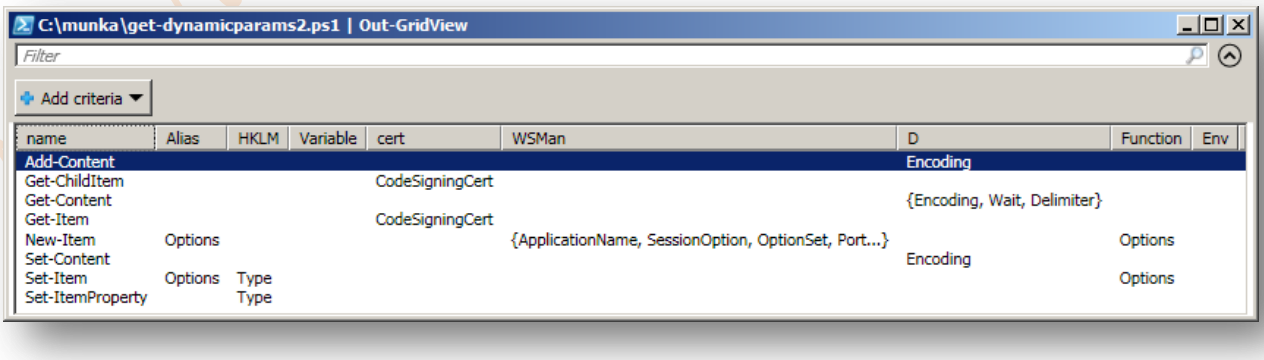
© Soós Tibor – soost@ IQJB.hu

Nem adott semmilyen találatot! Azaz abban a környezetben a fájlrendszerben látott dinamikus paraméterek már nem léteznek.

Ebből kiindulva nézzünk egy olyan szkriptet, ami feltérképezi az összes cmdlet összes provider környezetében levő dinamikus paramétereit:

```
function converthashtocollection ([hashtable] $h)
{
    $h.keys | ForEach-Object {$h[$_] | Where-Object {$_}}
}

$providers = get-psdrive | sort-object -unique provider |
     ForEach-Object {$_.name}

function Get-DynamicParameter {
    param(
    [string]$command,
    $drives
    )

    $ph = @{}

    foreach($d in $drives)
    {
        $ph[$d] = try { converthashtocollection ((Get-Command $command
                -ArgumentList "$($d):").parameters) |
               Where-Object {$_.isdynamic} | ForEach-Object {$_.name}
        }
        catch{}
    }

    if(converthashtocollection $ph){$ph["name"] = $command
          New-Object -TypeName psobject -Property $ph}
}

Get-Command -commandtype cmdlet |  foreach-object {
     get-dynamicparameter ($_.name) $providers} |
     Format-Table (,"name"+$providers) -Wrap
```

És nézzük ennek (illetve nem pont ennek, hanem egy `Out-Gridview`-re módosított változatának) kimenetét:



62. ábra Dinamikus paraméterek a beépített cmdletekben

Nem mondom, hogy egy perc alatt sikerült ezt a szkriptet összeraknom, de nem volt irdatlan nagy munka. A kódban látható, hogy segédfüggvényként készítettem egy függvényt, ami egy hashtáblából gyűjteményt készít. A `Get-DynamicParameter` függvényben végigmegyek a lehetséges meghajtókon, és mindegyikre megnézem az adott cmdlet paraméterezését. Miután ez némelyik cmdletre hibát ad (nem fogadnak elérési út jellegű paramétert), ezért az egész kifejezést egy `Try` blokkba ágyaztam, amihez egy üres `Catch` részt tartozik, így hibamentesen fut le minden cmdletnél.

Az egész adatábrázolást egy olyan hashtáblával oldottam meg (`$ph`), amelyben az egyes meghajtók a kulcsok, és a dinamikus paraméterek tömbje az érték. Ha volt dinamikus paraméter legalább egy is, akkor ezt még megtoldom a `name` kulccsal, ahova a cmdlet nevét töltöm be, és az egész függvény visszatérési értéke egy olyan egyedi objektum, amit ebből a `$ph` hashtáblából generálok, és amelynek tulajdonságai a „name” és a meghajtók lesznek és a tulajdonságértékek a dinamikus paraméterek tömbje, amit végül szép táblázatosan lehet megjeleníteni.

Ilyen dinamikus paramétereket mi magunk is tudunk létrehozni függvényeinkben. Ez hasonlóan, egy „template” alapján történhet, mint ahogy a meglevő cmdleteket kiegészítettük, azzal a különbséggel, hogy ehhez nem kapunk segítséget. Ezért álljon mintaként az alábbi példa, amelyben egy olyan függvény van, ami megszámolja a paraméterként átadott típusú elemeket az adott meghajtón. Az egyszerűség kedvéért ezt csak a fájlrendszerre és az Active Directory-ra készítettem el.

```
function get-itemsoftype
{
    Param
    (
        [parameter(
                Mandatory=$true,
                ValueFromPipeline=$true,
                ValueFromPipelineByPropertyName = $true,
                Position=0,
                ParameterSetName='pset')]
        [String]
        $path
    )
    DynamicParam
    {
        $attributes = new-object -TypeName `
                System.Management.Automation.ParameterAttribute
        $attributeCollection = new-object -TypeName `
                System.Collections.ObjectModel.Collection``1[System.Attribute]
        $paramDictionary = new-object -TypeName `
                System.Management.Automation.RuntimeDefinedParameterDictionary

        $attributes.ParameterSetName = 'pset'
        $attributes.Position = 1
          $attributes.Mandatory = $false

        if ((get-psdrive (
                (split-path (resolve-path $path) -Qualifier) -replace
            ":","")).provider.name -eq "FileSystem")
        {
                $vsa = New-Object -TypeName `
                      System.Management.Automation.ValidateSetAttribute(
                            "file", "folder", "*")
                $attributes.HelpMessage = "Lehetséges értékek: file, folder, *"
          }
        elseif ((get-psdrive (
```

```
                (split-path (resolve-path $path) -Qualifier) -replace
            ":","")).provider.name -eq "ActiveDirectory")
        {
                $vsa = New-Object -TypeName `
                        System.Management.Automation.ValidateSetAttribute(
                                "user", "group", "computer", "*")
                $attributes.HelpMessage =
                        "Lehetséges értékek: user, group, computer, *"
        }

        $attributeCollection.Add($attributes)
        $attributeCollection.Add($vsa)
        $ItemType = new-object -TypeName `
                System.Management.Automation.RuntimeDefinedParameter(
                        "ItemType", [string], $attributeCollection)

        $paramDictionary.Add("ItemType", $ItemType)
        return $paramDictionary
    }
    Process
    {
           switch($ItemType.value){
                "User" {$filter = {$_.objectclass -eq "user"}}
                "Group" {$filter = {$_.objectclass -eq "group"}}
                "Computer" {$filter = {$_.objectclass -eq "computer"}}
                "File" {$filter = {$_.psiscontainer -eq $false}}
                "Folder" {$filter = {$_.psiscontainer -eq $true}}
                "*" {$filter = {$true}}
                default {$filter = {$true}}
           }
        Get-ChildItem -Path $path | Where-Object $filter
     }
}
```

Az egésznek a lényegi része a `DynamicParam` szekció. Ez csak akkor alkalmazható, ha a függvényben van külön `Process` (és esetleg `Begin` és `End`) szekció is. Ezen szekción belül felépítek egy `$attributes` változót, amely a dinamikus paraméter jellemzőit tartalmazza. Ennek az objektumnak vannak olyan tulajdonságai, mint például `ParameterSetName`, `Position`, `Mandatory`.

Van ezen kívül egy `$attributeCollection` változóm, ami tartalmazza a dinamikus paraméter jellemzőit, úgy mint az előbb tárgyalt attribútumokat és ezen kívül a validálási szabályokat.

Ezen kívül van még az `$ItemType` változóm, ez maga a dinamikus paraméter, ahol futásidőben felépítem a paraméter nevét („`ItemType`”), típusát (`[string]`), és ehhez biggyesztem hozzá az előbb említett `$attributeCollection`-t. De ez még nem elég, ezt a paramétert hozzá kell fűzni a függvény dinamikus paramétereihez, ami a `$paramDictionary` változóban van, és amelyet ez a `DynamicParam` szekció ad vissza visszatérési értékként.

Huh! Nem túl egyszerű még csak elmagyarázni sem. Szerencsére ezt nem kell fejből tudni, itt a könyv példája, mindenki kimásolhatja és ez alapján építheti fel saját dinamikus paraméterét.

De mit is csinál a fenti függvény? A meghívásakor a kötelező `$path` tartalmát megvizsgálja, hogy vajon a fájlrendszerre vagy az Active Directory-ra mutat-e. Ennek megfelelően vagy egy olyan `ItemType` dinamikus paramétert vár, amely a „File” és „Folder” szavakat fogadja csak el, vagy egy olyat, ami a „User”, „group”, „computer” és „*” értékek valamelyikét fogadja el. A `Process` szekcióban ennek megfelelően felépítek egy szűrőt és a `Get-ChildItem` cmdlet kimenetét ezzel szűröm.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Természetesen ezt még tovább lehetne finomítani, de itt csak az volt a célom, hogy a dinamikus paraméterekből egy kis ízelítőt adjak.

Megjegyzés

Sajnos, ha egy „igazi” (nem dinamikus) paraméternek alaphelyzet szerinti értéke van, azt akkor még nem veszi fel, amikor a `DynamicParam` szekció kiértékelődik, így annak tartalmát sajnos ott nem tudjuk vizsgálni. Valószínű ez egy hiba, amit később PowerShell verziókban remélhetőleg kijavítanak.

2.4.7 Súgó készítése

Ha készítünk függvényeket és szkripteket, akkor nem árt magyarázatokkal ellátni a „forráskódot”, hogy ha a függvényünket, szkriptünket akár mi, akár mások is megérthessék, ha használni, megérteni vagy továbbfejleszteni szeretnék. Ha pedig nem PowerShellhez értők kezébe adjuk, akkor őket is jó lenne ellátni valamilyen útmutatással a használatot illetően. Nem biztos, hogy ők értenek annyira a forráskód elemzéséhez, hogy abból mindent megértsenek, viszont nagy valószínűséggel rávehetők, hogy a PowerShell beépített súgóját kezelő `Get-Help` cmdletet használják. A két szempontot a PowerShell fejlesztői egyesítették, hogy ne kelljen a kettővel nekünk külön-külön, más helyen és más technológiával bajlódnunk. A megoldás az un. „comment based help”, azaz a megjegyzések formájában beágyazott súgó.

Ez azt jelenti, hogy speciális megjegyzéseket helyezhetünk el a függvényünkbe vagy szkriptünkbe, amelyeket a `Get-Help` ugyanúgy jelenít meg, mint a „gyári” súgótémákat. Nézzünk erre egy példát a `Pithagoras` függvény segítségével:

```
function Pithagoras
{
<#
    .SYNOPSIS
    Kiszámolja a derégszögű háromszög átfogóját a befogókból.

    .DESCRIPTION
    Ez a függvény Pithagoras a^2+b^2=c^2 tétele alapján kiszámolja a
     derékszögű háromszög átfogóját a befogókból.

    .INPUTS
    Nincs, csővezetéken nem fogad paramétereket.

    .OUTPUTS
    System.Double. Kimenet a c átfogó hossza.

    .EXAMPLE
    C:\PS> Pithagoras 3 4
    5

    .EXAMPLE
    C:\PS> Pithagoras 4
    5,65685424949238
    Ebben a példában az egyenlő oldalú derékszögű háromszög átfogóját
     számolta ki.

    .LINK
    További információ Pithagorasz tételéről:
     http://hu.wikipedia.org/wiki/Pitagorasz-t%C3%A9tel
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    .LINK
    about_Arithmetic_Operators

    #>

    param
    (
        [Parameter(
        Mandatory = $true,
        Position = 0,
        HelpMessage = "Ez kötelezően kitöltendő"
        )]
        [double]
        # A derékszögű háromszög egyik befogója
        $a,

        [Parameter(
        Mandatory = $false,
        Position = 1
        )]
        [double]
        # A derékszögű háromszög másik befogója
        $b = $a
    )

    [math]::Sqrt($a*$a+$b*$b)
}
```

Látható, hogy a függvény definíciójának elejére egy speciális megjegyzésblokk (<#...#>) került. Ezen belül ponttal kezdődő címkék a súgón belüli részek kulcsszavai, utánuk meg az adott részhez tartozó leírás van. Vannak olyan részek, amelyek csak egyszer fordulhatnak elő, mint például a `.SYNOPSIS` vagy a `.DESCRIPTION`, de vannak olyanok is, amelyekből több is lehet, mint például az `.EXAMPLE`.

Nézzük meg, hogy mit ad a függvény definiálása után a teljes változatú `Get-Help`:

```
[55] PS C:\> get-help Pithagoras -full

NAME
    Pithagoras

SYNOPSIS
    Kiszámolja a derégszögű háromszög átfogóját a befogókból.

SYNTAX
    Pithagoras [-a] <Double> [[-b] <Double>] [<CommonParameters>]


DESCRIPTION
    Ez a függvény Pithagoras a^2+b^2=c^2 tétele alapján kiszámolja a
    derékszögű háromszög átfogóját a befogókból.


PARAMETERS
    -a <Double>
        A derékszögű háromszög egyik befogója

        Required?                    true
        Position?                    1
```

```
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    -b <Double>
        A derékszögű háromszög másik befogója

        Required?                    false
        Position?                    2
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer and OutVariable. For more information, type,
        "get-help about_commonparameters".

INPUTS
    Nincs, csővezetéken nem fogad paramétereket.


OUTPUTS
    System.Double. Kimenet a c átfogó hossza.


    -------------------------- EXAMPLE 1 --------------------------

    C:\PS>Pithagoras 3 4


    5




    -------------------------- EXAMPLE 2 --------------------------

    C:\PS>Pithagoras 4


    5,65685424949238
    Ebben a példában az egyenlő oldalú derékszögű háromszög átfogóját
    számolta ki.






RELATED LINKS
    További információ Pithagorasz tételéről:
    http://hu.wikipedia.org/wiki/Pitagorasz-t%C3%A9tel
    about_Arithmetic_Operators
```

Látszik, hogy vannak olyan részek, amelyeket nem én definiáltam, hanem a PowerShell automatikusan rakja össze a függvénydefiníció alapján. Ilyenek például a `NAME`, `SYNTAX`, `PARAMETERS`, `REMARKS`. Apropó

PARAMETERS! A függvénydefinícióban látható, hogy a paraméterdefiníciós részbe elhelyezett megjegyzéseket is kiemeli a Get-Help, és ennél a szekciónál megjeleníti.

Erre a súgót tartalmazó megjegyzésblokkra van néhány szigorú szabály és alternatív lehetőség. Ez a blokk függvények esetében három különböző helyen lehet, de egy függvény esetében el kell döntenünk, hogy melyiket választjuk. Lehet a súgóblokk a függvény definíciós részének elején, közvetlenül a function kulcsszót követő sorban, lehet a függvénydefiníciós rész végén, a lezáró kapcsos zárójel előtt, vagy a függvénydefiníció előtt, azaz a function kulcsszót közvetlenül megelőző részben. Szerintem a legjobb az a hely, amit én használtam, azaz a függvény eleje, de végül is ez ízlés dolga.

Megjegyzés

Ha bármelyik súgórészt szimbolizáló kulcsszót elírjuk, vagy nem létező kulcsszót használunk, akkor az egész súgónk érvénytelen lesz és ilyenkor a Get-Help csak az automatikusan generálódó súgórészeket adja vissza.

Súgót lehet készíteni XML formátumban is, erre példát találhatunk a „gyári” súgótémák között a PowerShell telepítési könyvtárában. Ezt itt nem ismertetem, ezt hagyjuk meg a profi fejlesztőknek.

A szkripteket ugyanilyen módon láthatjuk el súgóval. Tegyük fel, hogy az előbb látott Pithagoras függvényt sok más matematikával kapcsolatos függvénnyel együtt egy szkriptfájlba mentjük el matek.ps1 néven, és ennek is generálok egy megjegyzés-alapú súgót:

```
<#
    .SYNOPSIS
    Ebben a szkriptben a matematikai függvényeim vannak.

    .DESCRIPTION
    Függvények:
    Pithagoras

    #>


function Pithagoras
{
<#
    .SYNOPSIS
    Kiszámolja a derégszögű háromszög átfogóját a befogókból.
…
```

Itt is lehet a fájl elejére vagy végére tenni a súgóblokkot. Az elejénél vigyázni kell, mert ha a szkriptem egy függvénydefinícióval kezdődik, mint a fenti példában is, akkor a PowerShell értelmezője ezt a blokkot a függvényhez sorolja. Ezt meg tudjuk akadályozni, ha a súgóblokk és a function kulcsszó sora közé legalább két üres sort teszünk.

Nézzük hogyan kapjuk meg a szkript súgóját:

```
[56] PS C:\> Get-Help .\munka\matek.ps1

NAME
    C:\munka\matek.ps1
```

```
SYNOPSIS
    Ebben a szkriptben a matematikai függvényeim vannak.


SYNTAX
    C:\munka\matek.ps1 [<CommonParameters>]


DESCRIPTION
    Függvények:
    Pithagoras


RELATED LINKS

REMARKS
    To see the examples, type: "get-help C:\munka\matek.ps1 -examples".
    For more information, type: "get-help C:\munka\matek.ps1 -detailed".
    For technical information, type: "get-help C:\munka\matek.ps1 -full".
```

Természetesen ez nem egy „igazi” szkript, így a helpnek is csak elég jelképes értelme van.

2.4.8 Szkriptek nemzetköziesítése

Valószínű ez a téma már csak olyan PowerShell szkriptereket érint, akik nem csak saját munkájuk megkönnyítésére készítenek szkripteket, hanem szélesebb körben felhasználható okosságokat készítenek, akár egy multinacionális cég keretein belül, vagy akár a nagyközönség számára egy „termék” formájában.

Ilyen körben érdekes lehet, hogy a szkriptünk mindenkinek a saját nyelvén szóljon. Ennek feltétele egyrészt az, hogy információt szerezzünk a nyelvi környezetről, amiben a számítógép és azon belül a szkript fut. Ezt megkönnyítendő van néhány cmdletünk. Az első a `Get-Culture`, ami a gép területi beállításából (és nem a felhasználói felület nyelvéből) fakadó nyelv-specifikus jellemzőit adja vissza:

```
[26] PS C:\munka> get-culture | fl *


Parent                         : hu
LCID                           : 1038
KeyboardLayoutId               : 1038
Name                           : hu-HU
IetfLanguageTag                : hu-HU
DisplayName                    : Hungarian (Hungary)
NativeName                     : magyar (Magyarország)
EnglishName                    : Hungarian (Hungary)
TwoLetterISOLanguageName       : hu
ThreeLetterISOLanguageName     : hun
ThreeLetterWindowsLanguageName : HUN
CompareInfo                    : CompareInfo - 1038
TextInfo                       : TextInfo - 1038
IsNeutralCulture               : False
CultureTypes                   : SpecificCultures, InstalledWin32Cultures, Fra
                                 meworkCultures
NumberFormat                   : System.Globalization.NumberFormatInfo
DateTimeFormat                 : System.Globalization.DateTimeFormatInfo
Calendar                       : System.Globalization.GregorianCalendar
OptionalCalendars              : {System.Globalization.GregorianCalendar}
UseUserOverride                : True
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
IsReadOnly                      : False
```

Ez tehát nem magának a Windowsnak a nyelve, hanem a területi beállításból fakadó nyelv. Az én gépem például angol, de a területi beállítás magyar.

A kimenet egy összetett objektum, aminek tulajdonságai is összetettek, így érdemes mélyebbre ásni. Nézzük például a dátum és idő formátumának jellemzőit:

```
[28] PS C:\munka> (get-culture).datetimeformat


AMDesignator                    : de.
Calendar                        : System.Globalization.GregorianCalendar
DateSeparator                   : .
FirstDayOfWeek                  : Monday
CalendarWeekRule                : FirstDay
FullDateTimePattern             : yyyy. MMMM d. H:mm:ss
LongDatePattern                 : yyyy. MMMM d.
LongTimePattern                 : H:mm:ss
MonthDayPattern                 : MMMM d.
PMDesignator                    : du.
RFC1123Pattern                  : ddd, dd MMM yyyy HH':'mm':'ss 'GMT'
ShortDatePattern                : yyyy. MM. dd.
ShortTimePattern                : H:mm
SortableDateTimePattern         : yyyy'-'MM'-'dd'T'HH':'mm':'ss
TimeSeparator                   : :
UniversalSortableDateTimePattern : yyyy'-'MM'-'dd HH':'mm':'ss'Z'
YearMonthPattern                : yyyy. MMMM
AbbreviatedDayNames             : {V, H, K, Sze...}
ShortestDayNames                : {V, H, K, Sze...}
DayNames                        : {vasárnap, hétfő, kedd, szerda...}
AbbreviatedMonthNames           : {jan., febr., márc., ápr....}
MonthNames                      : {január, február, március, április...}
IsReadOnly                      : False
NativeCalendarName              : Gergely-naptár
AbbreviatedMonthGenitiveNames   : {jan., febr., márc., ápr....}
MonthGenitiveNames              : {január, február, március, április...}
```

Innen ki lehet nyerni akár a napok, vagy a hónapok neveit, látható az adott nyelvi környezetben megszokott dátum- és időformátum.

Hasonló kimenetet generál a `Get-UICulture` cmdlet is, azonban ez a Windows nyelve alapján adja meg az adatokat, az én gépem esetében az angolt:

```
[27] PS C:\munka> get-uiculture | fl *


Parent                          : en
LCID                            : 1033
KeyboardLayoutId                : 1033
Name                            : en-US
IetfLanguageTag                 : en-US
DisplayName                     : English (United States)
NativeName                      : English (United States)
EnglishName                     : English (United States)
TwoLetterISOLanguageName        : en
ThreeLetterISOLanguageName      : eng
ThreeLetterWindowsLanguageName  : ENU
CompareInfo                     : CompareInfo - 1033
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
TextInfo                       : TextInfo - 1033
IsNeutralCulture               : False
CultureTypes                   : SpecificCultures, InstalledWin32Cultures, Fra
                                 meworkCultures
NumberFormat                   : System.Globalization.NumberFormatInfo
DateTimeFormat                 : System.Globalization.DateTimeFormatInfo
Calendar                       : System.Globalization.GregorianCalendar
OptionalCalendars              : {System.Globalization.GregorianCalendar, Syst
                                 em.Globalization.GregorianCalendar}
UseUserOverride                : True
IsReadOnly                     : False
```

Ennek felhasználásával nézzük, hogyan lehet mondjuk a dátumokat nem a területi beállítás szerint, hanem a Windows nyelve szerint megjeleníteni:

```
PS C:\> get-date -Format (Get-UICulture).datetimeformat.fulldatetimepattern
kedd, február 23, 2010 12:53:28 du.
```

Ez csak részben sikerült sajnos. A szerkezet angolos, de a részletekben már a magyar nyelv előtört. Szerencsére az objektumoknál megtalálható `ToString` metódus rendelkezik nyelvi beállítás paraméterrel:

```
PS C:\> (get-date).ToString("dddd",(Get-UICulture))
Tuesday
```

Ennek alapján készítettem egy függvényt, ami már teljes egészében korrektül jeleníti meg a dátumokat:

```
function get-dateUICulture {
    param(
        [string] $format,
        [datetime] $datetime = (get-date),
          [object] $cultureinfo
    )

     if($cultureinfo -is [string]){
           $cultureinfo =
                 New-Object system.globalization.cultureinfo $cultureinfo
     }
     elseif($cultureinfo -isnot [System.Globalization.CultureInfo]){
           $cultureinfo = Get-UICulture
     }

    $datetime.tostring(
        $(if($format){$cultureinfo.datetimeformat.$format}),
        $cultureinfo)
}
```

A függvénynek három paraméter adható:

- `-Format`: a `(get-culture).datetimeformat` alatt található különböző formátum-elnevezések
- `-DateTime`: mit szeretnénk fordítani, ha nem adunk meg értéket, akkor az aktuális dátum
- `-CultureInfo`: milyen nyelvnek megfelelő időformátumot kérünk, alaphelyzetben a UICulture-nek megfelelő nyelvet veszi alapul.

Nézzünk pár futtatási példát:

```
PS C:\> get-dateUICulture -format fulldatetimepattern -cultureinfo "hu-hu"
2010. február 23. 13:07:47
PS C:\> get-dateUICulture -format fulldatetimepattern
Tuesday, February 23, 2010 1:07:52 PM
PS C:\> get-dateUICulture -format fulldatetimepattern -cultureinfo "ar-SA"
09/ربيع 4 1/الأول/31 01:07:59 م
PS C:\> get-dateUICulture -format fulldatetimepattern -cultureinfo "ur-PK"
23 فروری, 2010 2:17:42 PM
PS C:\> get-dateUICulture -format fulldatetimepattern -cultureinfo "th-TH"
23 กุมภาพันธ์ 2553 14:17:53
PS C:\> get-dateUICulture -format fulldatetimepattern -cultureinfo "zh-SG"
星期二, 23 二月, 2010 PM 2:18:23
```

A másik problémakör a nemzetközi szkriptekkel kapcsolatban a felhasználóknak szóló üzenetek különböző nyelvi változatainak kezelése. Erre a PowerShell 2.0-ban létrehoztak egy speciális adatszekciót a szkripteken belül, illetve speciális adatfájlok kezelésének lehetőségét. Az adatszekciót `DATA` kulcsszóval kell jelezni, és utána egy szkriptblokkba kell elhelyezni az adatokat, vagy szkriptet. Ebben a szkriptblokkban erősen leszűkített lehetőségekkel állunk szemben, gyakorlatilag alaphelyzetben sztringeket és csak egy cmdletet, a `ConvertFrom-StringData`-t tudjuk használni, de ez erre a feladatra pont elég. Az adatfájlok PSD1 kiterjesztésűek, és a főszkript melletti nyelvi kódnak megfelelő alkönyvtárban kell elhelyezni ugyanolyan név előtaggal, mint magát a szkriptet.

Ha szerepeltetjük a főszkriptünkben az `Import-LocalizedData` cmdletet, akkor az automatikusan az ő `-UICulture` paramétereként megadott nyelvi környezetnek megfelelő alkönyvtárt keresi, és onnan tölti fel a `-bindingVariable`-nek megadott változót. Érdemes itt az `-ErrorAction` hibakezelési paraméternek `SilentlyContinue`-t adni, mert különben ha nem találja a megadott nyelvnek megfelelő alkönyvtárt, akkor csúnya hibát jelez. Ilyenkor amúgy nem nyúl a változóhoz, azaz a korábban definiált `DATA` szekció jut érvényre.

Hogyan épül ez az egész fel? Az alábbiakban látható a fő szkript. Ez a `DATA` szekcióban tartalmazza a felhasználóknak szóló alaphelyzet szerinti magyar üzeneteket. A szkript `international.ps1` néven van elmentve, és nem túl sok izgalmasat csinál: kiírja a `$Messages` hashtábla `Start` kulcsában tárolt üzenetét, majd azt, hogy „Futás”, majd ugyanennek a hashtáblának az `End` kulcsában tárolt üzenetét.

```
$Messages = DATA {
    ConvertFrom-StringData @'
    # Alaphelyzet szerinti magyar szövegek
        Start = A szkript éppen indul.
        End = A szkript leállt.
'@ }

Import-LocalizedData -bindingVariable Messages -UICulture $PSCulture `
     -ErrorAction SilentlyContinue

$Messages.Start
Write-Host "---- Futás -----"
$Messages.End
```

Van ugyanakkor a szkript mellett egy „`en-us`” alkönyvtár is, benne az `international.psd1` adatfájl is:

```
PS C:\powershell2\v2\munka> dir .\en-us
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    Directory: C:\powershell2\v2\munka\en-us


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---       2010.02.22.     23:19        230 international.psd1
```

Ennek a tartalma a következő:

```
ConvertFrom-StringData @'
 # English strings
Start = The script has just started.
End = The script ended.
'@
```

Láthatjuk, hogy hasonló a tartalma, mint a szkript `DATA` szekciójának, csak éppen más nyelven vannak a szövegek. Nézzük akkor, hogy hogyan fut a szkript alaphelyzetben:

```
PS C:\powershell2\v2\munka> .\international.ps1
A szkript éppen indul.
---- Futás -----
A szkript leállt.
```

Ha átállítom a területi beállításokat angolra, akkor ugyanez a szkript már másképp szól hozzánk:

```
PS C:\powershell2\v2\munka> .\international.ps1
The script has just started.
---- Futás -----
The script ended.
```

Ezekkel a lehetőségekkel tehát olyan szkripteket tudunk készíteni, amelyek egy többnemzetiségű környezetben is érthetően és elegánsan működnek.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.5 Fájlkezelés

Már többször foglalkoztunk fájlokkal, használtuk a `get-childitem`, `get-item`, stb. cmdleteket, egy kicsit ássunk ebben a témában mélyebbre.

2.5.1 Fájl és könyvtár létrehozása (new-item), ellenőrzése (test-path)

Új fájlt létrehozni a `new-item` cmdlettel lehet:

```
[8] PS C:\scripts> new-item -Path . -Name szöveg.txt -type file -Value "Ez e
gy szöveg"


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
-a---   2008.04.22.    21:31           14 szöveg.txt
```

Természeten nem csak a fájlt, hanem könyvtárt is készíthetünk, talán ez gyakoribb:

```
[9] PS C:\scripts> New-Item -Path . -Name "Alkönyvtár" -type Directory


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\scripts


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
d----   2008.04.22.    21:33        <DIR> Alkönyvtár
```

Könyvtár létrehozása előtt érdemes megnézni, hogy létezik-e esetleg már a könyvtár. Erre a célra a `Test-Path` cmdlet áll rendelkezésünkre:

```
[12] PS C:\> Test-Path C:\scripts
True
```

Feladat lehet még a fájlok attribútumainak beállítása. Ezeket egyszerűen állíthatjuk be a fájlobjektumok tulajdonságainak módosításával:

```
[13] PS C:\> (Get-Item C:\scripts\alice.txt).attributes = "Archive, hidden"
```

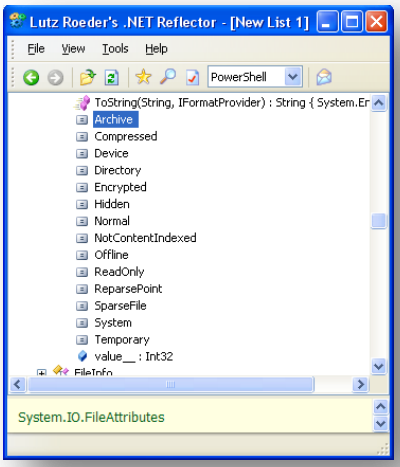
De vajon milyen lehetőségek közül válogathatok az attribútumoknál? Ebben is a Reflector segít, de előtte meg kell nézni, hogy valójában milyen típusú adat a fájlok attribútuma:

```
[14] PS C:\> (Get-Item C:\scripts\alice.txt).attributes.gettype().fullname
System.IO.FileAttributes
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nem meglepő módon ez `System.IO.FileAttributes` típusú, rákeresve erre a Reflectorban ezt láthatjuk:



63. ábra A fájlok attribútumai a Reflectorban

A PowerShellben a típuskonverzió annyira okos, hogy a [13]-as sorban sztringként megadott fájl-attribútumokat (vesszővel elválasztott két attribútum egy sztringben) is képes volt `System.IO.FileAttributes` típusúvá konvertálni.

2.5.2 További játékok az elérési utakkal

Nézzük kicsit tovább, hogy az elérési utakat kezelő cmdletekkel miket tudunk még csinálni. Az első ilyen probléma a meghajtó betűjelének leválasztása egy elérési út jellegű adatból. Legyen a vizsgálandó elérési út mondjuk a `c:\munka`. A korábban már látott `split-path`-szal lehet leválasztani az meghajtót:

```
[37] PS C:\> Split-Path c:\munka
c:\
```

Ha nem kell a „backslash”, akkor ezt még lehet fokozni:

```
[38] PS C:\> Split-Path c:\munka -Qualifier
c:
```

Ha a kettőspont sem kell, akkor ahhoz sajnos már nincsen további kapcsoló, így ahhoz már `-replace` operátor kell:

```
[39] PS C:\> (Split-Path c:\munka -Qualifier) -replace ":", ""
c
```

Ez természetesen a registry esetében is működik:

```
[40] PS C:\> (Split-Path HKLM:\Software -Qualifier) -replace ":", ""
HKLM
```

A könyv tanfolyami felhasználása nem engedélyezett!

Hogyan lehetne azt meghatározni, hogy egy elérési út mely provideren van? Az előző megoldásból kiindulva, az így kijövő meghajtót már átadhatjuk a `Get-PSDrive` cmdletnek, és ennek `Provider` tulajdonságának `Name` tulajdonsága adja a megoldást:

```
[44] PS C:\> (Get-PSdrive ((Split-Path HKLM:\Software -Qualifier) -replace ":",
 "")).provider.name
Registry
```

Mi van akkor, ha az „aktuális könyvtár” jelre is akarjuk, hogy ez a kifejezés működjön? Bele kell kombinálni a `Resolve-Path` cmdletet is, enélkül hibát ad:

```
[45] PS C:\> (Get-PSdrive ((Split-Path . -Qualifier) -replace ":", "")).provide
r.name
Split-Path : Cannot parse path because path '.' does not have a qualifier spec
ified.
At line:1 char:26
+ (Get-PSdrive ((Split-Path <<<<  . -Qualifier) -replace ":", "")).provider.na
me
    + CategoryInfo          : InvalidArgument: (.:String) [Split-Path], Forma
   tException
    + FullyQualifiedErrorId : ParsePathFormatError,Microsoft.PowerShell.Comma
   nds.SplitPathCommand

Get-PSDrive : Object reference not set to an instance of an object.
At line:1 char:13
+ (Get-PSdrive <<<<  ((Split-Path . -Qualifier) -replace ":", "")).provider.na
me
    + CategoryInfo          : NotSpecified: (:) [Get-PSDrive], NullReferenceE
   xception
    + FullyQualifiedErrorId : System.NullReferenceException,Microsoft.PowerSh
   ell.Commands.GetPSDriveCommand

[46] PS C:\> (Get-PSdrive ((Split-Path (resolve-path .) -Qualifier) -replace ":
", "")).provider.name
FileSystem
```

A megoldás a [46]-os sorban már működött.

Mit tud még a `Split-Path`:

```
[47] PS C:\> split-path C:\munka\a.txt -NoQualifier
\munka\a.txt
[48] PS C:\> split-path C:\munka\a.txt -Parent
C:\munka
[49] PS C:\> split-path C:\munka\a.txt -Leaf
a.txt
```

A [47]-es sorban a meghajtó nélküli elérési utat kaptam meg a `-NoQualifier` kapcsoló használatával, a [48]-as sorban a szülő mappa elérési útját adta vissza a parancs a `-Parent` kapcsolóval, a [49]-es sorban meg a „gyerek” objektum nevét a `-Leaf` kapcsolóval. Ezekkel a lehetőségekkel viszonylag kevés esetben kell nekünk sztringműveletekkel kibányászni a megfelelő elérési út szakaszokat.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.5.3 Rejtett fájlok

Nézzük meg a C: gyökerének elemeit (fájlok, könyvtárak), amelyek „p” betűvel kezdődnek:

```
[12] PS C:\> Get-ChildItem p*


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          2008.03.29.    20:47         <DIR> powershell2
d-r--          2008.04.16.    23:03         <DIR> Program Files
d-r--          2008.04.17.    22:48         <DIR> Program Files (x86)
```

De hol van például a pagefile.sys? Alaphelyzetben a `Get-ChildItem` a rejtett fájlokról nem vesz tudomást. Van neki egy `-force` kapcsolója, amellyel a rejtett fájlokat is láttathatjuk:

```
[30] PS C:\> Get-ChildItem p* -force


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          2008.03.29.    20:47         <DIR> powershell2
d-r--          2008.04.16.    23:03         <DIR> Program Files
d-r--          2008.04.17.    22:48         <DIR> Program Files (x86)
-a-hs          2008.04.18.    20:45    2145386496 pagefile.sys
```

Ugyanezt a kapcsolót kell használni a `get-item` cmdlet esetében is.

2.5.4 Szövegfájlok feldolgozása (Get-Content, Select-String)

Szöveges fájlok olvasására a `Get-Content` cmdlet szolgál, amint azt már korábban is láthattuk. Paraméterként egy, vagy több fájl nevét kell megadnunk, a kimenetbe pedig a fájlok tartalma kerül soronként egy-egy karakterlánc képében.

Például keressük meg a Windows mappában azokat a naplófájlokat, amelyekben szerepel az „error” kifejezés. A listában szerepeljen a fájl neve, a megtalált sor szövege, és hogy az hányadik sor az adott fájlon belül! Kezdjünk hozzá! Az első megoldásban tulajdonképpen semmi különleges nincsen, a `Get-Content` sorban megnyitja valamennyi logfájlt, az eredményt odaadjuk a `Select-Stringnek`, aki kiválogatja a megfelelő sorokat. Kell még egy kis formázgatás és készen is vagyunk.

```
PS C:\> Get-Content $env:windir\*.log | select-string -pattern "error" | f
ormat-list filename,line,linenumber


Filename   : InputStream
Line       : COM+[7:32:26]: Warning: error 0x800704cf in IsWin2001Primary
             DomainController
LineNumber : 260
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
...
```

A feladatot két különböző módon is meg fogjuk oldani, bár az első próbálkozás közel sem ad majd tökéletes eredményt. Ha megvizsgáljuk a csőben áramló adatok természetét, akkor nyilvánvalóvá válik, hogy ezzel a módszerrel nem is lehetséges a tökéletes megoldás. Sajnos azonban ezzel a módszerrel útközben olyan információt is eldobáltunk, amire feltétlenül szükségünk lenne a helyes eredmény előállításához. Vizsgáljuk meg, milyen kimenetet produkál a fenti esetben a `Get-Content`! A fájlok tartalma jelenik meg a kimeneten, soronként egy-egy karakterlánc képében, de mindenféle strukturáltság nélkül. A `Select-String` már semmiféle információt nem kap arról, hogy melyik karakterlánc melyik fájlhoz tartozott eredetileg, és még kevésbé tudhatja azt, hogy hányadik sor volt az a fájlban.

Mi került akkor a `Select-String` kimenetébe? A fájlnév helyén mindenhol az `InputStream` kifejezés található, a `LineNumber` pedig azt mutatja, hogy az adott karakterlánc hányadik volt a teljes bemenetben, vagyis az egymás mögé illesztett naplófájlokban. Hát ez nem az igazi!

A `Select-String` azonban bemenetként nem csak karakterláncokat, hanem közvetlenül szöveges fájlokat is fogadhat, a következő megoldásban ezt a tulajdonságot fogjuk felhasználni. Ebben az esetben a `Get-ChildItem` cmdlettől nem a fájlok tartalmát, hanem csak egy `FileInfo` objektumokból álló gyűjteményt kap a `Select-String`, a fájlok tartalmát már ő maga fogja kiolvasni.

```
PS C:\> Get-ChildItem $env:windir\*.log | select-string -pattern "error" -
list | format-list filename,line,linenumber


Filename   : comsetup.log
Line       : COM+[7:32:26]: Warning: error 0x800704cf in IsWin2001Primary
             DomainController
LineNumber : 220
...
```

Ebben az esetben minden szükséges információ rendelkezésre áll, és helyesen kerül be a `Select-String` kimenetébe, így helyesen jelenhet meg a táblázatban is. A `-list` paraméter arra utasítja a cmdletet, hogy csak az első találatig olvasson minden egyes fájlt, így a kapott lista már nem lesz olyan hosszú, mint korábban.

A hibák felderítésénél az is fontos, hogy a hibajelzés környékén milyen egyéb üzenetek vannak a naplófájlban. A `Select-String` ezt is megoldja! Ugyanis van egy `-Context` paramétere is, amellyel nem csak a mintának megfelelő találati sor jelenik meg, hanem annyi megelőző és utána jövő sor, amennyit megadunk a `-Context`-nek.

```
[90] PS C:\> (Select-String -Path C:\Windows\WindowsUpdate.log -Pattern "error"
 -Context 1)[0]

  Windows\WindowsUpdate.log:504:2009-11-09    22:15:25:564     872    4ec    A
U    UpdateDownloadProperties: download priority has changed from 3 to 2.
> Windows\WindowsUpdate.log:505:2009-11-09    22:15:25:564     872    4ec    A
U    WARNING: Failed to change download properties of call, error = 0x80070057
  Windows\WindowsUpdate.log:506:2009-11-09    22:15:25:564     872    4ec    A
U    UpdateDownloadProperties: download priority has changed from 3 to 2.
```

Most csak az első találatot kértem, az „igazi” találati sor a „>” jellel kezdődő, és előtte és mögötte ott van az előzmény és utózmány is. Nézzük meg ezt a kimenetet részletesebben:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[91] PS C:\> (Select-String -Path C:\Windows\WindowsUpdate.log -Pattern "error"
 -Context 1)[0] | fl *


IgnoreCase : True
LineNumber : 505
Line       : 2009-11-09    22:15:25:564     872    4ec    AU    WARNING: Faile
             d to change download properties of call, error = 0x80070057
Filename   : WindowsUpdate.log
Path       : C:\Windows\WindowsUpdate.log
Pattern    : error
Context    : Microsoft.PowerShell.Commands.MatchInfoContext
Matches    : {error}
```

Látható, hogy a találat a kimenet `Line` tulajdonságában van, de akkor hol az előzmény és utózmány? Ezek a `Context` tulajdonságban vannak valójában:

```
[92] PS C:\> (Select-String -Path C:\Windows\WindowsUpdate.log -Pattern "error"
 -Context 1)[0].context

PreContext          PostContext         DisplayPreContext   DisplayPostContext
----------          -----------         -----------------   ------------------
{2009-11-09    2... {2009-11-09    2... {2009-11-09    2... {2009-11-09    ...
```

Ezen belül is a `Context`-nek a `PreContext` és `PostContext` tulajdonságban van az előzmény és az utózmány.

Ha már szövegvizsgálatnál tartunk, készítsünk statisztikát például az `about_signing.help.txt` fájl tartalmáról! Kezdjük a legegyszerűbb, már ismert módszerrel, használjuk a `Measure-Object` cmdletet!

```
[2] PS C:\> Get-Content $pshome\en-US\about_signing.help.txt | Measure-Object -
line -word -character

             Lines               Words          Characters Property
             -----               -----          ---------- --------
               218                1576               11754
```

Eddig rendben is van, de mit kell tennünk, ha arra is kíváncsiak vagyunk, hogy egy adott szó hányszor szerepel a fájlban, vagy például arra, hogy melyik szó fordul elő benne a legtöbbször. A `Get-Content` soronként tördelt kimenetet ad, először is tördeljük ezt tovább szavakká:

```
[17] PS C:\> $szavak = Get-Content $pshome\en-us\about_signing.help.txt | forea
ch-object {-split $_} | Where-Object {$_}
```

Azt hiszem, a fenti parancs azért igényelhet némi magyarázatot. Először is készítünk egy tömböt, ami karakterlánc változókat tud majd fogadni, ebbe kell majd beledobálni a szöveg szavait. A `Get-Content` szállítja a szöveget soronként, a `Foreach-Object` pedig minden egyes sort szavakká tördel a `-split` operátor használatával. A `-split` minden sort a szavaiból álló karakterlánc-tömb képében ad vissza, ezeket adjuk hozzá egyesével a `$szavak` tömbhöz. A cső, vagy futószalag végén a `where-object` szűrés az üres karakterláncokat dobja el. Nézzük mi lett ebből:

```
[18] PS C:\> $szavak.length
1576
```

Remek! A szavak száma pontosan megegyezik azzal, amit a `Measure-Object` adott vissza, valószínűleg minden rendben van. A statisztika most már nem gond, a `Group-Object` csoportosít, a `Sort-Object` pedig az előfordulások száma szerint sorba rendez:

```
[19] PS C:\> $szavak | group-object | sort-object count -descending

Count Name                      Group
----- ----                      -----
  110 the                       {the, The, The, the, the, the, The, the, th...
   52 to                        {to, to, to, to, TO, TO, to, to, To, To...}
   49 you                       {you, you, you, you, you, you, you, you, yo...
   46 a                         {a, a, a, a, a, a, a, a, a, a...}
   34 certificate               {certificate, certificate, certificate, cer...
…
```

Nem probléma az sem, ha egy adott szó előfordulásainak számára vagyunk kíváncsiak, a sorba rendezés helyett egyszerűen a csoportosított listából ki kell választanunk a megfelelő sort. A „scripts” szó előfordulásainak számát például a következő parancs írja ki:

```
[21] PS C:\> $szavak | group-object | where-object {$_.Name -eq "scripts"}

Count Name                      Group
----- ----                      -----
   23 scripts                   {scripts, scripts, scripts, scripts, script...
```

Az egyszerűség☺ kedvéért egyetlen sorba is belesűríthetjük a feladat teljes megoldását, ebben az esetben nincs szükség a változóra csak a következő parancsot kell begépelnünk:

```
[22] PS C:\> Get-Content $pshome\en-us\about_signing.help.txt | foreach-object
{-split $_} | Where-Object {$_} | group-object | sort-object count -descending

Count Name                      Group
----- ----                      -----
  110 the                       {the, The, The, the, the, the, The, the, th...
   52 to                        {to, to, to, to, TO, TO, to, to, To, To...}
   49 you                       {you, you, you, you, you, you, you, you, yo...
   46 a                         {a, a, a, a, a, a, a, a, a, a...}
   34 certificate               {certificate, certificate, certificate, cer...
…
```

Megjegyzés

A fejezet elején használtuk ezt a formátumot a `windir` környezeti változó kiolvasásához:

```
[84] PS C:\> $env:windir
C:\Windows
```

Az „`env:`” egy PSDrive, jön az ötlet, hogy vajon fájlokat meg lehet-e ugyanilyen formában szólítani? Van nekem egy „futók.txt” fájlom a c:\munka könyvtárban:

```
[87] PS C:\munka> $c:futók.txt
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ez nem eredményezett semmit. Talán a teljes elérési út:

```
[88] PS C:\munka> $c:\munka\futók.txt
Unexpected token '\munka\futók.txt' in expression or statement.
At line:1 char:20
+ $c:\munka\futók.txt <<<<
    + CategoryInfo          : ParserError: (\munka\futók.txt:String) [], Pare
   ntContainsErrorRecordException
    + FullyQualifiedErrorId : UnexpectedToken
```

Még rosszabb, hiszen a parancsértelmező a visszaperjelet nem nagyon szereti a kifejezésekben. Korábban már láthattuk, hogy ha „zűrös” karakterek vannak a változók neveiben, akkor kapcsos zárójelbe téve a rendszer elfogadja azokat. Próbáljuk ezt itt is ki:

```
[89] PS C:\munka> ${c:\munka\futók.txt}
Név, Idő
Béla, 5:12
Dezső, 5:37
Karcsi, 5:36
```

Sikerült! Sőt, ugyanez a teljes elérési út nélkül is működik:

```
[90] PS C:\munka> ${c:futók.txt}
Név, Idő
Béla, 5:12
Dezső, 5:37
Karcsi, 5:36
```

Azaz a `get-content` helyett ezt is használhatjuk. Természetesen a `get-content` jóval több szolgáltatást nyújt számunkra, így inkább ezt csak érdekességként említettem.

2.5.5 Sortörés kezelése szövegfájlokban

Gyakran előfordulnak olyan szöveges fájlok, amelyekben az információblokkok nem korlátozódnak egy-egy sorra, hanem átnyúlnak sorvégeken. Ez speciális odafigyelést és feldolgozási módot igényel, hiszen – ahogy láttuk – például a `get-content` is soronként dolgozza fel a szöveges állományokat.

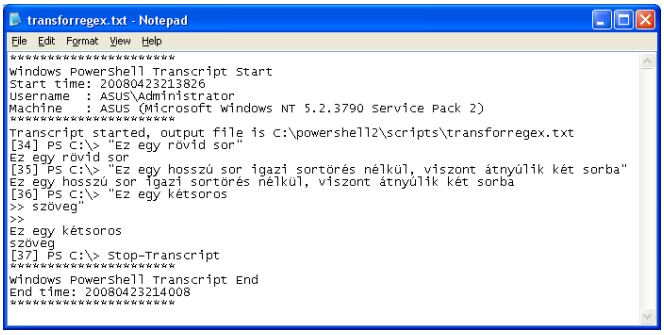
Példaként nézzünk egy olyan szöveges állományt, amelyet maga a PowerShell hoz létre a `start-transcript` cmdlet segítségével:

```
[33] PS C:\> Start-Transcript C:\powershell2\scripts\transforregex.txt
Transcript started, output file is C:\powershell2\scripts\transforregex.txt
[34] PS C:\> "Ez egy rövid sor"
Ez egy rövid sor
[35] PS C:\> "Ez egy hosszú sor igazi sortörés nélkül, viszont átnyúlik két
sorba"
Ez egy hosszú sor igazi sortörés nélkül, viszont átnyúlik két sorba
[36] PS C:\> "Ez egy kétsoros
>> szöveg"
>>
Ez egy kétsoros
szöveg
[37] PS C:\> Stop-Transcript
Transcript stopped, output file is C:\powershell2\scripts\transforregex.txt
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A fenti példában indított transcript tartalmaz mindenféle hosszúságú sort, mesterségesen nyitott új sort alpromptal. Az elkészült fájl a következőképpen néz ki notepaddal nézve:



64. ábra Transcript fájl

Láthatjuk, hogy csak ott tett a fájlba sortörést, ahol tényleg Entert ütöttünk. Olvassuk be ezt a fájlt:

```
[41] PS C:\> $text = get-content C:\powershell2\scripts\transforregex.txt
```

Az így kapott `$text` változó egy sztringtömböt eredményez, amelynek egyes elemei a fájl sorai:

```
[42] PS C:\> $text[8]
Ez egy rövid sor
```

Hogy lehetne ebből egy olyan sztringet készíteni, amely tartalmazza az összes sort? Szerencsére a .NET Framework itt is a segítségünkre siet:

```
[43] PS C:\> $text = [string]::join("`r`n",$text)
```

A `[string]` osztálynak tehát van egy `join` statikus metódusa, amellyel ilyen összefűzéseket lehet végezni. Paraméterként át kell adni egy olyan karaktert (vagy karaktersorozatot), amelyet az összefűzések helyére beilleszt, meg az összefűzendő szöveget. Hogy az eredeti információtartalmat megőrizzük, én egy „kocsivissza-újsor” kombinációval fűztem össze a szövegem darabjait, merthogy a Windowsban ez a „hivatalos” sortörés.

Másik lehetőség az, hogy eleve a beolvasást más módszerrel végezzük, szintén a .NET keretrendszer segítségével, a `File` objektumtípus `ReadAllText` statikus metódusának segítségével:

```
[44] PS C:\> $text2 = [System.IO.File]::ReadAllText("c:\powershell2\scripts\
transforregex.txt")
```

Akármelyik módszert is választjuk, ugyanolyan eredményt kapunk, így immár egyszerűbben tudunk akár soron átnyúló regex kifejezéssel keresni.

2.5.6 Fájl hozzáférési listája (get-acl, set-acl)

A fájlrendszer és a registry objektumainál kiolvasható a hozzáférési lista a `get-acl` cmdlet segítségével:

```
[87] PS C:\> get-acl C:\powershell2\scripts\1.txt | fl
```

Akon tanfolyami felhasználása nem engedélyezett!

```
Path   : Microsoft.PowerShell.Core\FileSystem::C:\powershell2\scripts\1.txt
Owner  : ASUS\Administrator
Group  : ASUS\None
Access : BUILTIN\Administrators Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  FullControl
         ASUS\Administrator Allow  FullControl
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
Audit  :
Sddl   : O:LAG:S-1-5-21-2919093906-1695458891-47906081-513D:(A;ID;FA;;;BA)(
         A;ID;FA;;;SY)(A;ID;FA;;;LA)(A;ID;0x1200a9;;;BU)
```

Az így visszakapott objektum egy `FileSecurity` típusú objektum, melynek a fent látható tulajdonságai közül az `Sddl` elég rémisztőnek néz ki, de szerencsére nem muszáj azzal foglalkozni, emberi fogyasztásra jobban alkalmas hozzáférési szabályok segítségével is lehet beállítani a hozzáférést a fájlokhoz, könyvtárakhoz. A hozzáférési lehetőségeket a következő táblázat tartalmazza:

Hozzáférési jogok (FileSystemRights)	
ListDirectory	WriteAttributes
ReadData	Write
WriteData	Delete
CreateFiles	ReadPermissions
CreateDirectories	Read
AppendData	ReadAndExecute
ReadExtendedAttributes	Modify
WriteExtendedAttributes	ChangePermissions
Traverse	TakeOwnership
ExecuteFile	Synchronize
DeleteSubdirectoriesAndFilesReadAttributes	FullControl

Ezek közül lehet összerakni a kívánt hozzáférési lehetőségeket a következő módon:

```
[17] PS C:\> $Acl = Get-Acl C:\scripts
[18] PS C:\> $entry = New-Object System.Security.AccessControl.FileSystemAcc
essRule("Szkriptelők","Read","Allow")
[19] PS C:\> $entry


FileSystemRights  : Read, Synchronize
AccessControlType : Allow
IdentityReference : Szkriptelők
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

[20] PS C:\> $acl.AddAccessRule($entry)
[21] PS C:\> set-acl C:\scripts $acl
[22] PS C:\> (Get-Acl C:\scripts).Access

FileSystemRights  : Read, Synchronize
AccessControlType : Allow
IdentityReference : ASUS\Szkriptelők
IsInherited       : False
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
InheritanceFlags   : None
PropagationFlags   : None
...
```

A fenti példában a c:\scripts könyvtárhoz szeretném adni a Szkriptelők csoportot olvasási jogosultsággal. Ehhez a [17]-es sorban kiolvasom a meglevő hozzáférési listát, a [18]-as sorban definiálom az új hozzáférési bejegyzést. Ezt ellenőrzésképpen kiíratom a [19]-es sorban. A [20]-as sorban hozzáadom ezt a bejegyzést a hozzáférési listához. Mivel ezt egyelőre csak a memóriában tárolt `$acl` objektum tartalmazza, ezért ezt ki is kell írni a könyvtárobjektumra, amit a [21]-es sorban teszek meg a `set-acl` cmdlettel. Végezetül, a [22]-es sorban ellenőrzésképpen kiolvasom az új hozzáférési listát, amelyben ott szerepel az imént hozzáadott bejegyzés.

Megjegyzés:

A registry elemeinek hozzáférési jogosultságait hasonló módon, de a `RegistryAccessRule` osztály objektumainak segítségével állíthatjuk be.

2.5.6.1 Fájlok tulajdonosai

Az előzőekben láttuk, hogy a `get-acl` kimentében a fájl vagy könyvtár tulajdonosa is kiolvasható. Nézzünk ennek felhasználására egy kis szkriptet, mely segítségével a tulajdonosok szerint szortírozom szét a fájlokat:

```
Set-Location C:\fájlok

Get-ChildItem |
Where-Object {-not $_.PSIsContainer} |
     ForEach-Object {
           $d = (Get-Acl $_).Owner.Split("\")[1]
           if(-not (Test-Path ((get-location).path + '\' + $d)))
           {
                 new-item -path (get-location).path -name $d `
                       -type directory | Out-Null
           }
           Move-Item -path $_.pspath `
                 -destination ((get-location).path + '\' + $d +'\')
}
```

Az elején beállítom az aktuális könyvtárat, majd kilistáztatom az összes fájlját és alkönyvtárát. Mivel nekem csak a fájlok kellenek, ezért a `where-object`-tel kiszűröm a `PSIsContainer` típusú objektumokat.

Az így megmaradt objektumokon egy `foreach-object` ciklussal végigszaladok. Képzem egy `$d` változóba a fájl tulajdonosának a nevét. Itt egy kis trükközésre van szükség, hiszen a felhasználó neve *tartomány\felhasználónév* vagy *gépnév\felhasználónév* formátumú. Nekem csak a felhasználónév kell, így `split()`-tel kettétöröm és veszem a 2. elemet ([1]-es indexű), ami a felhasználói név.

Ezután megvizsgálom a `Test-Path` cmdlettel, hogy van-e már a névnek megfelelő alkönyvtár. Ha nincs, akkor a `new-item` cmdlettel létrehozom. Mire a `move-item` cmdlethez érünk, addigra már biztos van a felhasználónévnek megfelelő alkönyvtár, így át tudom mozgatni oda a fájlt.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.5.6.2 Öröklődés ellenőrzése, beállítása

A fájlkiszolgálók üzemeltetése során különösen fontos annak felderítése, hogy hol vannak felszakított öröklődések, illetve hol vannak explicit jogok kiosztva. Ezekkel foglalkozom részletesebben ebben a fejezetben.

Kicsit korábban, a `Get-ACL` kimenetéből a „sima” `format-list` nem mutatott meg mindent. Nézzük meg csillagosan:

```
[17] PS C:\> get-acl C:\sokfájl\alkönyvtár | Format-List *


PSPath                  : Microsoft.PowerShell.Core\FileSystem::C:\sokfájl\alk
                          önyvtár
PSParentPath            : Microsoft.PowerShell.Core\FileSystem::C:\sokfájl
PSChildName             : alkönyvtár
PSDrive                 : C
PSProvider              : Microsoft.PowerShell.Core\FileSystem
AccessToString          : R2\G-Oktatók Deny  ReadData, AppendData, ReadExtende
                          dAttributes, WriteExtendedAttributes, Delete, Change
                          Permissions
                          R2\G-Oktatók Allow  CreateFiles, ExecuteFile, Delete
                          SubdirectoriesAndFiles, ReadAttributes, WriteAttribu
                          tes, ReadPermissions, TakeOwnership, Synchronize
                          NT AUTHORITY\SYSTEM Allow  FullControl
                          BUILTIN\Administrators Allow  FullControl
                          BUILTIN\Users Allow  ReadAndExecute, Synchronize
                          BUILTIN\Users Allow  AppendData
                          BUILTIN\Users Allow  CreateFiles
                          CREATOR OWNER Allow  268435456
AuditToString           :
Path                    : Microsoft.PowerShell.Core\FileSystem::C:\sokfájl\alk
                          önyvtár
Owner                   : BUILTIN\Administrators
Group                   : R2\Domain Users
Access                  : {System.Security.AccessControl.FileSystemAccessRule,
                           System.Security.AccessControl.FileSystemAccessRule,
                           System.Security.AccessControl.FileSystemAccessRule,
                           System.Security.AccessControl.FileSystemAccessRule.
                          ..}
Sddl                    : O:BAG:DUD:AI(D;OICIID;CCLCSWRPSDWD;;;S-1-5-21-339893
                          8913-3940250523-927435294-1645)(A;OICIID;0x1a01e2;;;
                          S-1-5-21-3398938913-3940250523-927435294-1645)(A;OIC
                          IID;FA;;;SY)(A;OICIID;FA;;;BA)(A;OICIID;0x1200a9;;;B
                          U)(A;CIID;LC;;;BU)(A;CIID;DC;;;BU)(A;OICIIOID;GA;;;C
                          O)
AccessRightType         : System.Security.AccessControl.FileSystemRights
AccessRuleType          : System.Security.AccessControl.FileSystemAccessRule
AuditRuleType           : System.Security.AccessControl.FileSystemAuditRule
AreAccessRulesProtected : False
AreAuditRulesProtected  : False
AreAccessRulesCanonical : True
AreAuditRulesCanonical  : True
```

Jóval több tulajdonság vált láthatóvá! Ezek közül az `AreAccessRulesProtected` az öröklődés szempontjából nagyon fontos, hiszen azok a fájlok vagy könyvtárak, ahol ez a tulajdonság `$true`, ott fel van szakítva az öröklődési lánc, így külön foglalkozni kell velük ha valamit változtatni akarunk a hozzáférési szabályokon.

Elsőként próbáljuk tehát felderíteni, hogy hol vannak felszakított öröklődésű objektumaink:

```
[21] PS C:\> Get-ChildItem C:\sokfájl -recurse | Get-Acl | where-object {$_.are
accessrulesprotected}


    Directory: C:\sokfájl


Path                        Owner                      Access
----                        -----                      ------
alkönyvtár                  BUILTIN\Administrators     R2\G-Oktatók Deny  Rea...
négy.txt                    BUILTIN\Administrators     R2\G-Oktatók Deny  Rea...
```

Hogyan lehetne megszűntetni az öröklődés helyreállítását? Vajon ezt a tulajdonságot közvetlenül átírhatjuk-e?

```
[23] PS C:\> Get-ChildItem C:\sokfájl -recurse | Get-Acl | where-object {$_.are
accessrulesprotected} | get-member -MemberType property | Format-Table -Wrap


   TypeName: System.Security.AccessControl.DirectorySecurity

Name                    MemberType Definition
----                    ---------- ----------
AccessRightType         Property   System.Type AccessRightType {get;}
AccessRuleType          Property   System.Type AccessRuleType {get;}
AreAccessRulesCanonical Property   System.Boolean AreAccessRulesCanonical {get
                                   ;}
AreAccessRulesProtected Property   System.Boolean AreAccessRulesProtected {get
                                   ;}
AreAuditRulesCanonical  Property   System.Boolean AreAuditRulesCanonical {get;
                                   }
AreAuditRulesProtected  Property   System.Boolean AreAuditRulesProtected {get;
                                   }
AuditRuleType           Property   System.Type AuditRuleType {get;}
…
```

Sajnos csak a `get` jelleggel, azaz csak leolvasással lehet ehhez a tulajdonsághoz hozzáférni. Szerencsére van ennek átállítására metódus (csak az `Access` szót tartalmazó metódusok listája):

```
[25] PS C:\> Get-ChildItem C:\sokfájl -recurse | Get-Acl | where-object {$_.are
accessrulesprotected} | get-member *access* -MemberType method | Format-Table -
Wrap


   TypeName: System.Security.AccessControl.DirectorySecurity

Name                    MemberType Definition
----                    ---------- ----------
AccessRuleFactory       Method     System.Security.AccessControl.AccessRule A
                                   ccessRuleFactory(System.Security.Principal
                                   .IdentityReference identityReference, int
                                   accessMask, bool isInherited, System.Secur
                                   ity.AccessControl.InheritanceFlags inherit
                                   anceFlags, System.Security.AccessControl.P
                                   ropagationFlags propagationFlags, System.S
                                   ecurity.AccessControl.AccessControlType ty
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                                    pe)
AddAccessRule             Method    System.Void AddAccessRule(System.Security.
                                    AccessControl.FileSystemAccessRule rule)
GetAccessRules            Method    System.Security.AccessControl.Authorizatio
                                    nRuleCollection GetAccessRules(bool includ
                                    eExplicit, bool includeInherited, type tar
                                    getType)
ModifyAccessRule          Method    bool ModifyAccessRule(System.Security.Acce
                                    ssControl.AccessControlModification modifi
                                    cation, System.Security.AccessControl.Acce
                                    ssRule rule, System.Boolean&, mscorlib, Ve
                                    rsion=2.0.0.0, Culture=neutral, PublicKeyT
                                    oken=b77a5c561934e089 modified)
PurgeAccessRules          Method    System.Void PurgeAccessRules(System.Securi
                                    ty.Principal.IdentityReference identity)
RemoveAccessRule          Method    bool RemoveAccessRule(System.Security.Acce
                                    ssControl.FileSystemAccessRule rule)
RemoveAccessRuleAll       Method    System.Void RemoveAccessRuleAll(System.Sec
                                    urity.AccessControl.FileSystemAccessRule r
                                    ule)
RemoveAccessRuleSpecific Method     System.Void RemoveAccessRuleSpecific(Syste
                                    m.Security.AccessControl.FileSystemAccessR
                                    ule rule)
ResetAccessRule           Method    System.Void ResetAccessRule(System.Securit
                                    y.AccessControl.FileSystemAccessRule rule)
SetAccessRule             Method    System.Void SetAccessRule(System.Security.
                                    AccessControl.FileSystemAccessRule rule)
SetAccessRuleProtection   Method    System.Void SetAccessRuleProtection(bool i
                                    sProtected, bool preserveInheritance)
…
```

Ezek között van az utolsó, a `SetAccessRuleProtection`. Nézzük hogyan tudjuk ezt munkára fogni:

```
[54] PS C:\> Get-ChildItem C:\sokfájl -recurse | Get-Acl | where-object {$_.are
accessrulesprotected}  | ForEach-Object {$_.setaccessruleprotection($false,$tru
e); $_ | set-acl}
```

A `SetAccessRuleProtection` két paramétert vár, az első, hogy legyen-e „védett” az ACL, azaz legyen-e felszakítva az öröklődés. Most nekünk pont az kell, hogy legyen öröklődés, tehát én erre `$false` értéket adok. A második paraméternek akkor van szerepe, amikor pont felszakítjuk az öröklődést és azt mondja meg, hogy szeretnénk-e a korábbi öröklött jogokat explicit jogokként rámásolni az ACL listára. Most ennek nincs szerepe, de sajnos kötelező szerepeltetni. A másik nehézség ezzel a metódussal, hogy közvetlenül a `$_` változóban tárolt objektumra végrehajtva még nem íródik ki ez az objektum a fájlra, így még kell egy `set-acl`, amivel ténylegesen kiírjuk a fájlrendszerbe ezt.

Még egy dologra ügyeljünk! Ilyenkor az történt, hogy az öröklött jogok a könyvtárakon és fájlokon megjelentek, de a korábbi explicit jogok is ottmaradtak. Ha ezek nekünk nem kellenek, akkor ezeket le kell szedni. Nézzük ennek módszerét! Elsőként derítsük fel az explicit jogokat:

```
[70] PS C:\> Get-ChildItem C:\sokfájl\ -Recurse | Get-Acl | ForEach-Object {$_.
access | where-object {! $_.isinherited} }


FileSystemRights  : ReadData, AppendData, ReadExtendedAttributes, WriteExtende
                    dAttributes, Delete, ChangePermissions
AccessControlType : Deny
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
IdentityReference : R2\G-Oktatók
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None
…
```

Az a baj, hogy az ACE (Access Control Entry, azaz egy bejegyzés az ACL listában) szinten van ábrázolva az öröklődés, viszont egy szinttel feljebb, az ACL szinten lehet eltávolítani a bejegyzéseket. A másik probléma, amivel már a korábbi feladatnál is találkoztunk, hogy vissza is kell írni a módosított ACL-eket a fájlokra, nem elég memóriában elvégezni a műveleteket. A teljes megoldás egy sorban is futtatható módon a következő:

```
Get-ChildItem C:\sokfájl\ -Recurse | Get-Acl |
     ForEach-Object {$aclnew = $_; $_.access |
           where-object {! $_.isinherited} |
                 foreach-object {[void] $aclnew.removeaccessrule($_)};
                       Set-Acl -path $aclnew.path -AclObject $aclnew
     }
```

A könnyebb érthetőség kedvéért a szkriptszerkesztő felületén kicsit áttördeltem. Mi történik itt? Veszem a fájlokat, könyvtárakat, veszem az ACL objektumukat, majd képzek egy átmeneti új ACL objektumot, hiszen ezután már az ACL-nek az Access tulajdonságában tárolt ACE objektumai kellenek, így ezen a ponton már a `$_` nem az ACL-t tartalmazza, amelyre nekem később szükségem lesz. Szóval elmentettem az ACL objektumot egy változóba, majd ennek ACE bejegyzéseit sorra veszem, és amelyek nem öröklöttek, azokat szépen kiveszem az elmentett ACL objektum ACE listájáról. Ha már nincs több ilyen, akkor az így „megegyelt” ACE listát tartalmazó új ACL objektumot visszaírom a `Set-ACL` cmdlettel a fájlra vagy könyvtárra. Természetesen lehetne ezt a kis „szkriptet” optimalizálni, például azzal, hogy ha nem is volt változás az ACL objektumon, akkor ne írjuk fel ismét a fájlra, de ezt az olvasóra bízom.

Összefoglalásul elmondható, hogy nem egyszerű az ACL objektumokkal dolgozni, de még mindig sokkal egyszerűbb keresni és tömeges módosításokat végezni PowerShell segítségével, mint a grafikus felületen dolgozni, ahol ugyanezeket a műveleteket szinte reménytelen elvégezni.

2.5.7 Ideiglenes fájlok létrehozása

Gyakran előfordul nagyobb adatfeldolgozással járó feladatoknál, hogy ki kell írni az átmeneti adatokat ideiglenes fájlokba, hiszen a rendelkezésre álló memória nem biztos, hogy elegendő.

A .NET egyik osztálya, a `System.IO.Path` ebben is segítségünkre van. A `GetTempFileName()` statikus metódusa a `temp` környezeti változó által meghatározott könyvtárba létrehoz egy üres fájlt, mely neve „tmp” karaktersorozattal kezdődik, majd jön egy sorszám, és a kiterjesztése „tmp”:

```
[95] PS C:\> [System.IO.Path]::GetTempFileName()
C:\Documents and Settings\Administrator\Local Settings\Temp\tmp16.tmp
```

Érdekes, hogy ezzel a kifejezéssel nem csak megkapjuk a lehetséges következő ideiglenes fájl nevét (mint ahogy a metódus neve sugallná), hanem létre is hozza az ideiglenes fájlt:

```
[104] PS C:\Documents and Settings\Administrator\Local Settings\Temp> dir tm
p*.tmp
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    Directory: Microsoft.PowerShell.Core\FileSystem::C:\Documents and Setti
    ngs\Administrator\Local Settings\Temp


Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-a---    2008.04.20.    11:46            0 tmp15.tmp
-a---    2008.04.24.    23:36            0 tmp16.tmp
```

Az így létrehozott üres fájlba átirányíthatjuk a PowerShell kifejezéseink kimenetét, és felhasználhatjuk majd azt későbbi feldolgozásra. Ha már nincs szükségünk már az ideiglenes fájlunkra, akkor ne feledkezzünk meg törlésükről.

2.5.8 XML fájlok kezelése

Korábban láttuk, hogy a PowerShell képes XML formátumban exportálni objektumait. Nézzük, hogy hogyan lehet egyéb forrásból származó XML állományokkal dolgozni. Példámban egy nagyon egyszerű Access adatbázist exportáltam XML formátumba. Vigyázni kell, hogy megfelelő kódolású legyen az XML fájl, mert az ékezetes betűket UTF-8 kódolással nem fogja tudni értelmezni a PowerShell.

Az XML fájl tartalmát a következő kifejezéssel tudjuk változóba tölteni:

```
[9] PS C:\> $xml = [xml] (get-content c:\powershell2\munka\emberek.xml)
```

Ha megnézzük ennek az `$xml` változónak a tartalmát, akkor már nem a fájlban tárolt karaktersorozatot kapjuk vissza, hanem az XML adatformátumnak megfelelő adattartalmakat, és ebből a *1.4.14 PSBase, PSAdapted, PSExtended, PSObject* fejezetben látottaknak megfelelően lehet az adatokat kiolvasni:

```
[22] PS C:\> $xml

xml                                 root
---                                 ----
                                    root
[23] PS C:\> $xml.root

xsd                 od                  schema              dataroot
---                 --                  ------              --------
http://www.w3.o...  urn:schemas-mic...  schema              dataroot


[24] PS C:\> $xml.root.dataroot

xsi                 generated           Emberek             Városok
---                 ---------           -------             -------
http://www.w3.o...  2008-08-10T08:3...  {1, 2, 3}           {1, 2}


[25] PS C:\> $xml.root.dataroot.emberek

ID                  Név                 Mellék              VárosID
--                  ---                 ------              -------
1                   Soós Tibor          1234                1
2                   Fájdalom Csilla     1230                1
3                   Beléd Márton        1299                2
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[26] PS C:\> $xml.root.dataroot.városok

ID                                      Név
--                                      ---
1                                       Budapest
2                                       Debrecen
```

Vajon hogyan lehetne a „nyers” XML állománnyá visszaalakítani az `$xml` változónkat? Ebben az `[XML]` adattípus `Save` metódusa segíthet. Alapvetően ez a mentés fájlba irányulna, egy elérési utat vár paraméterként. Ha a képernyőre akarjuk kiíratni, akkor át kell verni, a mentés helyének a konzolt kell megadni a következő módon:

```
[26] PS C:\> $xml.save([console]::out)
<?xml version="1.0" encoding="ibm852"?>
<root xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:od="urn:schemas-mic
rosoft-com:officedata">
  <xsd:schema>
    <xsd:element name="dataroot">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="Emberek" minOccurs="0" maxOccurs="unbounded" />
...
```

2.5.9 Megosztások és webmappák elérése

A megosztások elérése nagyon egyszerű a PowerShellből: egyszerűen a UNC névvel kell hivatkozni a megosztott könyvtárakra és az ott található fájlokra:

```
[2] PS C:\> Get-ChildItem \\asus\powershell


    Directory: Microsoft.PowerShell.Core\FileSystem::\\asus\powershell


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
d----   2008.05.04.    12:55        <DIR> cd
d----   2008.07.15.    20:01        <DIR> Copy of scripts
d----   2008.02.28.    21:14        <DIR> demo
```

Sőt! A TAB-kiegészítés is működik! Valamint az ebben a fejezetben leírt összes cmdlet is működik az UNC nevekkel is, például:

```
[4] PS C:\> get-acl \\asus\powershell\munka\tömbök.txt


    Directory: Microsoft.PowerShell.Core\FileSystem::\\asus\powershell\munk
    a


Path                    Owner                   Access
----                    -----                   ------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
tömbök.txt                  ASUS\Administrator       BUILTIN\Administrator...
```

Megosztások létrehozása már nem ilyen egyszerű, de ehhez is van megoldás, melyet a *2.11.3 WMI objektumok metódusainak meghívása* fejezetben mutatok egy WMI kifejezés segítségével.

Mindezen kívül a weben megosztott mappák is elérhetők ezekkel a cmdletekkel:

```
[5] PS C:\> get-childitem \\live.sysinternals.com\Tools


    Directory: Microsoft.PowerShell.Core\FileSystem::\\live.sysinternals.co
    m\Tools


Mode           LastWriteTime       Length Name
----           -------------       ------ ----
d---- 2008. 06. 02.     1:16        <DIR> WindowsInternals
-a--- 2008. 05. 30.    17:55          668 About_This_Site.txt
-a--- 2008. 07. 15.    17:39       229928 accesschk.exe
-a--- 2006. 11. 01.    14:06       174968 AccessEnum.exe
-a--- 2006. 11. 01.    22:05       121712 accvio.EXE
-a--- 2007. 07. 12.     7:26        50379 AdExplorer.chm
-a--- 2007. 11. 26.    13:21       422952 ADExplorer.exe
-a--- 2007. 11. 07.    10:13       401616 ADInsight.chm
-a--- 2007. 11. 20.    13:25      1049640 ADInsight.exe
-a--- 2006. 11. 01.    14:05       150328 adrestore.exe
-a--- 2006. 11. 01.    14:06       154424 Autologon.exe
-a--- 2008. 08. 20.    15:18        48986 autoruns.chm
...
```

A könyv tanfolyami felhasz

2.6 Az Eseménynapló feldolgozása

Az egyik legfontosabb információforrás a rendszergazdák számára az eseménynapló, de egyben a legnehézkesebben kezelhető a beépített Event Viewer eszközzel. A PowerShell ebben a tekintetben is nagyon jó szolgálatot tesz nekünk, mint ahogy a következőkben láthatjuk.

2.6.1 Hagyományos eseménynaplók kezelése (Get-EventLog)

A „hagyományos” Eseménynapló bejegyzéseinek kiolvasására a `Get-Eventlog` cmdletet használhatjuk, ami `EventLogEntry` (vagy `EventLog`) típusú objektumokat ad vissza. Először is tudjuk meg, hogy milyen naplók vannak a gépünkön:

```
[28] PS C:\> Get-EventLog -List

  Max(K) Retain OverflowAction        Entries Log
  ------ ------ --------------        ------- ---
     512      7 OverwriteOlder             36 Active Directory Web Services
  20 480      0 OverwriteAsNeeded         929 Application
  15 168      0 OverwriteAsNeeded          29 DFS Replication
     512      0 OverwriteAsNeeded         102 Directory Service
  16 384      0 OverwriteAsNeeded          23 DNS Server
     512      0 OverwriteAsNeeded          18 File Replication Service
  20 480      0 OverwriteAsNeeded           0 HardwareEvents
     512      7 OverwriteOlder              0 Internet Explorer
  20 480      0 OverwriteAsNeeded           0 Key Management Service
 131 072      0 OverwriteAsNeeded      65 291 Security
  20 480      0 OverwriteAsNeeded       2 102 System
  15 360      0 OverwriteAsNeeded       2 902 Windows PowerShell
```

A fenti parancs `EventLog` objektumokat ad vissza, ezektől lekérdezhetők az adott napló különféle tulajdonságai, maximális mérete, bejegyzéseinek száma, stb.

```
[29] PS C:\> (get-eventlog -list)[0] | Format-List *


Entries               : {dc.r2.dom, dc.r2.dom, dc.r2.dom, dc.r2.dom...}
LogDisplayName        : Active Directory Web Services
Log                   : Active Directory Web Services
MachineName           : .
MaximumKilobytes      : 512
OverflowAction        : OverwriteOlder
MinimumRetentionDays  : 7
EnableRaisingEvents   : False
SynchronizingObject   :
Source                :
Site                  :
Container             :
[30] PS C:\> (get-eventlog -list)[0].Entries.Count
36
```

Ha egy konkrét napló bejegyzéseire vagyunk kíváncsiak, akkor a napló nevét kell megadnunk paraméterként, a `-newest` paraméter után álló szám pedig a listába kerülő bejegyzések számát korlátozza. Az következő parancs a rendszernapló legutóbbi három bejegyzését fogja kiolvasni:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[31] PS C:\> Get-EventLog -LogName System -Newest 3

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
    2102 jan. 16 00:42 Warning     Microsoft-Windows...         1014 Name r...
    2101 jan. 16 00:41 Information Service Control M...   1073748860 The Wi...
    2100 jan. 16 00:41 Information Microsoft-Windows...            1 The de...
```

Az előző két parancs kombinálásával valamennyi napló legfrissebb bejegyzéseit is könnyen lekérdezhetjük egyetlen paranccsal. A bal oldali `Get-Eventlog` szállítja a naplók objektumait, az innen származó adatokat fogjuk odaadni paraméterként a `Foreach-Object` belsejében elinduló `Get-Eventlog`-nak, így szép sorban valamennyi napló bejegyzései elő fognak kerülni. (Közben még kiírjuk a napló nevét is, hogy tudjuk hol járunk.)

```
[41] PS C:\> Get-Eventlog -list | Foreach-Object {Write-Host $_.LogDisplayName;
 if($_.entries.count){Get-EventLog -LogName $_.LogDisplayName -Newest 3}}
Active Directory Web Services

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
      36 dec. 31 15:14 Information ADWS                   1073742828 Active...
      35 dec. 31 15:14 Information ADWS                   1073742830 Active...
      34 dec. 31 15:14 Information ADWS                   1073743024 Active...
Application
     929 jan. 16 00:42 Information SceCli                 1073743528 Securi...
     928 jan. 14 21:23 Information SceCli                 1073743528 Securi...
     927 jan. 13 21:07 Information HHCTRL                       1904 The de...
DFS Replication
      29 dec. 31 15:15 Information DFSR                   1073743030 The DF...
      28 dec. 31 15:14 Information DFSR                   1073747926 The DF...
      27 dec. 31 15:14 Information DFSR                   1073743138 The DF...
Directory Service
     102 jan. 11 21:29 Information NTDS ISAM                     701 NTDS (...
     101 jan. 11 21:29 Information NTDS ISAM                     700 NTDS (...
     100 jan. 07 21:40 Information NTDS ISAM                     701 NTDS (...
DNS Server
      23 jan. 06 06:55 Error       DNS                    3221229488 The DN...
      22 dec. 31 15:14 Information DNS                    1073741826 The DN...
      21 dec. 31 15:14 Information DNS                    1073741828 The DN...
File Replication Service
      18 dec. 31 15:14 Information NtFrs                  1073755340 The Fi...
      17 dec. 31 15:14 Warning     NtFrs                  2147497160 The Fi...
      16 dec. 31 15:14 Information NtFrs                  1073755325 The Fi...
Hardware Events
Internet Explorer
Key Management Service
Security
   65364 jan. 16 00:59 SuccessA... Microsoft-Windows...         4634 An acc...
   65363 jan. 16 00:59 SuccessA... Microsoft-Windows...         4624 An acc...
   65362 jan. 16 00:59 SuccessA... Microsoft-Windows...         4672 Specia...
System
    2102 jan. 16 00:42 Warning     Microsoft-Windows...         1014 Name r...
    2101 jan. 16 00:41 Information Service Control M...   1073748860 The Wi...
    2100 jan. 16 00:41 Information Microsoft-Windows...            1 The de...
Windows PowerShell
    2902 jan. 12 22:31 Information PowerShell                   400 Engine...
    2901 jan. 12 22:31 Information PowerShell                   600 Provid...
    2900 jan. 12 22:31 Information PowerShell                   600 Provid...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A belső `IF` vizsgálatra azért volt szükség, mert a naplóbejegyzések lekérése hibát adna abban az esetben, ha egy eseménynaplóban nincsen egy bejegyzés sem.

Természetesen a megszokott eszközök segítségével tetszés szerint szűrhetjük és formázhatjuk is a naplókból nyert objektumokat. Kiválogathatjuk például csak a hibákat vagy a figyelmeztetéseket, és válogathatunk a kiírandó jellemzők között. Például listázzuk ki a rendszernaplóból az utolsó héten keletkezett „Warning” típusú utolsó három bejegyzést időrend szerint:

```
[45] PS C:\> Get-EventLog -LogName System -EntryType warning -Newest 3 -After (
get-date).adddays(-7)

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
    2102 jan. 16 00:42 Warning     Microsoft-Windows...         1014 Name r...
    2098 jan. 14 22:53 Warning     NETLOGON                     5782 Dynami...
    2096 jan. 14 21:23 Warning     Microsoft-Windows...           16 Unable...
```

Nézzünk meg egy konkrét naplóbejegyzést részletesebben:

```
[24] PS C:\> (Get-EventLog -LogName application -InstanceID 1040)[0] | Format-L
ist *


EventID            : 1040
MachineName        : dc.r2.dom
Data               : {}
Index              : 816
Category           : (0)
CategoryNumber     : 0
EntryType          : Information
Message            : Beginning a Windows Installer transaction: \\192.168.1.20
                     0\c$\_munka\powershell2\tools\PowerGUI.1.9.5.966.msi. Cli
                     ent Process Id: 2568.
Source             : MsiInstaller
ReplacementStrings : {\\192.168.1.200\c$\_munka\powershell2\tools\PowerGUI.1.9
                     .5.966.msi, 2568, (NULL), (NULL)...}
InstanceId         : 1040
TimeGenerated      : 2009. 12. 31. 15:19:40
TimeWritten        : 2009. 12. 31. 15:19:40
UserName           : R2\Administrator
Site               :
Container          :
```

Látható, hogy az üzenet (`Message`) rész kétfajta részből áll össze: egy statikus, pusztán az EventID-tól függő szövegrészből, és az ebbe illeszkedő adott géptől, időtől és egyéb körülménytől függő behelyettesítő adatokból. Ez utóbbiak külön is megtalálhatók a `ReplacementStrings` tulajdonság-tömbben. Vajon ezt hogyan tudjuk praktikusan felhasználni? Például nézzük, hogy hogyan lehet a Security log információi alapján egy olyan táblázatot készíteni, hogy ki mikor lépett be a tartományba. Ehhez talán a következő naplóbejegyzés megkeresése a legpraktikusabb:

```
[51] PS C:\> Get-EventLog -LogName Security -InstanceId 4768 | fl *


EventID            : 4768
MachineName        : dc.r2.dom
Data               : {}
```

```
Index              : 66017
Category           : (14339)
CategoryNumber     : 14339
EntryType          : SuccessAudit
Message            : A Kerberos authentication ticket (TGT) was requested.

                     Account Information:
                         Account Name:         SoosTibor
                         Supplied Realm Name:    R2
                         User ID:            S-1-5-21-3398938913-3940250523-92
                     7435294-1178

                     Service Information:
                         Service Name:         krbtgt
                         Service ID:         S-1-5-21-3398938913-3940250523-927
                     435294-502

                     Network Information:
                         Client Address:         ::ffff:192.168.1.11
                         Client Port:         49257

                     Additional Information:
                         Ticket Options:         0x40810010
                         Result Code:         0x0
                         Ticket Encryption Type:     0x12
                         Pre-Authentication Type:     2

                     Certificate Information:
                         Certificate Issuer Name:
                         Certificate Serial Number:
                         Certificate Thumbprint:

                     Certificate information is only provided if a certificate
                      was used for pre-authentication.

                     Pre-authentication types, ticket options, encryption type
                     s and result codes are defined in RFC 4120.
Source             : Microsoft-Windows-Security-Auditing
ReplacementStrings : {SoosTibor, R2, S-1-5-21-3398938913-3940250523-927435294-
                     1178, krbtgt...}
InstanceId         : 4768
TimeGenerated      : 2010. 01. 16. 10:08:11
TimeWritten        : 2010. 01. 16. 10:08:11
UserName           :
Site               :
Container          :
```

Látható, hogy a felhasználó neve legegyszerűbben a `ReplacementStrings` tulajdonság-tömbből szedhető ki. Nézzük ezt meg részletesebben:

```
[52] PS C:\> (Get-EventLog -LogName Security -InstanceId 4768)[0].replacementst
rings | fl *
SoosTibor
R2
S-1-5-21-3398938913-3940250523-927435294-1178
krbtgt
S-1-5-21-3398938913-3940250523-927435294-502
0x40810010
0x0
0x12
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
2
::ffff:192.168.1.11
49257
```

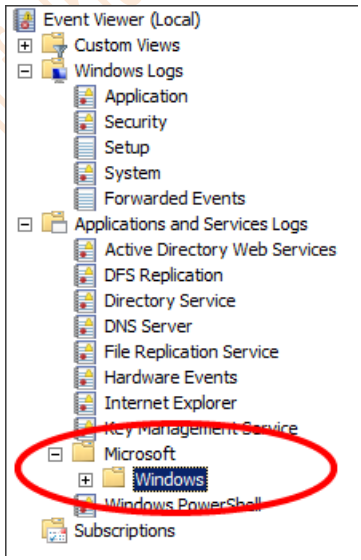
Látható, hogy az első elem a felhasználónév és a 10. elemben meg megtalálható a felhasználó számítógépének IP címe. Készítsünk ezek alapján egy speciális táblázatot:

```
[62] PS C:\> Get-EventLog -LogName Security -InstanceId 4768,4771 | Select-Obje
ct -Property @{n="user";e={$_.replacementstrings[0]}}, entrytype, timegenerated
, @{n="IP";e={$_.replacementstrings[9].substring(7)}} | ft -AutoSize

user            EntryType TimeGenerated          IP
----            --------- -------------          --
NagyBela     FailureAudit 2010. 01. 16. 10:19:50
SoosTibor    SuccessAudit 2010. 01. 16. 10:08:11 192.168.1.11
NemecsekBela SuccessAudit 2010. 01. 16. 10:04:13 192.168.1.11
SulyokIstvan SuccessAudit 2010. 01. 16. 10:03:39 192.168.1.11
SoosTibor    SuccessAudit 2010. 01. 16. 10:03:03 192.168.1.11
```

Ebbe belevettem még a sikertelen bejelentkezésre utaló 4771-es bejegyzést is. Látható, hogy sokkal áttekinthetőbben jeleníttethetők meg így az adatok, mint az Event Viewer eszköz segítségével.

2.6.2 Az új alkalmazás- és szolgáltatásnaplók kezelése (Get-WinEvent)

A Windows Vista / Server 2008 verziók óta újabb eseménynaplókkal bővült a rendszer. Az alábbi képen látható, hogy az Event Viewer alkalmazásban ezek hol találhatók:



65. ábra Alkalmazás- és szolgáltatásnaplók

Ez egy nagyon sok naplót tartalmazó rész, amelyeket a `Get-EventLog` nem kezel, ezzel szemben a `Get-WinEvent` igen. Ez a cmdlet kicsit más szintaxissal és lehetőségekkel rendelkezik. Nézzünk pár példát a használatára. A lehetséges napló kategóriák kilistázásához a `-ListLog` paraméter használható, viszont kötelező valamit megadni, akár egy *-ot:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[5] PS C:\> Get-WinEvent -ListLog *

LogName              MaximumSizeInBytes         RecordCount            LogMode
-------              ------------------         -----------            -------
Active Directory...             1052672                  36          Circular
Application                    20971520                 931          Circular
…
```

Nézzük meg egy konkrét naplókategória jellemzőit:

```
[31] PS C:\> Get-WinEvent -Listlog "Microsoft-Windows-WinRM/Operational" | fl *



FileSize                       : 1052672
IsLogFull                      : False
LastAccessTime                 : 2009. 11. 09. 21:13:04
LastWriteTime                  : 2009. 12. 31. 15:14:07
OldestRecordNumber             : 77489
RecordCount                    : 2500
LogName                        : Microsoft-Windows-WinRM/Operational
LogType                        : Operational
LogIsolation                   : Application
IsEnabled                      : True
IsClassicLog                   : False
SecurityDescriptor             : O:BAG:SYD:(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0
                                 x7;;;SO)(A;;0x3;;;IU)(A;;0x3;;;SU)(A;;0x3;;;S
                                 -1-5-3)(A;;0x3;;;S-1-5-33)(A;;0x1;;;S-1-5-32-
                                 573)
LogFilePath                    : %SystemRoot%\System32\Winevt\Logs\Microsoft-W
                                 indows-WinRM%4Operational.evtx
MaximumSizeInBytes             : 1052672
LogMode                        : Circular
OwningProviderName             : Microsoft-Windows-WinRM
ProviderNames                  : {Microsoft-Windows-WinRM}
ProviderLevel                  :
ProviderKeywords               :
ProviderBufferSize             : 64
ProviderMinimumNumberOfBuffers : 0
ProviderMaximumNumberOfBuffers : 64
ProviderLatency                : 1000
ProviderControlGuid            :
```

Egyes logok bejegyzéseinek listázásához szűrőfeltételek akár hashtábla formátumban is megadhatók:

```
[18] PS C:\> Get-WinEvent -FilterHashtable @{id = 169; logname="Microsoft-Windo
ws-WinRM/Operational"}

TimeCreated          ProviderName                        Id Message
-----------          ------------                        -- -------
2009. 12. 13. 12... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 22... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 22... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 22... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 22... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 21... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 21... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 21... Microsoft-Window...                169 User R2\hh auth...
2009. 12. 07. 21... Microsoft-Window...                169 User R2\hh auth...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
2009. 12. 07. 21... Microsoft-Window...                169 User R2\Adminis...
2009. 12. 07. 21... Microsoft-Window...                169 User R2\Adminis...
```

Látható, hogy itt kicsit másak az esemény tulajdonságainak a nevei, például nem `EventID`, hanem csak `ID`. Nézzük meg, hogy egy bejegyzés hogyan néz ki részletesen:

```
[19] PS C:\> (Get-WinEvent -FilterHashtable @{id = 169; logname="Microsoft-Wind
ows-WinRM/Operational"})[0] | fl *


Message               : User R2\hh authenticated successfully using Kerberos au
                        thentication
Id                    : 169
Version               : 0
Qualifiers            :
Level                 : 4
Task                  : 7
Opcode                : 0
Keywords              : 4611686018427387916
RecordId              : 79980
ProviderName          : Microsoft-Windows-WinRM
ProviderId            : a7975c8f-ac13-49f1-87da-5a984a4ab417
LogName               : Microsoft-Windows-WinRM/Operational
ProcessId             : 248
ThreadId              : 2780
MachineName           : dc.r2.dom
UserId                : S-1-5-20
TimeCreated           : 2009. 12. 13. 12:34:24
ActivityId            : 00000100-0000-0000-1f05-9ad1b17aca01
RelatedActivityId     :
ContainerLog          : microsoft-windows-winrm/operational
MatchedQueryIds       : {}
Bookmark              : System.Diagnostics.Eventing.Reader.EventBookmark
LevelDisplayName      : Information
OpcodeDisplayName     : Info
TaskDisplayName       : User authentication
KeywordsDisplayNames  : {Server, Security}
Properties            : {System.Diagnostics.Eventing.Reader.EventProperty, Syst
                        em.Diagnostics.Eventing.Reader.EventProperty}
```

Látható az is, hogy itt nem a `ReplacementStrings` tulajdonság, hanem a `Properties` tartalmazza a futás időben generálódó információkat. Nézzük meg ezt is:

```
[20] PS C:\> (Get-WinEvent -FilterHashtable @{id = 169; logname="Microsoft-Wind
ows-WinRM/Operational"})[0].properties

Value
-----
R2\hh
Kerberos
```

Innentől már hasonlóan lehet például informatív táblázatokat készíteni. Itt most konkrétan a WinRM szolgáltatás igénybevételével kapcsolatos statisztikát készítettem az eseménynapló bejegyzései alapján:

```
[27] PS C:\> Get-WinEvent -FilterHashtable @{id = 169; logname="Microsoft-Windo
ws-WinRM/Operational"} | Select-Object -Property TimeCreated, @{n="User";e={$_.
properties[0].value}}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
TimeCreated                           User
-----------                           ----
2009. 12. 13. 12:34:24                R2\hh
2009. 12. 07. 22:06:02                R2\hh
2009. 12. 07. 22:02:23                R2\hh
2009. 12. 07. 22:00:51                R2\hh
2009. 12. 07. 22:00:15                R2\hh
2009. 12. 07. 21:58:47                R2\hh
2009. 12. 07. 21:58:19                R2\hh
2009. 12. 07. 21:54:08                R2\hh
2009. 12. 07. 21:54:02                R2\hh
2009. 12. 07. 21:49:21                R2\Administrator
2009. 12. 07. 21:49:12                R2\Administrator
```

Vagy hasonló módon lehet kigyűjteni azt, hogy melyik frissítés mikor települt:

```
[35] PS C:\> Get-WinEvent -FilterHashTable @{ProviderName="Microsoft-Windows-Wi
ndowsUpdateClient"; ID=19} | Select-Object -Property timecreated, @{n="Patch";e
={$ .properties[0].value}} | ft -AutoSize

TimeCreated            Patch
-----------            -----
2009. 12. 28. 18:39:06 Windows Malicious Software Removal Tool x64 - Decemb...
2009. 12. 11. 23:31:19 Cumulative Security Update for Internet Explorer 8 f...
2009. 11. 25. 9:13:12  Update for Windows Server 2008 R2 x64 Edition (KB976...
2009. 11. 18. 8:59:01  Update for Internet Explorer 8 for Windows Server 20...
2009. 11. 09. 22:22:22 Security Update for Windows Server 2008 R2 x64 Editi...
2009. 11. 09. 22:22:22 Security Update for ActiveX Killbits for Windows Ser...
2009. 11. 09. 22:22:22 Security Update for Internet Explorer 8 for Windows ...
2009. 11. 09. 22:22:22 Update for Windows Server 2008 R2 x64 Edition (KB974...
2009. 11. 09. 22:22:22 Update for Windows Server 2008 R2 x64 Edition (KB974...
2009. 11. 09. 22:22:22 Security Update for Windows Server 2008 R2 x64 Editi...
2009. 11. 09. 22:22:22 Update for Internet Explorer 8 Compatibility View Li...
```

2.6.3 Távoli gépek eseménynaplóinak megtekintése

Mind a `get-eventlog`, mind a `get-winevent` cmdletek távoli gépek eseménynaplóinak megtekintésére is alkalmas a `–computername` paraméter használatával:

```
[95] PS C:\> Get-EventLog -ComputerName member,dc -LogName system -Newest 3

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
    1216 jan. 17 19:13 Information Microsoft-Windows...          35 The ti...
    1215 jan. 17 19:13 Information Service Control M...  1073748860 The Ap...
    1214 jan. 17 19:13 Warning     Microsoft-Windows...        1014 Name r...
    2124 jan. 17 19:14 Warning     Microsoft-Windows...          16 Unable...
    2123 jan. 17 19:12 Warning     Microsoft-Windows...        1014 Name r...
    2122 jan. 17 19:11 Information Service Control M...  1073748860 The Wi...
```

Látható a fenti példában, hogy a `get-eventlog` akár egyszerre több gép eseménynaplóját is el tudja érni. A `get-winevent` sajnos csak egyszerre egy gépét:

```
[100] PS C:\> Get-WinEvent -ComputerName member -MaxEvents 4
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
TimeCreated          ProviderName                    Id Message
-----------          ------------                    -- -------
2010. 01. 17. 19... Microsoft-Window...           4624 An account was ...
2010. 01. 17. 19... Microsoft-Window...           4672 Special privile...
2010. 01. 17. 19... Service Control ...           7036 The Windows Rem...
2010. 01. 17. 19... Microsoft-Window...            209 The Winrm servi...
```

2.6.4 Eseménynaplóval kapcsolatos egyéb műveletek

Az eseménynaplóval kapcsolatos leggyakoribb műveletek a bejegyzések közti keresés és az adatok megjelenítése, de a PowerShell további cmdleteket is tartalmaz ezzel kapcsolatban.

Az egyik ilyen cmdlet a `Show-EventLog`. Ez a parancs nem csinál sok mindent, egyszerűen megnyitja a grafikus EventViewer alkalmazást, egyetlen paramétere a számítógép-név, azaz hogy mely számítógép eseménynaplóját kívánjuk megnyitni.

A következő cmdlet a `Limit-EventLog`. Ezzel az egyes eseménynaplók maximális méretét állíthatjuk be, illetve azt, hogy a betelés esetén mi történjen: felülírja a régebbi bejegyzéseket, csak bizonyos korú bejegyzéseket írjon felül, stb.

Van még a `New-EventLog`, mellyel új eseménynaplót tudunk létrehozni. Ezzel a paranccsal klasszikus formátumú új eseménynapló hozható létre, mely a többi `EventLog` főnevet tartalmazó cmdlettel kezelhető. Ahhoz, hogy ilyen új eseménynaplót hozhassunk létre a naplóbejegyzések statikus szövegeit és tulajdonságneveit tartalmazó erőforrás dll-t is meg kell adnunk, valamint azt, hogy mi lesz az ide beírni kívánt bejegyzések forrása.

Az előző cmdlet párja a `Remove-EventLog`, mellyel eseménynaplót vagy forrást lehet megszűntetni.

Ennél szelídebb és talán gyakrabban használatos a `Clear-EventLog`, mellyel a klasszikus eseménynaplókat lehet üríteni, akár távoli gépen is.

Ami még praktikusabb, az a `Write-EventLog`. Segítségével bejegyzéseket írhatunk az eseménynaplókba. Nézzünk erre példát is:

```
[59] PS C:\> Write-EventLog -LogName application -EntryType information -Source
 PowerShellScript -Category 1 -EventId 10000 -Message "A szkriptem elindult" -R
awData 1,2
Write-EventLog : The source name "PowerShellScript" does not exist on computer
 "localhost".
At line:1 char:15
+ Write-EventLog <<<<  -LogName application -EntryType information -Source Pow
erShellScript -Category 1 -EventId 10000 -Message "A szkriptem elindult" -RawD
ata 1,2
    + CategoryInfo          : InvalidOperation: (:) [Write-EventLog], Invalid
   OperationException
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteEventLogComm
   and
```

Hoppá! Ez hibát adott. A hiba oka, hogy az Application naplóhoz még nem lett létrehozva lehetséges forrásként az általam megadott „PowerShellScript”. Pótoljuk ezt a hiányosságot, sajnos nem PowerShell cmdlettel, hanem .NET osztály statikus metódusával:

```
[61] PS C:\> [System.Diagnostics.EventLog]::CreateEventSource("PowerShellScript
","Application")
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Akkor most újra próbálom a bejegyzésemet beírni az eseménynaplóba:

```
[62] PS C:\> Write-EventLog -LogName application -EntryType information -Source
 PowerShellScript -Category 1 -EventId 10000 -Message "A szkriptem elindult" -R
awData 1,2
[63] PS C:\> Get-EventLog -LogName Application -Newest 1 | fl *


EventID            : 10000
MachineName        : dc.r2.dom
Data               : {1, 2}
Index              : 989
Category           : (1)
CategoryNumber     : 1
EntryType          : Information
Message            : A szkriptem elindult
Source             : PowerShellScript
ReplacementStrings : {A szkriptem elindult}
InstanceId         : 10000
TimeGenerated      : 2010. 01. 23. 22:45:23
TimeWritten        : 2010. 01. 23. 22:45:23
UserName           :
Site               :
Container          :
```

Na így már sikerült!

Megjegyzés

Azt, hogy egy naplóhoz milyen források vannak már bejegyezve egy registry kulcs segítségével nézhetjük meg:

```
[66] PS C:\> dir HKLM:\SYSTEM\CurrentControlSet\Services\Eventlog\Application\


    Hive: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Applic
    ation


SKC  VC Name                           Property
---  -- ----                           --------
  0   2 .NET Runtime                   {TypesSupported, EventMessageFile}
  0   2 .NET Runtime Optimization S... {TypesSupported, EventMessageFile}
  0   2 Application                    {CategoryCount, CategoryMessageFile}
  0   4 Application Error              {EventMessageFile, TypesSupported, C...
…
  0   1 PerfProc                       {ProviderGuid}
  0   1 PowerShellScript               {EventMessageFile}
  0   1 Process Exit Monitor           {providerGuid}
…
```

A fenti listában látható az általam bejegyzett PowerShellScript forrás.

Az általunk beírt eseménynapló bejegyzésekkel például két külön ablakban futó PowerShell szkript egymás közti kommunikációját is megoldhatjuk, vagy a Task Scheduler segítségével a mi általunk definiált

eseménynapló-bejegyzés megjelenésére tudunk indíttatni egy parancssort, többek között PowerShell szkriptet.

A könyv tanfolyami felhasználása nem engedélyezett!

2.7 Registry kezelése

Amikor a PowerShell egyszerűségét demonstrálják különböző szakmai rendezvényeken, akkor gyakran a registry kezelését is bemutatják, hiszen jól érzékeltethető az, hogy a fájlok kezelésével kapcsolatos cmdletek legtöbbje a registryre is használható, mivel a fájlok és a registry is un. PSDrive-ként érhető el. A személyes tapasztalatom az, hogy azért viszonylag ritkán kell a registryt szkriptből machinálni, de azért foglalkozzunk ezzel is.

2.7.1 Olvasás a registryből

Elsőként nézzük meg, hogy hogyan lehet olvasni a registryből. Például listázzuk ki a registry alapján a számítógépre telepített alkalmazások nevét!

Miután a registryt is egy PSDrive-on keresztül érhetjük el, érdemes lehet akár új, a minket érdeklő információkat tartalmazó ágra új „shortcut” PSDrive-ot definiálni, hogy később már rövidebb elérési úttal is hivatkozhatók legyenek az egyes elemek:

```
[1] PS I:\>New-PSDrive -Name Uninstall -PSProvider Registry -Root HKLM:\SOFT
WARE\Microsoft\Windows\CurrentVersion\Uninstall

Name        Provider      Root                                    CurrentLocatio
                                                                          n
----        --------      ----                                    --------------
Uninstall   Registry      HKEY_LOCAL_MACHINE\SOFTWARE\Micr...
```

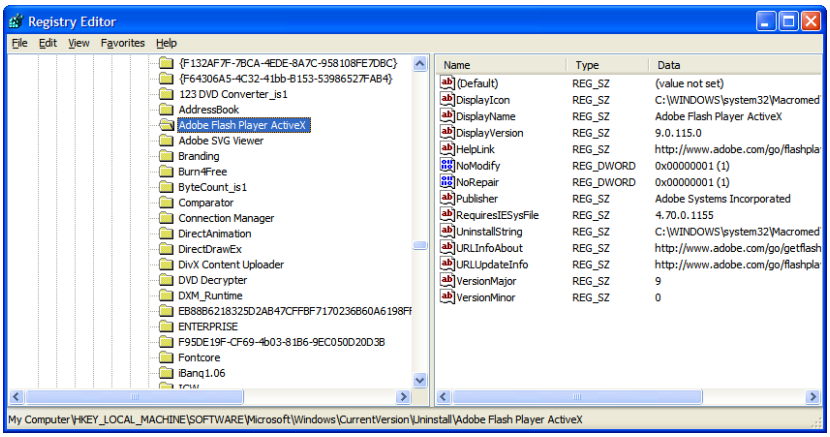
A mostani feladatunkban például az `Uninstall` registry kulcsot egy új, „Uninstall” nevű meghajtón keresztül lehet elérni. Erre már nagyon egyszerűen végezhetünk szokásos fájlműveleteket, például kilistázhatjuk az itt található kulcsokat a `Get-ChildItem` cmdlet segítségével:

```
[2] PS I:\>Get-ChildItem uninstall:


   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Mi
crosoft\Windows\CurrentVersion\Uninstall

SKC  VC Name                           Property
---  -- ----                           --------
  0  16 123 DVD Converter_is1          {Inno Setup: Setup Version, Inno ...
  0   1 AddressBook                    {(default)}
  0  13 Adobe Flash Player ActiveX     {DisplayName, DisplayVersion, Pub...
  0  12 Adobe SVG Viewer               {DisplayName, DisplayVersion, Dis...
  0   2 Branding                       {QuietUninstallString, RequiresIE...
  0   2 Burn4Free                      {DisplayName, UninstallString}
...
```

Nézzük meg ugyanezt a regedittel:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



66. ábra Uninstall registry-ág

Látható, hogy amit mi a Regeditben látunk „fájlként”, azaz az Adobe Flash Playernél például a `DisplayName`, az a PowerShell-ben nem „fájl”, hanem az a fájlnak, ami az „Adobe Flash Player ActiveX”-nek felel meg, a tulajdonsága (property). Ez egy kicsit zavarónak tűnhet, hiszen a szemünk előtt az „Adobe Flash Player ActiveX” mint mappa lebeg, holott a PowerShellben az maga a fájlszintű objektum a hierarchiában. De akkor hogyan jutunk hozzá az egyes tulajdonságokhoz egyszerűen? Erre is van cmdlet, a `Get-ItemProperty`:

```
[3] PS I:\>cd Uninstall:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
[4] PS Uninstall:\>Get-ItemProperty "Adobe Flash Player ActiveX"


PSPath            : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\
                    SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Ado
                    be Flash Player ActiveX
PSParentPath      : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\
                    SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
PSChildName       : Adobe Flash Player ActiveX
PSDrive           : Uninstall
PSProvider        : Microsoft.PowerShell.Core\Registry
DisplayName       : Adobe Flash Player ActiveX
DisplayVersion    : 9.0.115.0
Publisher         : Adobe Systems Incorporated
URLInfoAbout      : http://www.adobe.com/go/getflashplayer
VersionMajor      : 9
VersionMinor      : 0
HelpLink          : http://www.adobe.com/go/flashplayer_support/
URLUpdateInfo     : http://www.adobe.com/go/flashplayer/
DisplayIcon       : C:\WINDOWS\system32\Macromed\Flash\uninstall_activeX.ex
                    e
UninstallString   : C:\WINDOWS\system32\Macromed\Flash\uninstall_activeX.ex
                    e
RequiresIESysFile : 4.70.0.1155
NoModify          : 1
NoRepair          : 1
```

A [4]-es sorban lekérdezem az *Adobe Flash Player* elem tulajdonságait. Egy kicsit többet is visszaad nekem a PowerShell, mint ami ténylegesen a registryben található: `PSPath`, `PSPartentPath`, stb. Ezek természetesen nincsenek benne a registryben, ezeket a PowerShell típusadaptációs rétege teszi hozzá. Sajnos

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

ezeket a plusz adatokat akkor is hozzábiggyeszti, ha csak mondjuk a `DisplayName` paramétert akarjuk kiolvasni:

```
[14] PS Uninstall:\>Get-ItemProperty "Adobe Flash Player ActiveX" -name Disp
layName


PSPath       : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTW
               ARE\Microsoft\Windows\CurrentVersion\Uninstall\Adobe Flash P
               layer ActiveX
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTW
               ARE\Microsoft\Windows\CurrentVersion\Uninstall
PSChildName  : Adobe Flash Player ActiveX
PSDrive      : Uninstall
PSProvider   : Microsoft.PowerShell.Core\Registry
DisplayName  : Adobe Flash Player ActiveX
```

Szóval trükközni kell, ha tényleg csak az adott kulcs értékét akarom megtekinteni:

```
[24] PS Uninstall:\>(Get-ItemProperty "Adobe Flash Player ActiveX").DisplayN
ame
Adobe Flash Player ActiveX
```

Azaz külön hivatkozni kell az adott kulcs nevére, nem igazán jó a `-name` paraméter használata. Ez függvényért kiált, de ennek megvalósítását az olvasóra bízom.

Még két lehetőségre hívnám fel a figyelmet a registry kulcsok olvasásával kapcsolatban. Az egyik `GetValueNames()` metódus, amellyel az adott kulcs tényleges értékeinek neveit lehet kiolvasni. A nevek birtokában aztán például valamelyik ciklussal lehet műveleteket végezni az értékekkel. Erre példaként nézzünk egy szkriptet, amely összeszámolja, hogy gépünkre hány font van telepítve, és abból mennyi a *TrueType* típusú:

```
[15] PS C:\> $ttf=0; (get-item "hklm:\\Software\Microsoft\Windows NT\Current
Version\Fonts").GetValueNames() | Where-Object {$_.contains("TrueType")} | F
orEach-Object {$ttf++}; $ttf
277
```

Ugyanezt a funkciót valósítottam meg következő szkriptben is, csak itt nem a `GetValueNames()` metódust alkalmaztam, hanem a PowerShell `Select-Object` cmdletjét, az `ExtendProperty` paraméterrel, mellyel egy adott objektum egy adott, tömböt tartalmazó tulajdonságának értékeit lehet kifejteni:

```
[22] PS C:\> $ttf=0; Get-Item "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVe
rsion\Fonts" | Select-Object -ExpandProperty Property | where-object {$_.con
tains("TrueType")} | ForEach-Object {$ttf++}; $ttf
277
```

2.7.2 Registry elemek létrehozása, módosítása

Ezek után nem meglepő, hogy a registry elemek létrehozása is hasonló módon történik, mint a fájlok létrehozása. Megint fontos tudatosítani, hogy mi a „fájlszintű” objektum a registryben, és mi a tulajdonság.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Nézzük egy új tulajdonság létrehozását, hozzunk létre az Outlook törölt elemek visszaállíthatóságát megkönnyítő kulcsot:

```
[2] PS I:\>Set-Location HKLM:\SOFTWARE\Microsoft\Exchange\Client\Options
[3] PS HKLM:\SOFTWARE\Microsoft\Exchange\Client\Options>New-ItemProperty . -
Name DumpsterAlwaysOn -Value 1 -type DWORD


PSPath           : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\S
                   OFTWARE\Microsoft\Exchange\Client\Options
PSParentPath     : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\S
                   OFTWARE\Microsoft\Exchange\Client
PSChildName      : Options
PSDrive          : HKLM
PSProvider       : Microsoft.PowerShell.Core\Registry
DumpsterAlwaysOn : 1
```

A [2]-es sorban az aktuális helynek beállítom az a registry „mappát”, ahol új értéket akarok felvenni, majd a `new-itemproperty` cmdlettel hozom létre az új értéket. Ennek paraméterei a `path` (nincs kiírva, értéke egy darab pont, azaz az aktuális elérési út), a kulcs neve és típusa. Típusként az alábbi táblázat lehetőségeit használhatjuk fel a registryben:

Property típus	Leírás
Binary	bináris adat
DWord	UInt32 egész
ExpandString	Környezeti változókat kifejtő szöveg
MultiString	Többsoros szöveg
String	Szöveg
QWord	8 bájtos bináris adat

Meglevő kulcsok módosítására a `set-itemproperty` cmdlet áll a rendelkezésünkre:

```
[9] PS HKLM:\SOFTWARE\Microsoft\Exchange\Client\Options>set-ItemProperty . -
Name DumpsterAlwaysOn -Value 0
```

Ha esetleg új kulcsot kellene létrehoznunk, arra a `new-item` cmdletet használhatjuk:

```
[14] PS HKLM:\SOFTWARE>New-Item SoosTibor


   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE

SKC  VC Name                           Property
---  -- ----                           --------
  0   0 SoosTibor                      {}
```

Ezután ehhez a `new-itemproperty` cmdlettel lehet felvenni értékeket.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.7.3 Registry elemek hozzáférési listájának kiolvasása

A registry kulcsok is rendelkeznek hozzáférési listával, itt is tetten érhető a fájlokkal való analógia, azaz a registry esetében is a `get-acl` cmdlettel lehet a hozzáférési listát kiolvasni, illetve a `set-acl` cmdlettel módosítani. Miután ez tényleg a fájlokkal teljesen azonos módon történik, így csak a kiolvasásra mutatok egy példát:

```
[84] PS HKCU:\Software> get-acl -path hklm:\system\currentcontrolset\ | fl


Path   : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\system\curr
         entcontrolset
Owner  : BUILTIN\Administrators
Group  : NT AUTHORITY\SYSTEM
Access : BUILTIN\Users Allow  ReadKey
         BUILTIN\Users Allow  -2147483648
         BUILTIN\Power Users Allow  ReadKey
         BUILTIN\Power Users Allow  -2147483648
         BUILTIN\Administrators Allow  FullControl
         BUILTIN\Administrators Allow  268435456
         NT AUTHORITY\SYSTEM Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  268435456
         CREATOR OWNER Allow  268435456
Audit  :
Sddl   : O:BAG:SYD:AI(A;ID;KR;;;BU)(A;CIIOID;GR;;;BU)(A;ID;KR;;;PU)(A;CIIOI
         D;GR;;;PU)(A;ID;KA;;;BA)(A;CIIOID;GA;;;BA)(A;ID;KA;;;SY)(A;CIIOID;
         GA;;;SY)(A;CIIOID;GA;;;CO)
```

Megjegyzés:

Itt jöhetünk rá, hogy vajon miért nem az értékek a „fájlszintű” objektumok a registry PSDrive-ban. Azért, mert ACL-t állítani csak a kulcsokon lehet, a tulajdonságokon (property) nem. Így a fájlrendszerrel való analógia a „kulcs ≈ mappa, fájl” megfeleltetéssel a legoptimálisabb.

A könyv tanfolyami felhasználása

2.8 Tranzakciókezelés

Tranzakciónak hívjuk a logikailag összetartozó olyan műveletsort, amely vagy együttesen végrehajtásra kerül, vagy ha valami probléma adódik a műveletsor közben, akkor vissza lehet állni a műveletsor megkezdése előtti állapotba. Ráadásul a tranzakció közben a „félkész” műveletsor nem látható a rendszer többi eleme számára, így nem okoz számukra nem teljesen kitöltött adatsor problémát. Az SQL adatbázis-kezelőknél ez már régóta ismert lehetőség, de egyéb adattárakon is hasznos segítőnk lehet.

Emlékeztetőül vessünk egy pillantást a PSProviderek listájára:

```
[1] PS C:\> Get-PSProvider

Name                 Capabilities                 Drives
----                 ------------                 ------
WSMan                Credentials                  {WSMan}
Alias                ShouldProcess                {Alias}
Environment          ShouldProcess                {Env}
FileSystem           Filter, ShouldProcess        {C, A, D}
Function             ShouldProcess                {Function}
Registry             ShouldProcess, Transactions  {HKLM, HKCU}
Variable             ShouldProcess                {Variable}
Certificate          ShouldProcess                {cert}
```

A Registry provider az egyetlen jelenleg, ami támogatja a tranzakciókat, ráadásul csak Windows Vista / Windows Server 2008 és újabb operációs rendszereken. A gyakorlatban tehát ezzel a lehetőséggel több registry módosítást összefoghatunk egy tranzakcióba, és vagy az egészet egyben végrehajtatjuk, vagy ha valami gond adódik, akkor vissza tudjuk görgetni az egészet olyan állapotba, amikor még semmilyen változtatást nem tettünk a registryben. Ezzel fáradságos takarítási munkától óv meg minket a rendszer.

Egy időben a registryben több tranzakciót is indíthatunk, azonban mindig csak a legutóbb nyitott tranzakciókban lehet műveleteket végrehajtani, ez az un. aktív tranzakció. Korábban megnyitott tranzakcióban dolgozni csak akkor tudunk, ha a legújabbat vagy visszagörgetjük a kiinduló állapotba, vagy befejezzük.

Várhatólag későbbi PowerShell verziókban majd további providerekre is ki lesz terjesztve ez a tranzakcionálási lehetőség. Nézzük, hogy milyen cmdletek állnak rendelkezésünkre most ebben témában:

```
[2] PS C:\> Get-Command -noun transaction

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Complete-Transaction               Complete-Transaction [-Verb...
Cmdlet          Get-Transaction                    Get-Transaction [-Verbose] ...
Cmdlet          Start-Transaction                  Start-Transaction [-Timeout...
Cmdlet          Undo-Transaction                   Undo-Transaction [-Verbose]...
Cmdlet          Use-Transaction                    Use-Transaction [-Transacte...
```

Tranzakcióban csak erre alkalmas cmdletekkel lehet műveleteket végezni, ezek olyan cmdletek, melyeknek van `-UseTransaction` kapcsoló paramétere:

```
[7] PS C:\> get-help * -Parameter UseTransaction | Get-Command | Sort-Object no
un

CommandType     Name                               Definition
-----------     ----                               ----------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Cmdlet          Get-Acl                         Get-Acl [[-Path] <String[]>...
Cmdlet          Set-Acl                         Set-Acl [-Path] <String[]> ...
Cmdlet          Get-ChildItem                   Get-ChildItem [[-Path] <Str...
Cmdlet          Set-Content                     Set-Content [-Path] <String...
Cmdlet          Get-Content                     Get-Content [-Path] <String...
Cmdlet          Add-Content                     Add-Content [-Path] <String...
Cmdlet          Clear-Content                   Clear-Content [-Path] <Stri...
Cmdlet          Remove-Item                     Remove-Item [-Path] <String...
Cmdlet          New-Item                        New-Item [-Path] <String[]>...
Cmdlet          Get-Item                        Get-Item [-Path] <String[]>...
Cmdlet          Set-Item                        Set-Item [-Path] <String[]>...
Cmdlet          Clear-Item                      Clear-Item [-Path] <String[...
Cmdlet          Invoke-Item                     Invoke-Item [-Path] <String...
Cmdlet          Rename-Item                     Rename-Item [-Path] <String...
Cmdlet          Copy-Item                       Copy-Item [-Path] <String[]...
Cmdlet          Move-Item                       Move-Item [-Path] <String[]...
Cmdlet          Move-ItemProperty               Move-ItemProperty [-Path] <...
Cmdlet          Remove-ItemProperty             Remove-ItemProperty [-Path]...
Cmdlet          Get-ItemProperty                Get-ItemProperty [-Path] <S...
Cmdlet          Rename-ItemProperty             Rename-ItemProperty [-Path]...
Cmdlet          New-ItemProperty                New-ItemProperty [-Path] <S...
Cmdlet          Copy-ItemProperty               Copy-ItemProperty [-Path] <...
Cmdlet          Set-ItemProperty                Set-ItemProperty [-Path] <S...
Cmdlet          Clear-ItemProperty              Clear-ItemProperty [-Path] ...
Cmdlet          Set-Location                    Set-Location [[-Path] <Stri...
Cmdlet          Pop-Location                    Pop-Location [-PassThru] [-...
Cmdlet          Get-Location                    Get-Location [-PSProvider <...
Cmdlet          Push-Location                   Push-Location [[-Path] <Str...
Cmdlet          Resolve-Path                    Resolve-Path [-Path] <Strin...
Cmdlet          Convert-Path                    Convert-Path [-Path] <Strin...
Cmdlet          Join-Path                       Join-Path [-Path] <String[]...
Cmdlet          Test-Path                       Test-Path [-Path] <String[]...
Cmdlet          Split-Path                      Split-Path [-Path] <String[...
Cmdlet          Get-PSDrive                     Get-PSDrive [[-Name] <Strin...
Cmdlet          Remove-PSDrive                  Remove-PSDrive [-Name] <Str...
Cmdlet          New-PSDrive                     New-PSDrive [-Name] <String...
Cmdlet          Use-Transaction                 Use-Transaction [-Transacte...
```

Látható, hogy ez nem is olyan kicsi lista!

Kezdjünk is el mindjárt egy tranzakciót a `Start-Transaction` cmdlettel:

```
[8] PS C:\> Start-Transaction

Suggestion [1,Transactions]: Once a transaction is started, only commands that
get called with the -UseTransaction flag become part of that transaction.
[9] PS C:\> Get-Transaction

        RollbackPreference          SubscriberCount                 Status
        ------------------          ---------------                 ------
                     Error                        1                 Active
```

A [8]-as sor végrehajtása után kapjuk is a figyelmeztetést, hogy a tranzakcióban csak azok a műveletek fognak részt venni, amelyeknél használjuk a kapcsolót. Visszaolvasva a tranzakciókat a `Get-Transaction` cmdlettel látható, hogy ez az aktív tranzakció, és egy helyről indítottam el (egy Subscriber, azaz előfizetője van) és automatikusan visszagördítődik a tranzakció, ha valamilyen hiba lép fel. Ezt a `RollbackPreference` tulajdonság írja le, amelyet át is lehet írni, ha a `Start-Transaction`-t a

megfelelően megadott `-RollBackPreference` paraméterrel hívjuk meg. A lehetséges értékek: `Error`, `TerminatingError` és `Never`.

Ugyancsak szabályozni lehet a tranzakciók automatikus visszaállását a `Start-Transaction -TimeOut` paraméterével, ami alaphelyzetben az interaktívan indított tranzakcióknál nincsen, viszont a szkriptekből indított tranzakciók esetében 30 percig engedi a lezáratlan tranzakciókat létezni, utána automatikusan visszaállít.

Használjuk az előbb indított tranzakciót! Létrehozok egy elemet:

```
[12] PS C:\> New-Item 'HKCU:\Software\soostibi' -UseTransaction


    Hive: HKEY_CURRENT_USER\Software


SKC  VC Name                           Property
---  -- ----                           --------
  0   0 soostibi                       {}


[13] PS C:\> Get-Item 'HKCU:\Software\soostibi'
Get-Item : Cannot find path 'HKCU:\Software\soostibi' because it does not exis
t.
At line:1 char:9
+ Get-Item <<<<  'HKCU:\Software\soostibi'
    + CategoryInfo          : ObjectNotFound: (HKCU:\Software\soostibi:String
   ) [Get-Item], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetI
   temCommand

[14] PS C:\> Get-Item 'HKCU:\Software\soostibi' -UseTransaction


    Hive: HKEY_CURRENT_USER\Software


SKC  VC Name                           Property
---  -- ----                           --------
  0   0 soostibi                       {}
```

Tehát létrehoztam egy „soostibi” nevű elemet a registry HKCU/Software ágában. Mikor visszaolvastam ezt a `Get-Item` cmdlettel ([13]-as sor), akkor hibát kaptam, hiszen itt nem használtam a `-UseTransaction` paramétert és a tranzakción belüli műveletek a külvilág számára nem látszanak. Amikor már használtam ezt a kapcsolót a [14]-es sorban, akkor természetesen már látható volt a létrehozott elem.

Ha minden egyéb körülménnyel meg vagyok elégedve, akkor lezárhatom a tranzakciót a `Complete-Transaction` cmdlettel, és ha ekkor olvasom vissza a registry kulcsot „normál” módon, akkor már látható az imént felvett elem:

```
[15] PS C:\> Complete-Transaction
[16] PS C:\> Get-Item 'HKCU:\Software\soostibi'


    Hive: HKEY_CURRENT_USER\Software


SKC  VC Name                           Property
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
---  -- ----                              --------
  0   0 soostibi                         {}
```

Természetesen az `Undo-Transaction` segítségével semmissé lehetett volna tenni a tranzakciót. Ugyanígy semmissé válik egy tranzakció a már korábban említett hibajelenségekkor, vagy ha bizonyos ideig nem zárjuk le.

Ha a tranzakciót lezártuk, vagy semmissé tettük, akkor is lekérdezhető az állapota:

```
[15] PS C:\> Get-Transaction

        RollbackPreference           SubscriberCount                   Status
        ------------------           ---------------                   ------
                     Error                         0                Committed
```

A `Status` tulajdonság mutatja, hogy milyen módon fejeződött be.

Ha többször is kiadjuk a `Start-Transaction` cmdletet, akkor alaphelyzetben nem nyílik újabb és újabb tranzakciós környezet, hanem csak az előfizetők száma növekszik. Azaz a tranzakciót használó szkriptek, még ha egymástól függetlenül is indultak el, mégis egy közös tranzakciós terepen dolgoznak. Bármelyikük miatt is történik egy esetleges visszaállás az eredeti helyzetbe, ezt mindegyik előfizető „megérzi”. Ezzel szemben a tranzakciót annyiszor kell lezárni, ahány előfizető van, így a teljes tranzakciós környezet csak akkor zárul sikeresen, ha minden folyamat ehhez hozzájárult.

Ha egymástól független tranzakciós terepeket akarunk nyitni, akkor a `Start-Transaction` cmdletet az `-Independent` kapcsolóval hívjuk meg.

A könyv tanfolyami felhasználása nem engedélyezett!

2.9 Számítógépek és a hálózati kapcsolatok cmdletei

A rendszergazdák egyik gyakori tevékenysége a számítógépekkel, számítógép-fiókokkal kapcsolatos. Erre a PowerShell 2.0-ban már rendelkezésre állnak cmdletek.

A számítógép tartományba vagy munkacsoportba való léptetéséhez az `Add-Computer` cmdletet használhatjuk. Munkacsoportba léptetése esetén:

```
PS C:\Users\Administrator> Add-Computer -WorkGroupName WG
WARNING: The changes will take effect after you restart the computer MEMBER.
```

Maga a parancs nem indítja újra a gépet, de figyelmeztet, hogy a változások csak újraindítás után jutnak érvényre. Újraindítani a `Restart-Computer` cmdlettel lehet:

```
PS C:\Users\Administrator> Restart-Computer
```

Tartományba léptetéshez ugyancsak az `Add-Computer` használatos, de természetesen más paraméterezéssel:

```
PS C:\> Add-Computer -DomainName "r2.dom" -Credential r2\administrator -OUPath
"ou=managed users,dc=r2,dc=dom"
WARNING: The changes will take effect after you restart the computer MEMBER.
```

Látható, hogy megadható a szervezeti egység, ahol a tartományi számítógépfiók létrejön. Menet közben még a megadott rendszergazda fiók jelszavát is bekéri a parancs. Ha nem akarjuk a grafikus felületen interaktívan megadni a jelszót, akkor a hitelesítési adatokat külön objektumként létre kell hozni. Ehhez nézzük a tartományból kiléptetés példáját a `remove-computer` cmdlet példáján:

```
PS C:\Users\Administrator> $cred = new-object -typename System.Management.Autom
ation.PSCredential -argumentlist "r2\administrator", ( Convertto-SecureString "
Pa$$w0rd" -force -asplaintext)
PS C:\Users\Administrator> Remove-Computer -Credential $cred

Confirm
After you leave the domain, you will need to know the password of the local
Administrator account to log onto this computer. Do you want to remove this
computer from the domain?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
WARNING: The changes will take effect after you restart the computer MEMBER.
```

A fenti példában látszik, hogy egy `$cred` változóba létrehozok egy `System.Management.Automation.PSCredential` objektumot, a konstruktor paramétereként megadom a felhasználó nevét és a jelszavát `SecureString` formátumban, amit a `ConvertTo-SecureString` cmdlet végez el.

Számítógépet leállítani a `Stop-Computer` cmdlettel lehet:

```
[89] PS C:\> Stop-Computer -ComputerName member
```

Itt most ennek a parancsnak azt a változatát mutattam, amellyel távoli futtatást alkalmaztam. A fenti parancsok ilyen paraméterezéssel futtathatók távoli módon és akár háttérfolyamatként is.

Nyomtatásban nem engedélyezett!

Bár a főnevében nem számítógép, hanem kapcsolat, de azért mégis a számítógépekhez köthető leggyakrabban a `Test-Connection` cmdlet, mely az ősi „ping” segédprogram PowerShelles megfelelője. Nézzük, mit tud:

```
[36] PS C:\> Test-Connection member

WARNING: 2 columns do not fit into the display and were removed.

Source        Destination     IPV4Address      IPV6Address
------        -----------     -----------      -----------
DC            member          192.168.1.11     {}
DC            member          192.168.1.11     {}
DC            member          192.168.1.11     {}
DC            member          192.168.1.11     {}
```

(Sajnos a könyv méretadottságai miatt 2 oszlop lemaradt: az átküldött bájtok száma és a válaszidő.) Eddig ez egy ping eredményére hasonlít. Miben tud ez többet? Például abban, hogy ha csak a kapcsolat meglétét akarjuk ellenőrizni, nincs szükségünk részletes statisztikára, akkor kérhetünk egy nagyon tömör visszatérési értéket is:

```
[38] PS C:\> Test-Connection member -Quiet
True
```

Ilyenkor csak egy True/False választ kapunk. De még ennél is többet tud ez! Egyrészt több gépet is meg lehet „pingelni” egyszerre:

```
[39] PS C:\> Test-Connection -ComputerName member, dc

WARNING: 2 columns do not fit into the display and were removed.

Source        Destination     IPV4Address      IPV6Address
------        -----------     -----------      -----------
DC            dc              192.168.1.10     fe80::f9d9:a919:495d:a8fb%11
DC            member          192.168.1.11     {}
DC            dc              192.168.1.10     fe80::f9d9:a919:495d:a8fb%11
DC            member          192.168.1.11     {}
DC            dc              192.168.1.10     fe80::f9d9:a919:495d:a8fb%11
DC            member          192.168.1.11     {}
DC            dc              192.168.1.10     fe80::f9d9:a919:495d:a8fb%11
DC            member          192.168.1.11     {}
```

Másrészt nem feltétlenül kell, hogy az a gép legyen a forrás, ahol épp ülünk, távoli gépről még távolabbi gépre is kezdeményezhetjük a pinget:

```
[48] PS C:\> Test-Connection -Source Member -ComputerName "192.168.1.200"

WARNING: 2 columns do not fit into the display and were removed.

Source        Destination     IPV4Address      IPV6Address
------        -----------     -----------      -----------
MEMBER        192.168.1.200   192.168.1.200    {}
MEMBER        192.168.1.200   192.168.1.200    {}
MEMBER        192.168.1.200   192.168.1.200    {}
MEMBER        192.168.1.200   192.168.1.200    {}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A fenti példában én a DC nevű gépen dolgozom, de a Member nevű gép kapcsolatát szeretném ellenőrizni a 192.168.1.200-as címen található eszközzel. Ebben az esetben a mélyen az én gépem felvette a kapcsolatot a Member gép WMI interfészével, és azt kérte meg a ping kiadására. Ilyen esetben szükséges lehet autentikáltatni magunkat ezzel a géppel, ha az éppen belépett felhasználó nem jogosult a WMI felület elérésére. Erre a -Credential paraméter megadása biztosít lehetőséget.

Még „szorosabb" kapcsolatot tesztel a Test-ComputerSecureChannel cmdlet. Ezzel a tartományi gépek közti biztonságos, titkosított adatcsatorna meglétét és működőképességét lehet tesztelni:

```
[49] PS C:\> Test-ComputerSecureChannel -Server member
True
```

Ha esetleg ez a kapcsolat sérült lenne, akkor ezen cmdlet -repair kapcsolójával lehet megjavítani azt.

Még talán ebbe a fejezetbe sorolható a gépre telepített javítások listázását végző Get-HotFix cmdlet:

```
[7] PS C:\> Get-HotFix | Sort-Object installedon -Descending

Source        Description      HotFixID      InstalledBy          InstalledOn
------        -----------      --------      -----------          -----------
TIBI-PC       Update           KB978637      NT AUTHORITY\SYSTEM  2010.02.2...
TIBI-PC       Update           KB977863      NT AUTHORITY\SYSTEM  2010.02.2...
TIBI-PC       Update           KB976662      NT AUTHORITY\SYSTEM  2010.02.2...
TIBI-PC       Update           KB979306      NT AUTHORITY\SYSTEM  2010.02.2...
TIBI-PC       Update           KB976264      NT AUTHORITY\SYSTEM  2010.02.2...
TIBI-PC       Security Update  KB978251      NT AUTHORITY\SYSTEM  2010.02.1...
…
```

Ezt távoli gépre is tudjuk futtatni.

Szitnén ide tartozik a Reset-ComputerMachinePassword cmdlet, mellyel a számítógép jelszavának módosítását lehet kezdeményezni. Ez a parancs csak helyi gépre működik.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.10 Helyreállítási pontok kezelése

A legújabb Windows operációs rendszerekben megtalálható a Volume Shadow Copy szolgáltatás, mellyel a számítógép meghajtóiról (volumes) készíthető egy „pillanatfelvétel”, mellyel tulajdonképpen az adott meghajtónak két nézete lesz: egy „normál” nézet, melyben minden fájl a megszokott módon működik, és egy olyan „rejtett” nézet, melyben a fájlok csak olvashatók és a pillanatfelvétel időpontjának állapotát tükrözik. Azaz ebben a nézetben az amúgy zárolt és még csak nem is olvasható fájlok is olvashatók.

Ezt a fajta „lefagyasztását” a meghajtóknak hívjuk helyreállítási pontoknak (restore point), hiszen innen vissza tudunk nyerni régebben létező fájlokat, vagy fájlok régebbi verzióját. Ezt a technológiát használja a beépített mentő szoftver (Windows Backup és Windows Server Backup) úgy, hogy mentés első lépéseként készít egy ilyen helyreállítási pontot, majd a háttérben ezt írja ki írható médiára vagy hálózati meghajtóra. Ezen kívül a rendszer a frissítés során is készít ilyen helyreállítási pontokat, hogy ha egy frissítés mégsem sikerül, akkor lehetőség legyen visszatérni egy régebbi állapotba.

Ezek a helyreállítási pontok nem igényelnek nagyon nagy helyet a meghajtókon, hiszen a lefagyasztott területhez képest csak a változások foglalnak el új helyet. Így még arra is van lehetőség, hogy ilyen „lefagyasztott terület” több is legyen, azaz egyidejűleg több helyreállítási pontot is fenntarthatunk. Persze idővel az így fenntartott helyek a diszken már túl sok helyet foglalhatnak el, ilyenkor a rendszer felszabadítja a régebbi helyreállítási pontok által foglalt területeket.

A PowerShell 2.0-ban lehetőség van ezen helyreállítási pontok kezelésére. Sajnos ezeket csak a kliens operációsrendszereken kezelhetjük, a kiszolgáló változatokon nem.

Elsőként nézzük meg, hogy milyen helyreállítási pontjaink vannak a `Get-ComputerRestorePoint` cmdlet segítségével:

```
PS C:\> Get-ComputerRestorePoint

WARNING: column "RestorePointType" does not fit into the display and was remov
ed.

CreationTime            Description                   SequenceNumber    EventT
                                                                          ype
------------            -----------                   --------------    ------
2010.01.27. 21:40:25    Windows Update                314               BEG...
2010.01.28. 1:53:56     Windows Update                315               BEG...
2010.01.28. 9:54:02     Windows Update                316               BEG...
2010.01.28. 21:55:21    Windows Update                317               BEG...
2010.01.29. 9:54:34     Windows Update                318               BEG...
2010.01.29. 21:54:54    Windows Update                319               BEG...
2010.01.30. 9:59:40     Windows Update                320               BEG...
2010.01.31. 9:48:18     Windows Update                321               BEG...
```

Látható, hogy ezek a helyreállítási pontok a Windows Update szolgáltatás által jöttek létre. Én magam is készíthetek ilyen helyreállítási pontot a `Checkpoint-Computer` cmdlet segítségével:

```
PS C:\> Checkpoint-Computer -Description "Manuális pont"
PS C:\> Get-ComputerRestorePoint

WARNING: column "RestorePointType" does not fit into the display and was remov
ed.

CreationTime            Description                   SequenceNumber    EventT
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                                                                    ype
------------           -----------                  --------------  ------
2010.01.27. 21:40:25   Windows Update               314             BEG...
2010.01.28. 1:53:56    Windows Update               315             BEG...
2010.01.28. 9:54:02    Windows Update               316             BEG...
2010.01.28. 21:55:21   Windows Update               317             BEG...
2010.01.29. 9:54:34    Windows Update               318             BEG...
2010.01.29. 21:54:54   Windows Update               319             BEG...
2010.01.30. 9:59:40    Windows Update               320             BEG...
2010.01.31. 9:48:18    Windows Update               321             BEG...
2010.01.31. 13:02:28   Manuális pont                322             BEG...
```

A fenti példában létrehoztam egy helyreállítási pontot „Manuális pont” néven. Ez kb. egy percet vett igénybe, amíg futott egy *progress bar* mutatta, hogy mennyit kell még várnom. A futás után visszaolvasva az utolsó helyen látható az általam létrehozott helyreállítási pont.

Korábbi helyreállítási pontokra a `Restore-Computer` paranccsal lehet visszaállni. Ez a gépet újraindítja és a tényleges visszaállást az boot folyamat során végzi el.

A helyreállásnak lehetnek káros következményei is. Gondoljunk arra, hogy visszaállunk egy korábbi állapotra, mert például eszközmeghajtó szoftver telepítése nem sikerült, de ezzel az időközben készített dokumentumaink is odavesznek. Ennek elkerülésére ki lehet kapcsolni bizonyos meghajtókra a helyreállítást a `Disable-ComputerRestore` cmdlet segítségével. Ezzel a lehetőséggel természetesen csak akkor élhetünk, ha több logikai meghajtónk (partíciónk, kötetünk) van, így érdemes már eleve így kialakítani a számítógépeinket. Visszakapcsolni ezt az `Enable-ComputerRestore` cmdlettel lehet.

A könyv tanfolyami felhasználása nem engedélyezett!

2.11 WMI, processzek, rendszerszolgáltatások

A WMI technológia biztosítja a hálózatfelügyeleti szoftverek (így például a Microsoft Systems Management Server) számára szükséges infrastruktúrát. Kifejlesztésének célja elsősorban az volt, hogy egységes keretet (és felületet) biztosítson a már létező felügyeleti technológiáknak (SNMP, DMI, stb.).

A WMI első verziója a Windows NT SP4 CD-n jelent meg (Option Pack), de az SMS 2.0 verziója már teljes egészében erre épült. Jelentős különbségek vannak azonban a Windows 2000-ben, az XP-ben és a Windows Server 2003-ban található változatok között. A legfontosabb különbség az, hogy a későbbi WMI változatok egyre több írható tulajdonságot tartalmaznak, vagyis lehetőséget adnak nem csak a rendszerjellemzők lekérdezésére, hanem beállítására is.

A WMI tartozéka a WMI Tools nevű csomag, de ez nem része a Windows telepítésnek, külön kell letölteni és telepíteni a Microsoft „downloads” weboldaláról. A csomag tartalmazza többek között a CIM Studio és az Object Browser nevű alkalmazásokat; ezekről a függelékben található, részletesebb, a WMI elméletével foglalkozó fejezetben írok majd.

2.11.1 WMI objektumok elérése PowerShell-ből

Nézzük meg a WMI-vel kapcsolatos PowerShell cmdleteket:

```
[1] PS C:\> Get-Command -Noun wmi*

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Get-WmiObject                       Get-WmiObject [-Class] <Str...
Cmdlet          Invoke-WmiMethod                    Invoke-WmiMethod [-Class] <...
Cmdlet          Register-WmiEvent                   Register-WmiEvent [-Class] ...
Cmdlet          Remove-WmiObject                    Remove-WmiObject [-Class] <...
Cmdlet          Set-WmiInstance                     Set-WmiInstance [-Class] <S...
```

A leggyakrabban használatos ezek közül a `Get-WMIObject`. Nézzük meg a lehetséges szintaxisokat:

```
[2] PS C:\> (Get-Help Get-WmiObject).syntax

Get-WmiObject [-Authority <string>] [-Amended] [-AsJob] [-Authentication {Defa
ult | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Uncha
nged}] [-ComputerName <string[]>] [-Credential <PSCredential>] [-EnableAllPriv
ileges] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delega
te}] [-Locale <string>] [-Namespace <string>] [-ThrottleLimit <int>] [<CommonP
arameters>]
Get-WmiObject [[-Class] <string>] [-Authority <string>] [-List] [-Recurse] [-A
mended] [-AsJob] [-Authentication {Default | None | Connect | Call | Packet |
PacketIntegrity | PacketPrivacy | Unchanged}] [-ComputerName <string[]>] [-Cre
dential <PSCredential>] [-EnableAllPrivileges] [-Impersonation {Default | Anon
ymous | Identify | Impersonate | Delegate}] [-Locale <string>] [-Namespace <st
ring>] [-ThrottleLimit <int>] [<CommonParameters>]
Get-WmiObject [-Authority <string>] [-Amended] [-AsJob] [-Authentication {Defa
ult | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Uncha
nged}] [-ComputerName <string[]>] [-Credential <PSCredential>] [-EnableAllPriv
ileges] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delega
te}] [-Locale <string>] [-Namespace <string>] [-ThrottleLimit <int>] [<CommonP
arameters>]
Get-WmiObject [-Class] <string> [[-Property] <string[]>] [-Authority <string>]
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
 [-DirectRead] [-Filter <string>] [-Amended] [-AsJob] [-Authentication {Defaul
t | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchang
ed}] [-ComputerName <string[]>] [-Credential <PSCredential>] [-EnableAllPrivil
eges] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate
}] [-Locale <string>] [-Namespace <string>] [-ThrottleLimit <int>] [<CommonPar
ameters>]
Get-WmiObject -Query <string> [-Authority <string>] [-DirectRead] [-Amended] [
-AsJob] [-Authentication {Default | None | Connect | Call | Packet | PacketInt
egrity | PacketPrivacy | Unchanged}] [-ComputerName <string[]>] [-Credential <
PSCredential>] [-EnableAllPrivileges] [-Impersonation {Default | Anonymous | I
dentify | Impersonate | Delegate}] [-Locale <string>] [-Namespace <string>] [-
ThrottleLimit <int>] [<CommonParameters>]
```

Látszik, hogy öt különböző szintaxisa is van, bár ebből az első és a harmadik szemre ugyanaz. Az elsőként nézzük meg, hogy milyen WMI osztályok vannak. Ehhez fel kell deríteni a WMI névtereit is, amit a hierarchikus WMI adatbázis `ROOT` elemének segítségével tehetjük meg:

```
[11] PS C:\> get-wmiobject -namespace root -list


   NameSpace: ROOT

Name                                  Methods              Properties
----                                  -------              ----------
__SystemClass                         {}                   {}
__thisNAMESPACE                       {}                   {SECURITY_DESCRIPTOR}
__CacheControl                        {}                   {}
__EventConsumerProviderCacheControl   {}                   {ClearAfter}
__EventProviderCacheControl           {}                   {ClearAfter}
__EventSinkCacheControl               {}                   {ClearAfter}
__ObjectProviderCacheControl          {}                   {ClearAfter}
__PropertyProviderCacheControl        {}                   {ClearAfter}
__NAMESPACE                           {}                   {Name}
__ArbitratorConfiguration             {}                   {OutstandingTasksP...
…
```

Ebben a listában látható a `__NAMESPACE` elem, ezt kell tüzetesebben lekérdeznünk:

```
[13] PS C:\> Get-WmiObject -Namespace root -Class __NAMESPACE | Format-Table na
me

name
----
subscription
DEFAULT
MicrosoftDfs
CIMV2
Cli
nap
MicrosoftActiveDirectory
SECURITY
RSOP
MicrosoftDNS
WMI
directory
Policy
Interop
Hardware
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
ServiceModel
Microsoft
aspnet
```

Az alábbi kifejezéssel összes névteret fel tudjuk deríteni:

```
[45] PS C:\> Get-WmiObject -Namespace root -list -recurse -class __namespace| f
ormat-table @{l="név";e={$_.__Namespace}}

név
---
ROOT
ROOT\subscription
ROOT\DEFAULT
ROOT\MicrosoftDfs
ROOT\CIMV2
ROOT\Cli
ROOT\nap
ROOT\MicrosoftActiveDirectory
ROOT\SECURITY
ROOT\rsop
ROOT\MicrosoftDNS
ROOT\WMI
ROOT\directory
ROOT\Policy
ROOT\Interop
ROOT\Hardware
ROOT\ServiceModel
ROOT\Microsoft
ROOT\aspnet
ROOT\CIMV2\Security
ROOT\CIMV2\power
ROOT\CIMV2\TerminalServices
ROOT\RSOP\User
ROOT\RSOP\Computer
ROOT\directory\LDAP
ROOT\Microsoft\HomeNet
ROOT\CIMV2\Security\MicrosoftTpm
ROOT\RSOP\User\S_1_5_21_1609476370_3801812663_245660911_500
ROOT\RSOP\User\S_1_5_21_3398938913_3940250523_927435294_500
```

Ezután már egy konkrét névtér felderítése a következő kifejezéssel végezhető el:

```
[19] PS C:\> get-wmiobject -namespace "root\CIMV2" -list


   NameSpace: ROOT\CIMV2

Name                                  Methods              Properties
----                                  -------              ----------
__SystemClass                         {}                   {}
__thisNAMESPACE                       {}                   {SECURITY_DESCRIPTOR}
__NAMESPACE                           {}                   {Name}
__Provider                            {}                   {Name}
__Win32Provider                       {}                   {ClientLoadableCLS...
__ProviderRegistration                {}                   {provider}
__EventProviderRegistration           {}                   {EventQueryList, p...
__ObjectProviderRegistration          {}                   {InteractionType, ...
__ClassProviderRegistration           {}                   {CacheRefreshInter...
```

```
__InstanceProviderRegistration      {}                  {InteractionType, ...
__MethodProviderRegistration        {}                  {provider}
__PropertyProviderRegistration      {}                  {provider, Support...
__EventConsumerProviderRegistration {}                  {ConsumerClassName...
__IndicationRelated                 {}                  {}
__EventFilter                       {}                  {CreatorSID, Event...
__EventConsumer                     {}                  {CreatorSID, Machi...
__FilterToConsumerBinding           {}                  {Consumer, Creator...
__AggregateEvent                    {}                  {NumberOfEvents, R...
__TimerNextFiring                   {}                  {NextEvent64BitTim...
  Event                             {}                  {SECURITY DESCRIPT...
__ExtrinsicEvent                    {}                  {SECURITY_DESCRIPT...
Win32_DeviceChangeEvent             {}                  {EventType, SECURI...
Win32_SystemConfigurationChangeE... {}                  {EventType, SECURI...
…
```

Itt fontos a `-list` kapcsoló használata, hiszen enélkül kérné a `-class` paramétert, amellyel egy konkrét WMI osztály elemeit térképezhetjük fel.

Ha már tudjuk, hogy milyen osztály objektumait keressük, akkor használhatjuk az első szintaxist. Például keressük az adott gép hálózati adaptereit:

```
[81] PS C:\> Get-WmiObject -Namespace root\cimv2 -Class win32_networkadapter |
Format-Table

ServiceName MACAddress  AdapterTyp DeviceID   Name       NetworkAdd Speed
                        e                               resses
----------- ----------  ---------- --------   ----       ---------- -----
RasSstp                            0          WAN Min...
RasAgileVpn                        1          WAN Min...
Rasl2tp                            2          WAN Min...
PptpMini...                        3          WAN Min...
RasPppoe                           4          WAN Min...
NdisWan                            5          WAN Min...
NdisWan                            6          WAN Min...
E1G60       08:00:27... Etherne... 7          Intel(R...             1000000000
tunnel                  Tunnel     8          Microso...             100000
NdisWan                            9          WAN Min...
tunnel                  Tunnel     10         Teredo ...             100000
AsyncMac    20:41:53... Wide Ar... 11         RAS Asy...
```

A fenti listában elég sok hálózati kártya látszik, de ezek közül „igazi” csak egy van. Ezt a következő szűrőfeltétel megadásával lehet kiszűrni:

```
[86] PS C:\> Get-WmiObject -Namespace root\cimv2 -Class win32_networkadapter -F
ilter "AdapterType = 'Ethernet 802.3'"


ServiceName      : E1G60
MACAddress       : 08:00:27:90:23:A2
AdapterType      : Ethernet 802.3
DeviceID         : 7
Name             : Intel(R) PRO/1000 MT Desktop Adapter
NetworkAddresses :
Speed            : 1000000000
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ennél jóval több információt hordoz egy ilyen objektum, ezt a `Format-List *` segítségével tehetjük láthatóvá:

```
[87] PS C:\> Get-WmiObject -Namespace root\cimv2 -Class win32_networkadapter -F
ilter "AdapterType = 'Ethernet 802.3'" | format-list *


Availability                : 3
Name                        : Intel(R) PRO/1000 MT Desktop Adapter
Status                      :
StatusInfo                  :
DeviceID                    : 7
__GENUS                     : 2
__CLASS                     : Win32_NetworkAdapter
__SUPERCLASS                : CIM_NetworkAdapter
__DYNASTY                   : CIM_ManagedSystemElement
__RELPATH                   : Win32_NetworkAdapter.DeviceID="7"
__PROPERTY_COUNT            : 40
__DERIVATION                : {CIM_NetworkAdapter, CIM_LogicalDevice, CIM_Logi
                              calElement, CIM_ManagedSystemElement}
__SERVER                    : DC
__NAMESPACE                 : root\cimv2
__PATH                      : \\DC\root\cimv2:Win32_NetworkAdapter.DeviceID="7
                              "
AdapterType                 : Ethernet 802.3
AdapterTypeId               : 0
AutoSense                   :
Caption                     : [00000007] Intel(R) PRO/1000 MT Desktop Adapter
ConfigManagerErrorCode      : 0
ConfigManagerUserConfig     : False
CreationClassName           : Win32_NetworkAdapter
Description                 : Intel(R) PRO/1000 MT Desktop Adapter
ErrorCleared                :
ErrorDescription            :
GUID                        : {FF327D73-8AE0-4949-98C2-BAD7DABFC3CB}
Index                       : 7
InstallDate                 :
Installed                   : True
InterfaceIndex              : 11
LastErrorCode               :
MACAddress                  : 08:00:27:90:23:A2
Manufacturer                : Intel
MaxNumberControlled         : 0
MaxSpeed                    :
NetConnectionID             : Local Area Connection
NetConnectionStatus         : 2
NetEnabled                  : True
…
```

Látható, hogy a `Filter` paraméternek egy WQL lekérdezés `Where` feltételét kell megadni, azaz itt nem PowerShell, hanem WQL szintaxis használandó. Ha valaki nagyon profi WQL lekérdezések megfogalmazásában, akkor a fenti szűrést mindenestül is megadhatja WQL formában:

```
[89] PS C:\> Get-WmiObject -Namespace root\cimv2 -Query "select * from win32_ne
tworkadapter where AdapterType = 'Ethernet 802.3'"


ServiceName      : E1G60
MACAddress       : 08:00:27:90:23:A2
```

```
AdapterType      : Ethernet 802.3
DeviceID         : 7
Name             : Intel(R) PRO/1000 MT Desktop Adapter
NetworkAddresses :
Speed            : 1000000000
…
```

Megjegyzés

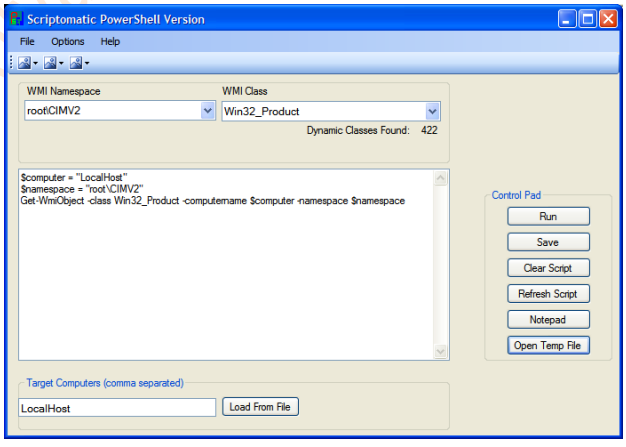
Vigyázni kell az idézőjelezésre, hiszen a WQL kifejezést idézőjelezni kell, amin belül szintén idézőjel szerepel általában a `where` feltételnél. Ezt legegyszerűbben a kétfajta idézőjel használatával oldhatjuk meg, mint ahogy a fenti példában is tettem, és akkor nem kell sem az *escape* (` Alt Gr+7) karaktert használni, sem többszörözni az idézőjeleket.

A WMI-ben az még a nagyszerű, hogy távoli gépekre is kiadhatók lekérdezések. Az alábbi paranccsal két gép, a DC és a MEMBER nevű gép hálózati kártyáit kérdezhetjük le:

```
[5] PS C:\> Get-WmiObject -Namespace root\cimv2 -ComputerName dc,member -Class
Win32_NetworkAdapter -Filter "AdapterType = 'Ethernet 802.3'" | Format-Table sy
stemname, name, macaddress

systemname                 name                       macaddress
----------                 ----                       ----------
DC                         Intel(R) PRO/1000 MT D...  08:00:27:90:23:A2
MEMBER                     Intel(R) PRO/1000 MT D...  08:00:27:8D:0D:AE
```

A `-credential` paraméterrel még azt is megadhatjuk, hogy kinek a nevében akarunk csatlakozni a WMI felület eléréséhez. Természetesen a parancs sikeres futtatásához biztosítani kell a Windows tűzfalon a megfelelő portok megnyitását, hogy a kommunikáció sikeres legyen.

A WMI osztályok böngészésében, és az osztályok példányainak kilistázásához szükséges PowerShell parancs összeállításában segít a PowerShellScriptOMatic ingyenesen letölthető apró kis eszköz:



67. ábra PowerShellScriptOMatic

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.11.2 Folyamatok és rendszerszolgáltatások

Korábban már találkozhattunk a `Get-Process` és a `Get-Service` cmdlettel, amelyek segítségével a rendszerfolyamatok és a szolgáltatások listáját kérhetjük le. A folyamatokkal és szolgáltatásokkal kapcsolatban azonban nem csak listázást, hanem bármilyen más feladatot is elvégezhetünk a PowerShell cmdletjeivel, illetve ha minden kötél szakad, közvetlenül a megfelelő .NET komponensek segítségével.

Először is nézzük meg, hogy milyen cmdleteket használhatunk a folyamatok kezelésére:

```
[75] PS C:\> Get-Command -noun process

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Debug-Process                       Debug-Process [-Name] <Stri...
Cmdlet          Get-Process                         Get-Process [[-Name] <Strin...
Cmdlet          Start-Process                       Start-Process [-FilePath] <...
Cmdlet          Stop-Process                        Stop-Process [-Id] <Int32[]>...
Cmdlet          Wait-Process                        Wait-Process [-Name] <Strin...
```

Négy cmdlet került a listába. A `Debug-Process` a paraméterként megadott processzhez tartozó debuggert nyitja meg, igazából fejlesztőknek szánt cmdlet ez.

A `Start-Process` segítségével lehet újabb folyamatokat indítani. Például nézzük meg a Notepad.exe indítását:

```
[1] PS C:\> Start-Process notepad.exe
```

Ez így nem mutat túl sokat, akár így is elindíthattam volna:

```
PS C:\> notepad
```

Ez a megoldás valóban egyszerű, azonban korántsem egyenértékű az előzővel. Merthogy a `Start-Process`-nek további paramétereket is megadhatunk:

```
[2] PS C:\> Start-Process notepad.exe -ArgumentList C:\fájl.txt -PassThru

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     57       7     1256       5144    72     0,19   3176 notepad
```

A `-PassThru` paraméterrel visszakapjuk az elindított processz adatait. Ez ahhoz kell nekünk, hogy például pontosan ezt a processzt tudjuk a `Stop-Process` segítségével bezárni, mert név alapján a `Stop-Process` minden adott nevű folyamatot bezár:

```
[3] PS C:\> stop-process -name "notepad"
```

Ha tudjuk a processz azonosítóját, akkor a `Wait-Process` segítségével várakozhatunk annak befejeződéséig:

```
[4] PS C:\> Wait-Process -Id 3176
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Sajnos a processzek kezelése csak helyi gépen történhet. Távoli gépeken próbálkozhatunk az `Invoke-Command` segítségével:

```
[5] PS C:\> Invoke-Command -ComputerName member -ScriptBlock {start-process not
epad}
```

Ezzel az a baj, hogy nem az éppen bejelentkezett felhasználó desktop felületén nyílik meg az alkalmazás, hanem háttérben, így nem sok minden látszik belőle. Konkrétan a Notepad le is áll. Egy másik módszer a távoli indításra a WMI-n keresztül:

```
[5] PS C:\> Invoke-WmiMethod -Path Win32_Process -Name create -ArgumentList not
epad.exe -ComputerName member


__GENUS           : 2
__CLASS           : __PARAMETERS
__SUPERCLASS      :
__DYNASTY         : __PARAMETERS
__RELPATH         :
__PROPERTY_COUNT  : 2
__DERIVATION      : {}
__SERVER          :
__NAMESPACE       :
__PATH            :
ProcessId         : 1124
ReturnValue       : 0
```

Ez ugyanúgy háttérben futtatja a processzt, így itt sem érdemes grafikus alkalmazásokkal játszani, csak olyan folyamatokat indítsunk, amelyek nem igényelnek ablakot, például ilyenek általában a parancssori eszközök.

Nézzük meg most azt, hogy a rendszerfolyamatokkal kapcsolatosan milyen cmdletek állnak rendelkezésünkre:

```
[76] PS C:\> Get-Command -noun service

CommandType     Name                                Definition
-----------     ----                                ----------
Cmdlet          Get-Service                         Get-Service [[-Name] <Strin...
Cmdlet          New-Service                         New-Service [-Name] <String...
Cmdlet          Restart-Service                     Restart-Service [-Name] <St...
Cmdlet          Resume-Service                      Resume-Service [-Name] <Str...
Cmdlet          Set-Service                         Set-Service [-Name] <String...
Cmdlet          Start-Service                       Start-Service [-Name] <Stri...
Cmdlet          Stop-Service                        Stop-Service [-Name] <Strin...
Cmdlet          Suspend-Service                     Suspend-Service [-Name] <St...
```

Például nézzük, hogy hogyan tudnánk leállítani a gépünkön az összes leállítható szolgáltatást! Először is kérnünk kell egy listát a szolgáltatásokról, és ki kell válogatnunk közülük azokat, amelyek leállíthatók, vagyis a `CanStop` tulajdonságuk „igaz” értéket ad vissza:

```
[4] PS C:\> get-service | where-object {$_.CanStop}

Status   Name               DisplayName
------   ----               -----------
```

```
Running  ADWS                Active Directory Web Services
Running  Appinfo             Application Information
Running  BFE                 Base Filtering Engine
Running  CertPropSvc         Certificate Propagation
...
```

A kiírt táblázatban nem szerepel a `CanStop` érték, így ha nem vagyunk biztosak a dolgunkban, érdemes lehet egy olyan listát kérni, amelyben saját szemünkkel is meggyőződhetünk a helyes eredményről:

```
[6] PS C:\> get-service | where-object {$_.CanStop} | format-table name, cansto
p -autosize

Name                CanStop
----                -------
ADWS                   True
Appinfo                True
BFE                    True
CertPropSvc            True
...
```

Ezután már csak végig kell lépkednünk a gyűjtemény valamennyi elemén, és mindegyikre meghívni a `Stop-Service` cmdletet:

```
PS C:\> get-service | where-object {$_.CanStop} | Stop-Service
```

Vigyázat! A szolgáltatások leállítása után valószínűleg újra kell indítanunk a számítógépet, mert olyannyira leállítottunk mindent, hogy mi sem igazán fogunk tudni dolgozni.

Érdekes módon, a `Get-Service` támogatja a távoli gépek rendszerszolgáltatásainak lekérdezését a `Computername` paraméter megadásával, de a `Start-Service` és `Stop-Service` nem. Természetesen ezt is megtehetjük, ha használjuk a PowerShell vagy a WMI távoli futtatási lehetőségeit, használatukkal lekérdezhetők és felügyelhetők a távoli gépeken futó folyamatok és szolgáltatások is.

```
[11] PS C:\> Invoke-Command -ComputerName member -ScriptBlock {start-service au
diosrv}
```

A fenti példában az `Invoke-Command` cmdletet alkalmaztam. Nézzünk egy WMI-s megoldást is:

```
[13] PS C:\> Get-WmiObject -ComputerName member -Class win32 service -Filter "n
ame = 'audiosrv'" | Invoke-WmiMethod -Name StopService


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
```

2.11.2.1 Szolgáltatások Startup tulajdonsága

Ha már `get-service`, akkor nézzük meg, hogy az előbb látott hasznos `CanStop` tulajdonság mellett, vajon van-e `StartupType`, vagy valami hasonló tulajdonsága a szolgáltatás-objektumoknak? Nézzük meg a tulajdonság jellegű tagjellemzőket:

```
[15] PS C:\> get-service audiosrv | get-member -MemberType properties


   TypeName: System.ServiceProcess.ServiceController

Name                MemberType    Definition
----                ----------    ----------
Name                AliasProperty Name = ServiceName
RequiredServices    AliasProperty RequiredServices = ServicesDependedOn
CanPauseAndContinue Property      System.Boolean CanPauseAndContinue {get;}
CanShutdown         Property      System.Boolean CanShutdown {get;}
CanStop             Property      System.Boolean CanStop {get;}
Container           Property      System.ComponentModel.IContainer Containe...
DependentServices   Property      System.ServiceProcess.ServiceController[]...
DisplayName         Property      System.String DisplayName {get;set;}
MachineName         Property      System.String MachineName {get;set;}
ServiceHandle       Property      System.Runtime.InteropServices.SafeHandle...
ServiceName         Property      System.String ServiceName {get;set;}
ServicesDependedOn  Property      System.ServiceProcess.ServiceController[]...
ServiceType         Property      System.ServiceProcess.ServiceType Service...
Site                Property      System.ComponentModel.ISite Site {get;set;}
Status              Property      System.ServiceProcess.ServiceControllerSt...
```

Engem igazából az indulási állapot érdekelne (Automatic, Manual, Disabled). Nézzük meg, hogy pl. a sokat sejtető `ServiceType` vajon ezt rejti-e?

```
[18] PS C:\> get-service audiosrv | Format-Table -Property Name, Status, Servic
eType -auto

Name      Status       ServiceType
----      ------       -----------
audiosrv Stopped Win32ShareProcess
```

Sajnos nem. Hát ez bosszantó! Annál is inkább, mert a `set-service` cmdlet vidáman tartalmaz erre a tulajdonságra vonatkozó beállítási lehetőséget:

```
[19] PS C:\> get-help Set-Service

NAME
    Set-Service

SYNOPSIS
    Starts, stops, and suspends a service, and changes its properties.


SYNTAX
    Set-Service [-Name] <string> [-Description <string>] [-DisplayName <string
    >] [-PassThru] [-StartupType {Automatic | Manual | Disabled}] [-Status <st
    ring>] [-ComputerName <string[]>] [-Confirm] [-WhatIf] [<CommonParameters>
    ]
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    Set-Service [-Description <string>] [-DisplayName <string>] [-InputObject
    <ServiceController>] [-PassThru] [-StartupType {Automatic | Manual | Disab
    led}] [-Status <string>] [-ComputerName <string[]>] [-Confirm] [-WhatIf] [
    <CommonParameters>]
...
```

Na de WMI-ből a `StartupType` adathoz is hozzáférhetünk, ugyan ott `StartMode`-nak hívják:

```
PS I:\> $service=[WMI] "Win32_Service.Name=""Alerter"""
PS I:\> $service.StartMode
Disabled
```

Hogyan lehetne azt elérni, hogy ne kelljen két menetben kiszedni egy szolgáltatás adatait (egy `get-service` PowerShell cmdlet és egy WMI lekérdezés), hanem valahogy ezeket egyben látni? Szerencsére a PowerShell annyira rugalmas, hogy még ezt is meg lehet tenni az *1.4.13 Osztályok (típusok) testre szabása* fejezetben látott módon, *types.ps1xml* fájl létrehozásával.

Létrehoztam egy `soost.service.ps1xml` fájlt ugyanazon elérési úton, mint ahol az eredeti `types.ps1xml` is van.

```
<Types>
   <Type>
      <Name>System.ServiceProcess.ServiceController</Name>
      <Members>
        <ScriptProperty>
          <Name>StartupType</Name>
          <GetScriptBlock>
              ([Wmi] "Win32_Service.Name=""$($this.Name)""").StartMode
          </GetScriptBlock>
        </ScriptProperty>
      </Members>
   </Type>
</Types>
```

A szerkezet majdnem magáért beszél. Egyrészt a "`GetScriptBlock`" szorul magyarázatra. Általában egy tulajdonságot kiolvashatunk, esetleg értéket is adhatunk neki. Én itt ebben a fájlban csak kiolvasási viselkedését definiáltam, az értékadásit nem. (Az egyébként a `SetScriptBlock` lenne.) Másrészt itt nem `$_` változóval hivatkozunk magunkra, hanem a `$this` változóval.

Most már csak be kell etetni a rendszerbe az én típusmódosításomat és már nézhetjük is az eredményt:

```
PS C:\> Update-TypeData soost.service.ps1xml
PS C:\> (Get-Service alerter).StartupType
Disabled
```

2.11.3 WMI objektumok metódusainak meghívása (Invoke-WMIMethod)

A WMI objektumok is rendelkeznek metódusokkal. Ezek felderítésére használhatjuk a PowerShell `get-member` cmdletjét a `-list` kapcsolóval. Például nézzük az előbb lekérdezett hálózati kártyák WMI metódusait:

```
[8] PS C:\> Get-WmiObject -Namespace root\cimv2 -Class win32_networkadapter -li
st | Format-List *


Name              : Win32_NetworkAdapter
__GENUS           : 1
__CLASS           : Win32_NetworkAdapter
__SUPERCLASS      : CIM_NetworkAdapter
__DYNASTY         : CIM_ManagedSystemElement
__RELPATH         : Win32_NetworkAdapter
__PROPERTY_COUNT  : 40
__DERIVATION      : {CIM_NetworkAdapter, CIM_LogicalDevice, CIM_LogicalElement,
                    CIM_ManagedSystemElement}
__SERVER          : DC
__NAMESPACE       : ROOT\cimv2
__PATH            : \\DC\ROOT\cimv2:Win32_NetworkAdapter
Path              : \\DC\ROOT\cimv2:Win32_NetworkAdapter
Derivation        : {CIM_NetworkAdapter, CIM_LogicalDevice, CIM_LogicalElement,
                    CIM_ManagedSystemElement}
Methods           : {SetPowerState, Reset, Enable, Disable}
Scope             : System.Management.ManagementScope
Options           : System.Management.ObjectGetOptions
ClassPath         : \\DC\ROOT\cimv2:Win32_NetworkAdapter
Properties        : {AdapterType, AdapterTypeId, AutoSense, Availability, Capti
                    on, ConfigManagerErrorCode, ConfigManagerUserConfig, Creati
                    onClassName, Description, DeviceID...}
SystemProperties  : {__GENUS, __CLASS, __SUPERCLASS, __DYNASTY, __RELPATH, __PR
                    OPERTY_COUNT, __DERIVATION, __SERVER, __NAMESPACE, __PATH}
Qualifiers        : {dynamic, Locale, provider, UUID}
Site              :
Container         :
```

Látható, hogy a `Methods` tulajdonság tartalmazza az adott WMI osztály objektumaira meghívható metódusok listáját. Nézzük meg ezeket kicsit részletesebben:

```
[9] PS C:\> (Get-WmiObject -Namespace root\cimv2 -Class win32_networkadapter -l
ist).methods


Name          : SetPowerState
InParameters  : System.Management.ManagementBaseObject
OutParameters : System.Management.ManagementBaseObject
Origin        : CIM_LogicalDevice
Qualifiers    : {}

Name          : Reset
InParameters  :
OutParameters : System.Management.ManagementBaseObject
Origin        : CIM_LogicalDevice
Qualifiers    : {}

Name          : Enable
InParameters  :
OutParameters : System.Management.ManagementBaseObject
Origin        : Win32_NetworkAdapter
Qualifiers    : {Implemented, MappingStrings}

Name          : Disable
InParameters  :
OutParameters : System.Management.ManagementBaseObject
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Origin          : Win32_NetworkAdapter
Qualifiers      : {Implemented, MappingStrings}
```

Például rendelkezésünkre áll a `Disable` metódus, amellyel az adott hálózati kártyát tudjuk kikapcsolni. Ezt meghívni az `Invoke-WMIMethod` cmdlettel tudjuk legegyszerűbben:

```
[22] PS C:\> Get-WmiObject -Namespace root\cimv2 -Class Win32_NetworkAdapter -F
ilter "netconnectionid = 'Local Area Connection'" | Invoke-WmiMethod -Name Disa
ble


__GENUS           : 2
__CLASS           : __PARAMETERS
__SUPERCLASS      :
__DYNASTY         : __PARAMETERS
__RELPATH         :
__PROPERTY_COUNT  : 1
__DERIVATION      : {}
__SERVER          :
__NAMESPACE       :
__PATH            :
ReturnValue       : 0
```

Látható, hogy én a csővezetéken keresztül töltöttem fel az `Invoke-WMIMethod` azon paraméterét, ami megmutatta számára, hogy melyik objektumon is kívánom ezt a metódust meghívni. Használhattam volna magának az `Invoke-WMIMethod` paraméterezését is erre, de szerintem ez a csövezéses módszer egyszerűbb.

Fontos nyomon követni, hogy milyen visszatérési értéket is ad a metódus a meghívása után. Jelen esetben a `ReturnValue` 0 volt, ez jelzi, hogy nem történt hiba. Egyéb esetben meg kellett volna vizsgálni, hogy mi a hiba oka.

Nem minden metódus meghívása lesz sikeres az `Invoke-WMIMethod` cmdlet segítségével, mert sok esetben a WMI nem a legmagasabb szintű jogosultsággal futtatja ezeket biztonsági megfontolásokból. Ha ilyen jellegű hibára gyanakszunk, akkor a cmdletnek adjunk még egy `-EnableAllPrivileges` kapcsolót is.

Az előző metódus egyszerű volt, hiszen nem igényelt paramétert. Nehezebb a helyzetünk, ha olyan WMI metódussal állunk szemben, amelyik paramétereket vár. Ilyen például a megosztások létrehozása. Nézzük meg, hogyan jutnánk el a megfelelő metódushoz:

```
[1] PS C:\> (Get-WmiObject -Namespace root\cimv2 -Class Win32_Share -List).meth
ods


Name            : Create
InParameters    : System.Management.ManagementBaseObject
OutParameters   : System.Management.ManagementBaseObject
Origin          : Win32_Share
Qualifiers      : {Constructor, Implemented, MappingStrings, Static}

Name            : SetShareInfo
InParameters    : System.Management.ManagementBaseObject
OutParameters   : System.Management.ManagementBaseObject
Origin          : Win32_Share
Qualifiers      : {Implemented, MappingStrings}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name           : GetAccessMask
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Implemented, MappingStrings}

Name           : Delete
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Destructor, Implemented, MappingStrings}
```

Sajnos nem sok segítséget ad nekünk ez a lista a paraméterezésre vonatkozólag. Egy másik módszer a következő. Definiálok egy új `Win32_Share WMIClass` típusú objektumot és annak megnézem a tagjellemzőit:

```
[2] PS C:\> $wc = New-Object wmiclass Win32_Share
[3] PS C:\> $wc | gm


   TypeName: System.Management.ManagementClass#ROOT\cimv2\Win32_Share

Name                MemberType    Definition
----                ----------    ----------
Name                AliasProperty Name = __Class
Create              Method        System.Management.ManagementBaseObject Cr...
__CLASS             Property      System.String __CLASS {get;set;}
__DERIVATION        Property      System.String[] __DERIVATION {get;set;}
__DYNASTY           Property      System.String __DYNASTY {get;set;}
__GENUS             Property      System.Int32 __GENUS {get;set;}
__NAMESPACE         Property      System.String __NAMESPACE {get;set;}
__PATH              Property      System.String __PATH {get;set;}
__PROPERTY_COUNT    Property      System.Int32 __PROPERTY_COUNT {get;set;}
__RELPATH           Property      System.String __RELPATH {get;set;}
__SERVER            Property      System.String __SERVER {get;set;}
__SUPERCLASS        Property      System.String __SUPERCLASS {get;set;}
ConvertFromDateTime ScriptMethod  System.Object ConvertFromDateTime();
ConvertToDateTime   ScriptMethod  System.Object ConvertToDateTime();
```

A fenti listában látható „`Create`” a `Win32_Share` WMI osztály statikus metódusának fogható fel. Az előzőekben látott négy metódus pedig a „normál”, azaz példány-metódusoknak felelnek meg. Ennek a statikus metódusnak a paraméterezését a következő módon tudjuk megnézni, a tényleges paraméterezés a `Value` mellett látható:

```
[9] PS C:\> $wc.create


MemberType          : Method
OverloadDefinitions : {System.Management.ManagementBaseObject Create(System.St
                      ring Path, System.String Name, System.UInt32 Type, Syste
                      m.UInt32 MaximumAllowed, System.String Description, Syst
                      em.String Password, System.Management.ManagementObject#W
                      in32_SecurityDescriptor Access)}
TypeNameOfValue     : System.Management.Automation.PSMethod
Value               : System.Management.ManagementBaseObject Create(System.Str
                      ing Path, System.String Name, System.UInt32 Type, System
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                      .UInt32 MaximumAllowed, System.String Description, Syste
                      m.String Password, System.Management.ManagementObject#Wi
                      n32_SecurityDescriptor Access)
Name                : Create
IsInstance          : True
```

Szerencsére nem minden paramétert kötelező kitölteni, így egy egyszerű megosztás a következő kifejezéssel is létrehozható:

```
[10] PS C:\> $wc.Create("c:\munka","MunkaShare",0)


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
```

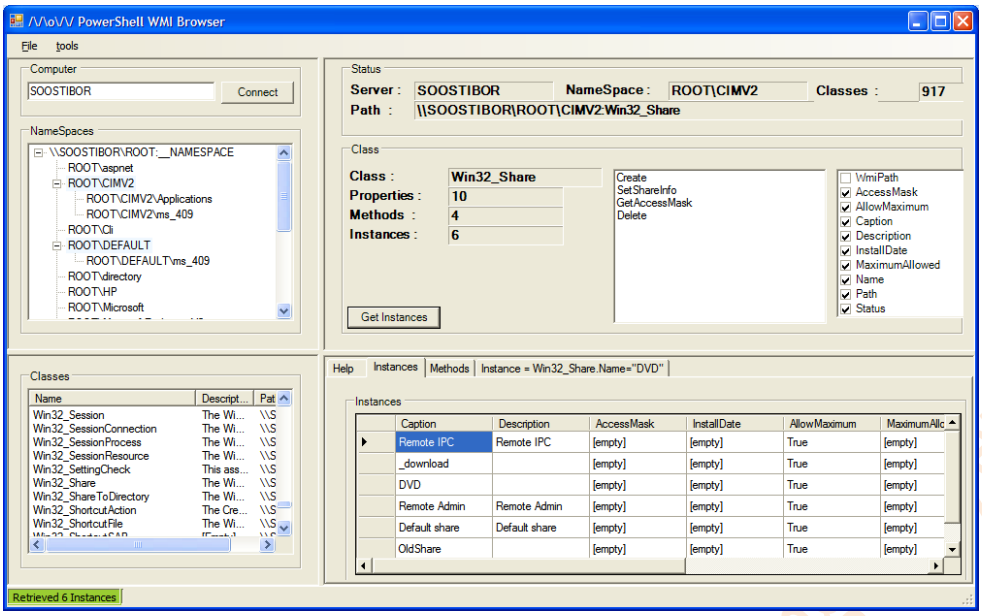
Láthatjuk a 0-s visszatérési értékből, hogy a művelet sikeres volt, így kilistázhatjuk az aktuális megosztásokat, melyek között ott az új megosztásunk:

```
[11] PS C:\> Get-WmiObject win32_share

Name                        Path                        Description
----                        ----                        -----------
ADMIN$                      C:\Windows                  Remote Admin
C$                          C:\                         Default share
IPC$                                                    Remote IPC
MunkaShare                  c:\munka
NETLOGON                    C:\Windows\SYSVOL\sysv...   Logon server share
SYSVOL                      C:\Windows\SYSVOL\sysvol    Logon server share
```

A metódusok böngészésére is alkalmas másik eszköz a *PowerShell WMI Browser*, vagy más néven a *WMIExplorer.ps1*:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



68. ábra PowerShell WMI Browser

Ez az eszköz azért is érdekes, mert színtisztán PowerShellben van megírva! Érdemes belenézni a szkript forráskódjába, látható benne, hogy hogyan szólítja meg a .NET keretrendszer grafikus osztályait, melyek segítségével felépíti a fenti képen látható ablakot a sok vezérlőelemmel együtt. Persze a PowerShellt nem grafikus alkalmazások írására találták ki, így ilyeneket nem túl egyszerű létrehozni. A PowerShell WMI Browser alkotója, a „The PowerShell Guy” sem Notepad előtt ülve pötyögte be a programsorokat, hanem egy C#-ban megírt programot a forráskód alapján egy szkript segítségével alakított át színtiszta PowerShell szkriptté.

2.11.4 WMI objektumok tulajdonságainak módosítása, új objektumok létrehozása és eltávolítása (Set-WMIInstance, Remove-WMIObject)

Új WMI példányok és meglevők tulajdonságainak módosítására a `Set-WMIInstance` cmdletet használhatjuk. Ahhoz, hogy ezt megfelelően használhassuk, meg kell tudnunk a keresett WMI objektum példány elérési útját. Ezt azért fontos, mert egy WMI osztálynak több példánya is lehet, így nem mindegy, hogy melyiknek módosítjuk a tulajdonságait. Elsőként tehát keressük meg az objektumot, jelen esetben példaként a Restore Point (helyreállítási pontok) konfigurációját tartalmazó WMI objektumot:

```
PS C:\> Get-WmiObject -Namespace ROOT\DEFAULT -Class systemrestoreconfig


__GENUS            : 2
__CLASS            : SystemRestoreConfig
__SUPERCLASS       :
__DYNASTY          : SystemRestoreConfig
__RELPATH          : SystemRestoreConfig.MyKey="SR"
__PROPERTY_COUNT   : 5
__DERIVATION       : {}
__SERVER           : TIBI-PC
__NAMESPACE        : ROOT\DEFAULT
__PATH             : \\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey="SR"
DiskPercent        : 15
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
MyKey             : SR
RPGlobalInterval  :
RPLifeInterval    :
RPSessionInterval : 1
```

Ez azért kell nekem, hogy beállíthassam, hogy a helyreállítási pontokat mennyi ideig őrizze a rendszer. Ezt az `RPLifeInterval` tulajdonság módosításával tehetem be, itt másodpercekben kell megadni az időt. A `Set-WMIInstace` cmdletnek a `-Path` paraméterének az előbb látott keresés `__PATH` adatát kell átadni, ezzel teljesen egyértelmű, hogy melyik objektumot is akarjuk módosítani:

```
PS C:\> Set-WmiInstance -Path '\\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey
="SR"' -arguments @{RPLifeInterval = 864000}


__GENUS           : 2
__CLASS           : SystemRestoreConfig
__SUPERCLASS      :
__DYNASTY         : SystemRestoreConfig
__RELPATH         : SystemRestoreConfig.MyKey="SR"
__PROPERTY_COUNT  : 5
__DERIVATION      : {}
__SERVER          : TIBI-PC
__NAMESPACE       : ROOT\DEFAULT
__PATH            : \\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey="SR"
DiskPercent       : 15
MyKey             : SR
RPGlobalInterval  :
RPLifeInterval    : 864000
RPSessionInterval : 1
```

Az `arguments` paraméternek egy olyan hashtáblát kell átadni, amelyben a tulajdonságnév – érték párok vannak.

Nézzük meg, hogy mi történt volna, ha csak az osztályt határozzuk meg a `Set-WMIInstance` számára:

```
PS C:\> set-Wmiinstance -Namespace ROOT\DEFAULT -Class systemrestoreconfig -Arg
uments @{RPLifeInterval = 864000}


__GENUS           : 2
__CLASS           : SystemRestoreConfig
__SUPERCLASS      :
__DYNASTY         : SystemRestoreConfig
__RELPATH         : SystemRestoreConfig.MyKey="{0F45EB5A-31C3-4161-BB1A-3CCFC2
                    50DC20}"
__PROPERTY_COUNT  : 5
__DERIVATION      : {}
__SERVER          : TIBI-PC
__NAMESPACE       : ROOT\DEFAULT
__PATH            : \\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey="{0F45EB5
                    A-31C3-4161-BB1A-3CCFC250DC20}"
DiskPercent       :
MyKey             : {0F45EB5A-31C3-4161-BB1A-3CCFC250DC20}
RPGlobalInterval  :
RPLifeInterval    : 864000
RPSessionInterval :
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Látszólag minden úgy történt majdnem, mint eddig, de ha újra kilistázom a `SystemRestoreConfig` osztály elemeit, akkor láthatjuk, hogy most nem a meglevő elemet módosítottunk, hanem egy új objektumot hoztunk létre:

```
PS C:\> Get-WmiObject -Namespace ROOT\DEFAULT -Class systemrestoreconfig


__GENUS           : 2
__CLASS           : SystemRestoreConfig
__SUPERCLASS      :
__DYNASTY         : SystemRestoreConfig
__RELPATH         : SystemRestoreConfig.MyKey="{0F45EB5A-31C3-4161-BB1A-3CCFC2
                    50DC20}"
__PROPERTY_COUNT  : 5
__DERIVATION      : {}
__SERVER          : TIBI-PC
__NAMESPACE       : ROOT\DEFAULT
__PATH            : \\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey="{0F45EB5
                    A-31C3-4161-BB1A-3CCFC250DC20}"
DiskPercent       :
MyKey             : {0F45EB5A-31C3-4161-BB1A-3CCFC250DC20}
RPGlobalInterval  :
RPLifeInterval    : 864000
RPSessionInterval :

__GENUS           : 2
__CLASS           : SystemRestoreConfig
__SUPERCLASS      :
__DYNASTY         : SystemRestoreConfig
__RELPATH         : SystemRestoreConfig.MyKey="SR"
__PROPERTY_COUNT  : 5
__DERIVATION      : {}
__SERVER          : TIBI-PC
__NAMESPACE       : ROOT\DEFAULT
__PATH            : \\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKey="SR"
DiskPercent       : 15
MyKey             : SR
RPGlobalInterval  :
RPLifeInterval    : 864000
RPSessionInterval : 1
```

A nem szükséges példányt a `Remove-WMIObject` cmdlettel tudjuk eltávolítani, itt is a pontos elérési utat kell megadni az objektum pontos azonosítása érdekében:

```
PS C:\> Remove-WmiObject -Path '\\TIBI-PC\ROOT\DEFAULT:SystemRestoreConfig.MyKe
y="{0F45EB5A-31C3-4161-BB1A-3CCFC250DC20}"'
```

2.11.5 Fontosabb WMI osztályok

A következőkben néhány egyszerűbb példát válogattam össze a WMI legkülönbözőbb területeiről, amelyeket gyakrabban használjuk gépünkkel kapcsolatos információk kinyerésére:

Alapvető számítógép-információk kijelzése:

```
[45] PS I:\>get-wmiobject -class "Win32_ComputerSystem" -namespace "root\CIM
V2"
```

BIOS információk:

```
[46] PS I:\>get-wmiobject -class "Win32_BIOS" -namespace "root\CIMV2"
```

Alaplap információk:

```
[47] PS I:\>get-wmiobject -class "Win32_BaseBoard" -namespace "root\CIMV2"
```

Számítógép házának információi (pl. sorozatszám):

```
[48] PS I:\>get-wmiobject -class "Win32_SystemEnclosure" -namespace "root\CI
MV2"
```

Processzor-információk:

```
[53] PS I:\>get-wmiobject -class "Win32_Processor" -namespace "root\CIMV2"
```

Részletes memóriainformációk (blokkok szintjén is):

```
[55] PS I:\>get-wmiobject -class "Win32_PhysicalMemory" -namespace "root\CIM
V2"
```

Plug'n'Play eszközök:

```
[59] PS I:\>get-wmiobject -class "Win32_PnPEntity" -namespace "root\CIMV2"
```

Videokártya információk:

```
[61] PS I:\>get-wmiobject -class "Win32_DisplayConfiguration" -namespace "ro
ot\CIMV2"
```

Eventlog állományok:

```
[64] PS I:\>get-wmiobject -class "Win32_NTEventlogFile" -namespace "root\CIM
V2"
```

Eventlog állományok távoli gépről (lásd *2.6 Az Eseménynapló feldolgozása* fejezet):

```
[64] PS I:\>get-wmiobject -class "Win32_NTEventlogFile" -namespace "root\CIM
V2" -credential IQJB\soostibor -computer J-CRM
```

Hálózati adapterek konfigurációja:

```
[66] PS I:\>get-wmiobject -class "Win32_NetworkAdapterConfiguration" -namesp
ace "root\CIMV2"
```

Login információk:

```
[68] PS I:\>get-wmiobject -class "Win32_NetworkLoginProfile" -namespace "roo
t\CIMV2"
```

A könyv tanfolyami felhasználása nem engedélyezett!

A Windows verzióinformációi:

```
[69] PS I:\>get-wmiobject -class "Win32_OperatingSystem" -namespace "root\CI
MV2"
```

Page-file információk:

```
[70] PS I:\>get-wmiobject -class "Win32_PageFile" -namespace "root\CIMV2"
```

Gép helyi ideje:

```
[73] PS I:\>get-wmiobject -class "Win32_LocalTime" -namespace "root\CIMV2"
```

Időzóna:

```
[74] PS I:\>get-wmiobject -class "Win32_TimeZone" -namespace "root\CIMV2"
```

Printer-információk:

```
[76] PS I:\>get-wmiobject -class "Win32_Printer" -namespace "root\CIMV2"
```

Telepített javítócsomagok:

```
[77] PS I:\>get-wmiobject -class "Win32_QuickFixEngineering" -namespace "roo
t\CIMV2"
```

Partíciók:

```
[78] PS I:\>get-wmiobject -class "Win32_DiskPartition" -namespace "root\CIMV
2"
```

Logikai meghajtók:

```
[79] PS I:\>get-wmiobject -class "Win32_LogicalDisk" -namespace "root\CIMV2"
```

Meghajtók:

```
[80] PS I:\>get-wmiobject -class "Win32_DiskDrive" -namespace "root\CIMV2"
```

Biztonsági szoftverek:

```
PS C:\> "AntiSpywareProduct", "AntiVirusProduct", "FirewallProduct" | ForEach-O
bject {Get-WmiObject -Namespace root\SecurityCenter -Class $_} | ForEach-Object
 {$_.displayname}
Microsoft Forefront Client Security
Windows Defender
Microsoft Forefront Client Security
```

A könyv tanfolyami felhasználása nem engedélyezett!

2.12 Teljesítmény-monitorozás (Get-Counter, Export-, Import-Counter)

A PowerShell 2.0 már tartalmaz a teljesítmény-számlálók lekérdezésére és megjelenítésére alkalmas cmdleteket, melyek az alábbi táblázatban láthatók:

```
[77] PS C:\> Get-Command -noun counter

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Export-Counter                     Export-Counter [-Path] <Str...
Cmdlet          Get-Counter                        Get-Counter [[-Counter] <St...
Cmdlet          Import-Counter                     Import-Counter [-Path] <Str...
```

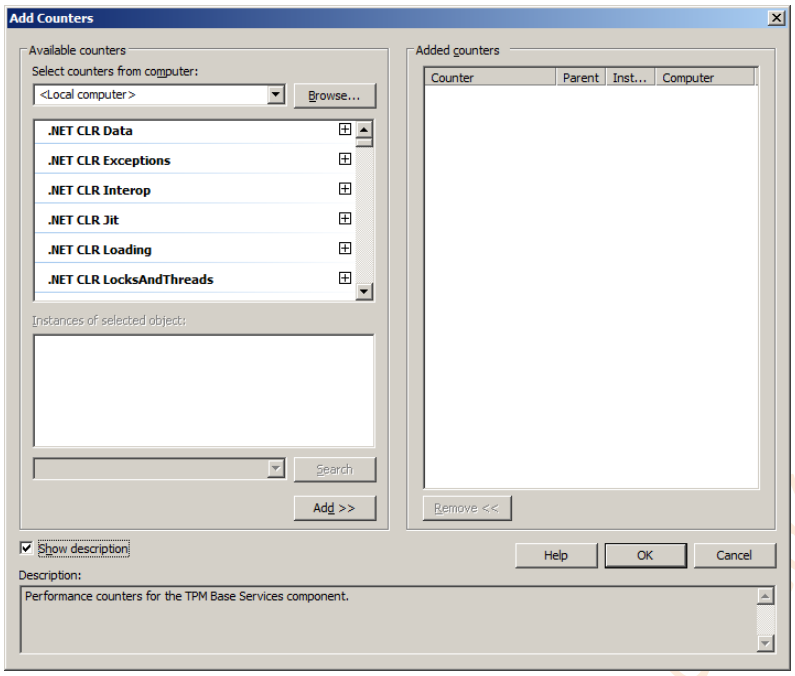
Ha csak magában, mindenféle paraméterezés nélkül használjuk a `Get-Counter` cmdletet, az alábbi kimenetet kapjuk:

```
[22] PS C:\> Get-Counter

Timestamp                 CounterSamples
---------                 --------------
2010. 01. 24. 16:45:34    \\dc\network interface(intel[r] pro_1000 mt desktop
                          adapter)\bytes total/sec :
                          41,6342057540426

                          \\dc\network interface(isatap.{ff327d73-8ae0-4949-98
                          c2-bad7dabfc3cb})\bytes total/sec :
                          0

                          \\dc\network interface(teredo tunneling pseudo-inter
                          face)\bytes total/sec :
                          0

                          \\dc\processor(_total)\% processor time :
                          13,8461538461538

                          \\dc\memory\% committed bytes in use :
                          50,7137807497772

                          \\dc\memory\cache faults/sec :
                          0

                          \\dc\physicaldisk(_total)\% disk time :
                          0

                          \\dc\physicaldisk(_total)\current disk queue length
                          :
                          0
```

Látható, hogy már alapesetben is kapunk kimenetet a legfontosabbnak ítélt teljesítmény-számlálókból. Nézzük meg, hogy hogyan lehet tájékozódni a teljesítményszámláló kategóriák és maguk a számlálók között. Ehhez emlékeztetőül nézzük meg a grafikus *Performace Monitor* alkalmazás felületét, ahol ki lehet választani, hogy mely számlálókat kívánjuk megjeleníteni:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



69. ábra Performance Monitor adatai

A bal felső listában található kategóriákhoz a következő paranccsal jutunk el:

```
[21] PS C:\> Get-Counter -ListSet * | sort-object -property countersetname | Fo
rmat-Table -Property countersetname

CounterSetName
--------------
.NET CLR Data
.NET CLR Exceptions
.NET CLR Interop
.NET CLR Jit
.NET CLR Loading
.NET CLR LocksAndThreads
.NET CLR Memory
.NET CLR Networking
.NET CLR Remoting
.NET CLR Security
.NET Data Provider for Oracle
.NET Data Provider for SqlServer
ADWS
Authorization Manager Applications
BatteryStatus
…
```

Mielőtt ebbe belemennénk, nézzünk meg egy számláló-csoportot:

```
[29] PS C:\> Get-Counter -ListSet "Network Interface"


CounterSetName     : Network Interface
MachineName        : .
CounterSetType     : MultiInstance
Description        : The Network Interface performance object consists of coun
                     ters that measure the rates at which bytes and packets ar
                     e sent and received over a TCP/IP network connection.  It
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
                      includes counters that monitor connection errors.
Paths                 : {\Network Interface(*)\Bytes Total/sec, \Network Interfac
                        e(*)\Packets/sec, \Network Interface(*)\Packets Received/
                        sec, \Network Interface(*)\Packets Sent/sec, \Network Int
                        erface(*)\Current Bandwidth, \Network Interface(*)\Bytes
                        Received/sec, \Network Interface(*)\Packets Received Unic
                        ast/sec, \Network Interface(*)\Packets Received Non-Unica
                        st/sec, \Network Interface(*)\Packets Received Discarded,
                         \Network Interface(*)\Packets Received Errors...}
PathsWithInstances    : {\Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)
                        \Bytes Total/sec, \Network Interface(isatap.{FF327D73-8AE
                        0-4949-98C2-BAD7DABFC3CB})\Bytes Total/sec, \Network Inte
                        rface(Teredo Tunneling Pseudo-Interface)\Bytes Total/sec,
                         \Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)
                        \Packets/sec, \Network Interface(isatap.{FF327D73-8AE0-49
                        49-98C2-BAD7DABFC3CB})\Packets/sec, \Network Interface(Te
                        redo Tunneling Pseudo-Interface)\Packets/sec, \Network In
                        terface(Intel[R] PRO_1000 MT Desktop Adapter)\Packets Rec
                        eived/sec, \Network Interface(isatap.{FF327D73-8AE0-4949-
                        98C2-BAD7DABFC3CB})\Packets Received/sec, \Network Interf
                        ace(Teredo Tunneling Pseudo-Interface)\Packets Received/s
                        ec, \Network Interface(Intel[R] PRO_1000 MT Desktop Adapt
                        er)\Packets Sent/sec...}
Counter               : {\Network Interface(*)\Bytes Total/sec, \Network Interfac
                        e(*)\Packets/sec, \Network Interface(*)\Packets Received/
                        sec, \Network Interface(*)\Packets Sent/sec, \Network Int
                        erface(*)\Current Bandwidth, \Network Interface(*)\Bytes
                        Received/sec, \Network Interface(*)\Packets Received Unic
                        ast/sec, \Network Interface(*)\Packets Received Non-Unica
                        st/sec, \Network Interface(*)\Packets Received Discarded,
                         \Network Interface(*)\Packets Received Errors...}
```

Látszik, hogy elég összetett tulajdonságokkal rendelkező objektum a kimenet. A `Counter` és a `Paths` tulajdonságok ugyanazt tartalmazzák, merthogy a `Counter` a `Paths` álnév-tulajdonsága. Ezekben az adott kategóriába tartozó számlálókat találjuk az összes példányra nézve. Itt konkrétan az összes hálózati interfész együttes `Bytes Total/sec`, `Packets/sec`, stb. számlálóit tudnánk megnézni.

Itt tehát még konkrét számlálóértékekről nincs szó, csak a kategóriák és azon belüli számlálók felderítéséről. Ha az egyes példányok számlálóit szeretnénk felderíteni, akkor a `PathsWithInstances` tulajdonságokat kell lekérdezni:

```
[38] PS C:\> (Get-Counter -ListSet "Network Interface").pathswithinstances
\Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)\Bytes Total/sec
\Network Interface(isatap.{FF327D73-8AE0-4949-98C2-BAD7DABFC3CB})\Bytes Total/
sec
\Network Interface(Teredo Tunneling Pseudo-Interface)\Bytes Total/sec
\Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)\Packets/sec
…
```

Ha így felderítettük a számlálókat, akkor a tényleges teljesítményadathoz a `Get-Counter` egy másik paraméterezésével jutunk el:

```
[41] PS C:\> Get-Counter -Counter "\Network Interface(Intel[R] PRO_1000 MT Desk
top Adapter)\Bytes Total/sec"

Timestamp                 CounterSamples
---------                 --------------
```

```
2010. 01. 24. 17:23:54    \\dc\network interface(intel[r] pro_1000 mt desktop
                          adapter)\bytes total/sec :
                          129,285069405517
```

Itt is a számadat eléggé „belül” található, a `CounterSamples` tulajdonsággal jutunk hozzá egy kicsit közelebb:

```
[47] PS C:\> (Get-Counter -Counter "\Network Interface(Intel[R] PRO_1000 MT Des
ktop Adapter)\Bytes Total/sec").countersamples

Path                        InstanceName                           CookedValue
----                        ------------                           -----------
\\dc\network interface(... intel[r] pro_1000 mt d...          179,232170109183
```

De még itt sem a szám van, még mélyebbre kell ásnunk:

```
[48] PS C:\> (Get-Counter -Counter "\Network Interface(Intel[R] PRO_1000 MT Des
ktop Adapter)\Bytes Total/sec").countersamples[0].cookedvalue
132,790382777453
```

A `CounterSamples` általában lehet tömb is, ha egyszerre több számláló több mintavételezését kérjük:

```
[57] PS C:\> $minták = Get-Counter -Counter "\Network Interface(Intel[R] PRO_10
00 MT Desktop Adapter)\Bytes Total/sec", "\Network Interface(Intel[R] PRO_1000
MT Desktop Adapter)\Current Bandwidth" -SampleInterval 1 -MaxSamples 5
[58] PS C:\> $minták[0]

Timestamp                 CounterSamples
---------                 --------------
2010. 01. 24. 17:37:39    \\dc\network interface(intel[r] pro_1000 mt desktop
                          adapter)\bytes total/sec :
                          153,4636610096

                          \\dc\network interface(intel[r] pro_1000 mt desktop
                          adapter)\current bandwidth :
                          1000000000
```

A fenti példában 5 mintavétet kértem 1 másodperces mintavételi idővel, egyszerre két számlálóról. Az így kapott tömb 0. eleme tartalmazza az első mintavét idején készült mindkét számlálót, így az első időpontban az első számláló értékét az alábbi kifejezéssel kapjuk meg:

```
[65] PS C:\> $minták[0].countersamples[0].cookedvalue
153,4636610096
```

Ha egy komplett táblázatot szeretnénk kapni az értékekből, akkor a következő kis szkriptet használhatjuk:

```
$minták = Get-Counter -Counter `
"\Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)\Bytes Total/sec",
"\Network Interface(Intel[R] PRO_1000 MT Desktop Adapter)\Current Bandwidth" `
-SampleInterval 1 -MaxSamples 5

$tábla = @()
$minták | %{ $táblaelem = New-Object psobject
    $_.countersamples | %{$táblaelem |
          add-member -name $_.path -membertype NoteProperty `
```

A könyv tanfolyami felhasználása nem engedélyezett!

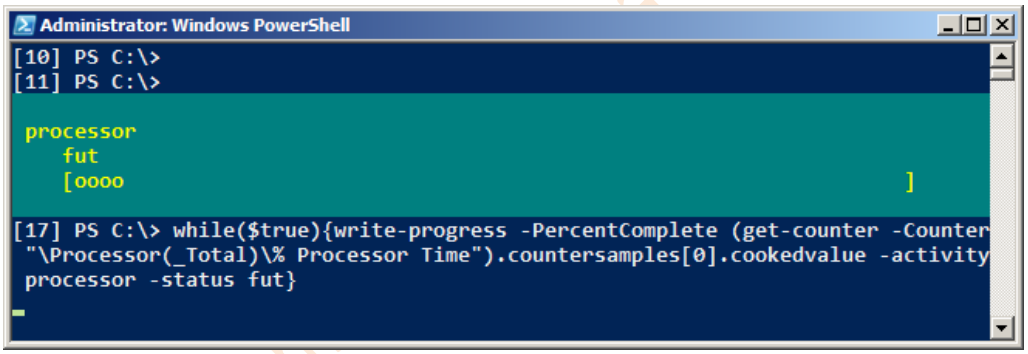
© Soós Tibor – soost@ IQJB.hu

```
            -value $_.cookedvalue}
      $táblaelem |
            Add-Member -Name TimeStamp -MemberType NoteProperty `
            -Value $_.timestamp
      $tábla += $táblaelem
}
$tábla
```

És ennek kimenete:

```
\\dc\network interface(int \\dc\network interface(in TimeStamp
el[r] pro_1000 mt desktop  tel[r] pro_1000 mt deskto
  adapter)\bytes total/sec p adapter)\current bandwi
                                              dth
-------------------------- -------------------------- ---------
          153,505227985749               1000000000 2010. 01. 24. 21:28:59
          152,063724868109               1000000000 2010. 01. 24. 21:29:00
          152,040846660975               1000000000 2010. 01. 24. 21:29:01
          151,857656129348               1000000000 2010. 01. 24. 21:29:02
          152,206731700545               1000000000 2010. 01. 24. 21:29:03
```

A következő kis példa egy „grafikus” megjelenítését adja a processzorterhelést mutató teljesítményszámlálónak a PowerShell „progressbar” megjelenítője segítségével:



70. ábra Karakteres Performance Monitor

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

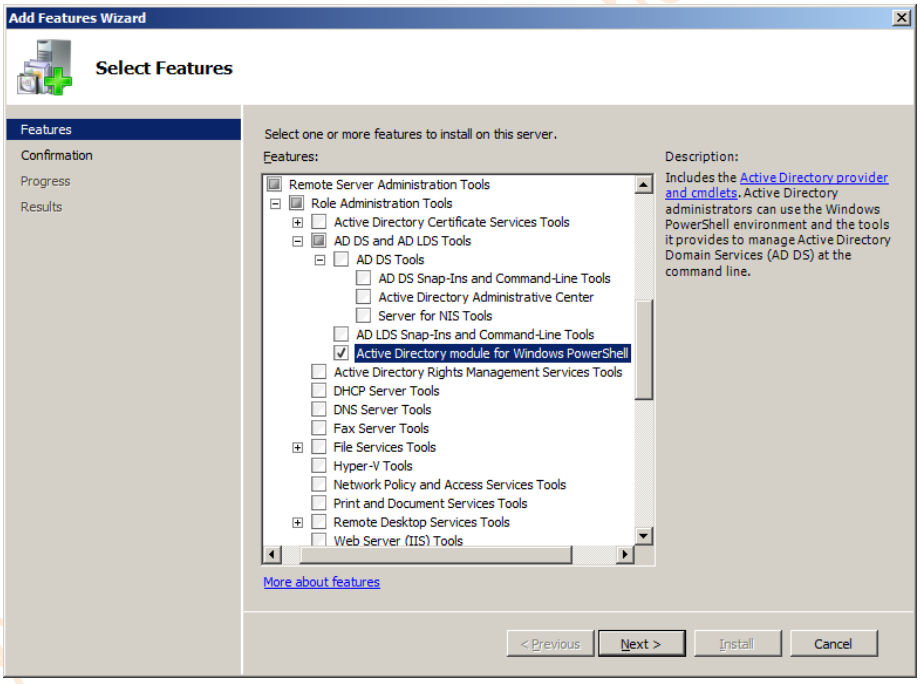
2.13 Az ActiveDirectory modul

Bár igazából ez a modul nem az alap PowerShell része, de a Windows rendszerek üzemeltetőinek az Active Directory (újabb nevén Active Directory Domain Services) annyira fontos terület, hogy fontosnak tartottam ebben a könyvben ismertetni.

A PowerShell 2.0 egyik legjelentősebb gyakorlati újdonsága az Active Directory-val kapcsolatos cmdletek megjelenése. Ugyan a PowerShell 1.0 is kezelte az AD-t, de nagyon nehézkesen, „elrettentésül" az *3.5 Active Directory* függeléki fejezetben meghagytam az ősi módszer ismertetését, ki tudja, esetleg valaki rászorul erre is, ha nincs lehetőség Windows Server 2008 R2 vagy Windows 7 és a Management Gateway telepítésére, amelyekkel akár régebbi tartományvezérlők is kezelhetők az új cmdletekkel.

Ennek a fejezetnek nem célja, hogy a kedves olvasót komplett Active Directory tanfolyamban részesítse, hanem csak a PowerShell szempontjából vett érdekességeket emelem ki, az Active Directory mibenlétét, a benne tárolt adatok jellegzetességeit ismertnek tételezem fel.

Ahhoz, hogy az Active Directory-val kapcsolatos cmdletek és az AD provider elérhető legyen elsőként importálni kell az `ActiveDirectory` modult. Ezt akkor tehetjük meg, ha a Windows Server 2008 R2 rendszergazda eszközei között található *Active Directory module for Windows PowerShell* képességet telepítjük.



71. ábra Az ActiveDirectory PowerShell modul telepítése

A képesség birtokában maga az import így néz ki:

```
[22] PS C:\> Import-Module activedirectory
```

Ezzel összesen 76 új cmdletet kapunk. Mint említettem, itt most nem célom az összes ismertetése, inkább csak a használatuk sajátosságaira szeretném felhívni a figyelmet. Az egyik ilyen sajátosság, hogy minden főnév AD előtagot kapott, így ADUser, ADComputers, ADGroup, stb. főnevekkel fogunk találkozni. Ennek oka az, hogy más modulok és korábbi snap-in-ek is kezeltek AD objektumokat, így az Exchange Server 2007-nek volt

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

például `Get-User` cmdletje, és annak érdekében, hogy a két különböző, nem teljesen egyformán működő cmdlet ne okozzon zavart a scriptjeinkben, ezért ezzel a kis előtaggal teljesen egyértelműsítette a Microsoft a helyzet.

Nézzük tehát ezeket a főneveket:

```
[36] PS C:\> Get-Command -Module activedirectory | Group-Object noun

Count Name                      Group
----- ----                      -----
    3 ADComputerServiceAccount  {Add-ADComputerServiceAccount, Get-ADComput...
    3 ADDomainControllerPass... {Add-ADDomainControllerPasswordReplicationP...
    3 ADFineGrainedPasswordP... {Add-ADFineGrainedPasswordPolicySubject, Ge...
    3 ADGroupMember             {Add-ADGroupMember, Get-ADGroupMember, Remo...
    3 ADPrincipalGroupMember... {Add-ADPrincipalGroupMembership, Get-ADPrin...
    2 ADAccountExpiration       {Clear-ADAccountExpiration, Set-ADAccountEx...
    4 ADAccount                 {Disable-ADAccount, Enable-ADAccount, Searc...
    3 ADOptionalFeature         {Disable-ADOptionalFeature, Enable-ADOption...
    1 ADAccountAuthorization... {Get-ADAccountAuthorizationGroup}
    1 ADAccountResultantPass... {Get-ADAccountResultantPasswordReplicationP...
    4 ADComputer                {Get-ADComputer, New-ADComputer, Remove-ADC...
    2 ADDefaultDomainPasswor... {Get-ADDefaultDomainPasswordPolicy, Set-ADD...
    2 ADDomain                  {Get-ADDomain, Set-ADDomain}
    1 ADDomainController        {Get-ADDomainController}
    1 ADDomainControllerPass... {Get-ADDomainControllerPasswordReplicationP...
    4 ADFineGrainedPasswordP... {Get-ADFineGrainedPasswordPolicy, New-ADFin...
    2 ADForest                  {Get-ADForest, Set-ADForest}
    4 ADGroup                   {Get-ADGroup, New-ADGroup, Remove-ADGroup, ...
    7 ADObject                  {Get-ADObject, Move-ADObject, New-ADObject,...
    4 ADOrganizationalUnit      {Get-ADOrganizationalUnit, New-ADOrganizati...
    1 ADRootDSE                 {Get-ADRootDSE}
    6 ADServiceAccount          {Get-ADServiceAccount, Install-ADServiceAcc...
    4 ADUser                    {Get-ADUser, New-ADUser, Remove-ADUser, Set...
    1 ADUserResultantPasswor... {Get-ADUserResultantPasswordPolicy}
    1 ADDirectoryServer         {Move-ADDirectoryServer}
    1 ADDirectoryServerOpera... {Move-ADDirectoryServerOperationMasterRole}
    1 ADServiceAccountPassword  {Reset-ADServiceAccountPassword}
    1 ADAccountControl          {Set-ADAccountControl}
    1 ADAccountPassword         {Set-ADAccountPassword}
    1 ADDomainMode              {Set-ADDomainMode}
    1 ADForestMode              {Set-ADForestMode}
```

Elsőként pedig próbáljuk meg kinyerni a címtárban tárolt felhasználókat a `Get-ADUser` cmdlet segítségével:

```
[31] PS C:\> Get-ADUser

cmdlet Get-ADUser at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
Filter: *


DistinguishedName : CN=Administrator,CN=Users,DC=r2,DC=dom
Enabled           : True
GivenName         :
Name              : Administrator
ObjectClass       : user
ObjectGUID        : fff8a606-7f70-4744-9437-a7b41d618335
```

```
SamAccountName    : Administrator
SID               : S-1-5-21-3398938913-3940250523-927435294-500
Surname           :
UserPrincipalName :

DistinguishedName : CN=Guest,CN=Users,DC=r2,DC=dom
Enabled           : False
GivenName         :
Name              : Guest
ObjectClass       : user
ObjectGUID        : f42aba4d-8c3f-4214-82f0-8b8f275f3463
SamAccountName    : Guest
SID               : S-1-5-21-3398938913-3940250523-927435294-501
Surname           :
UserPrincipalName :

DistinguishedName : CN=krbtgt,CN=Users,DC=r2,DC=dom
Enabled           : False
GivenName         :
Name              : krbtgt
ObjectClass       : user
ObjectGUID        : d1b27942-8d0e-4ae5-b83b-d26af96170c5
SamAccountName    : krbtgt
SID               : S-1-5-21-3398938913-3940250523-927435294-502
Surname           :
UserPrincipalName :

DistinguishedName : CN=Helpdesk Hugó,OU=IT Admins,DC=r2,DC=dom
Enabled           : True
GivenName         : Hugó
Name              : Helpdesk Hugó
ObjectClass       : user
ObjectGUID        : 69896454-02e3-4c3c-bc6f-9598dd96b5c5
SamAccountName    : hh
SID               : S-1-5-21-3398938913-3940250523-927435294-1113
Surname           : Helpdesk
UserPrincipalName : hh@r2.dom
…
```

Látható, hogy meg kell adni kötelező paraméterként a `Filter`-nek valamilyen szűrőfeltételt, ami persze lehet a „*”, azaz az összes felhasználó is kiolvasható, de általában nem valószínű, hogy mindre szükségünk van. A kimenetben megkaptam a felhasználókat, de csak néhány tulajdonságukkal együtt. Ez megint csak - a kötelező szűrővel együtt – azt a célt szolgálja, hogy lehetőleg kíméljük mind a tartományvezérlőket, mind a hálózatot, hiszen ezekkel a korlátokkal kevesebb adatot kell küldenie számunkra. Ha több vagy más tulajdonságadatot szeretnénk, akkor használhatjuk a `Property` paramétert:

```
[39] PS C:\> Get-ADUser -Identity soostibor -Properties title


DistinguishedName : CN=Soós Tibor,OU=gyuri,DC=r2,DC=dom
Enabled           : True
GivenName         : Soós
Name              : Soós Tibor
ObjectClass       : user
ObjectGUID        : 60ee5a80-3305-43be-beb7-a71055d92225
SamAccountName    : SoosTibor
SID               : S-1-5-21-3398938913-3940250523-927435294-1178
Surname           : Tibor
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Title             : oktató
UserPrincipalName : SoosTibor@r2.dom
```

Itt láthatjuk, hogy megjelent a alapadatok mellett a `Title` tulajdonság is. Vagy nézzük meg egy felhasználó összes AD-ban tárolt tulajdonságát:

```
[40] PS C:\> Get-ADUser soostibor -Properties *


AccountExpirationDate              :
accountExpires                     : 9223372036854775807
AccountLockoutTime                 :
AccountNotDelegated                : False
AllowReversiblePasswordEncryption  : False
BadLogonCount                      : 0
badPasswordTime                    : 0
badPwdCount                        : 0
CannotChangePassword               : False
CanonicalName                      : r2.dom/gyuri/Soós Tibor
Certificates                       : {}
City                               : Management
CN                                 : Soós Tibor
codePage                           : 0
Company                            :
Country                            :
countryCode                        : 0
Created                            : 2010. 01. 07. 22:51:23
createTimeStamp                    : 2010. 01. 07. 22:51:23
…
```

A fenti példában már nem a `Filter`-rel szűrtem, hanem egy konkrét felhasználóra voltam kíváncsi, amit az `Identity` paraméternél adtam meg. De mire is hivatkoztam itt? Az `Identity` a help alapján a következőket jelentheti:

```
[41] PS C:\> get-help get-aduser -Parameter identity

-Identity <ADUser>
    Specifies an Active Directory user object by providing one of the followin
    g property values. The identifier in parentheses is the LDAP display name
    for the attribute.

      Distinguished Name
        Example:  CN=SaraDavis,CN=Europe,CN=Users,DC=corp,DC=contoso,DC=com
      GUID (objectGUID)
        Example: 599c3d2e-f72d-4d20-8a88-030d99495f20
      Security Identifier (objectSid)
        Example: S-1-5-21-3165297888-301567370-576410423-1103
      SAM account name  (sAMAccountName)
        Example: saradavis
…
```

Azaz az `Identity` több AD attribútum is lehet a *Distinguished Name*, *GUID*, *SID* és a *„Pre-Windows 2000 logon name*”. Sajnos a *User Principal Name* nem lehet, meg egyéb nevek sem (*Display Name*, *CN*), de azért ezek alapján is hivatkozhatunk a felhasználókra, csak ekkor filtert kell használni.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

2.13.1 Szűrés AD objektumokra

Nézzük akkor, hogy hogyan lehet más azonosító alapján szűrni a felhasználókra a `Filter` paraméterrel:

```
[46] PS C:\> Get-ADUser -Filter "userprincipalname -eq 'soostibor@r2.dom'" -Pro
perties title, department | Format-Table name, department, title

name                       department                title
----                       ----------                -----
Soós Tibor                 Management                oktató
```

Látható, hogy a filter egy PowerShell szintaxissal leírt szűrőfeltételt vár. Ezeket hívjuk OPATH szűrőnek. Valójában a háttérben a PowerShell az ezekben található tulajdonságneveket átfordítja LDAP nevekké, és a feltételeket és logikai kapcsolatokat meg átfordítja az LDAP szűrőkben használatos operátorokká. Nézzük meg, hogy milyen módon:

```
[47] PS C:\> get-help about_ActiveDirectory_Filter
TOPIC
    Active Directory Filter

SHORT DESCRIPTION
    Describes the syntax and behavior of the search filter supported by the
    Active Directory module for Windows PowerShell.
…
  Supported Operators
    The following table shows frequently used search filter operators.


                                                        LDAP
    Operator         Description                        Equivalent
    ---------------- ---------------------------------- ----------------------
    -eq              Equal to. This will                =
                     not support wild card
                     search.
    -ne              Not equal to. This will            !x = y
                     not support wild card
                     search.
    -approx          Approximately equal to             ~=
    -le              Lexicographically less than        <=
                     or equal to
    -lt              Lexicographically less than        !x >= y
    -ge              Lexicographically greater          >=
                     than or equal to
    -gt              Lexicographically greater than     !x <= y

    -and             AND                                &
    -or              OR                                 |
    -not             NOT                                !
    -bor             Bitwise OR                         :1.2.840.113556.1.4.804:=
    -band            Bitwise AND                        :1.2.840.113556.1.4.803:=
    -recursivematch  Use LDAP_MATCHING_RULE_IN_CHAIN    :1.2.840.113556.1.4.1941:
=
                     (Note: This control only works
                      with Windows 2008 and later.)
    -like            Similar to -eq and supports        =
                     wildcard comparison. The only
                     wildcard character
                     supported is: *
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    -notlike          Not like. Supports wild          !x = y
                      card comparison.

    Note: PowerShell wildcards, other than "*", such as "?" are not
    supported by the -Filter parameter syntax.
…
```

A fenti táblázatban tehát a PowerShell - LDAP operátorok szótárát láthatjuk.

Megjegyzés

Be kell valljam, hogy az `-approx` vagy `~=` operátor működéséhez semmi támpontot nem találtam. Sem az LDAP-ot definiáló RFC-ben, sem sehol másutt. Így én még nem mertem használni, mondjuk nem is igazán tudnék elképzelni olyan helyzetet, ahol ez nekem jól jönne.

Különösen a bitszintű ÉS, VAGY és a rekurzív keresés operátorainál spórolunk sokat. De természetesen LDAP formátumban is megadhatunk szűrő feltételeket:

```
[63] PS C:\> Get-ADUser -LDAPFilter '(userPrincipalName=soostibor@r2.dom)' -pro
perties title, department | Format-Table name, department, title

name                       department               title
----                       ----------               -----
Soós Tibor                 Management               oktató
```

Ugyanúgy megtaláltuk a felhasználót, mint a [46]-os sorban.

Megjegyzés

Fontos, hogy az LDAP filternél a zárójelpár az nem elhagyható, része az LDAP szintaxisnak. Ha elhagyjuk, akkor nem kapunk hibajelzést, hanem üres halmaz lesz a kimenet.

Nézzünk pár további szűrést. Időre szűrök, keresem a 2010. január 1. után létrehozott „Soós T”-szerű felhasználókat:

```
[69] PS C:\> $date = [datetime] "2010.01.01"
[70] PS C:\> Get-ADUser -Filter {created -gt $date -and name -like "Soós T*"} |
 Format-Table name

name
----
Soós Tamás
Soós Tibor
Soós Tímea
```

Nézzük meg ennek egy kicsit módosított változatát, az utóbbi 35 napban létrehozott felhasználók:

```
[71] PS C:\> Get-ADUser -Filter {created -gt $((get-date).adddays(-35)) -and na
me -like "Soós T*"} | Format-Table name
Get-ADUser : Variable: '' found in expression: $ is not defined.
At line:1 char:11
```

```
+ Get-ADUser <<<<  -Filter {created -gt $((get-date).adddays(-35)) -and name -
like "Soós T*"} | Format-Table name
    + CategoryInfo          : InvalidArgument: (:) [Get-ADUser], ArgumentExce
   ption
    + FullyQualifiedErrorId : Variable: '' found in expression: $ is not defi
   ned.,Microsoft.ActiveDirectory.Management.Commands.GetADUser
```

Ezt nem szerette.

Megjegyzés

A szűrőben a PowerShell nem bont ki összetett kifejezéseket, csak változókat. Így mindent számoljunk ki előre egy változóba és csak azt szerepeltessük a szűrőben.

Azaz az előző példa így már működik:

```
[72] PS C:\> $date = (get-date).adddays(-35); Get-ADUser -Filter {created -gt $
date -and name -like "Soós T*"} | Format-Table name

name
----
Soós Tamás
Soós Tibor
Soós Tímea
```

Természetesen hasonló módon lehet keresni például csoportokat is. A következő példában keresem a Groups szervezeti egységben létrehozott globális biztonsági csoportokat:

```
[1] PS C:\> Get-ADGroup -SearchBase "ou=groups,dc=r2,dc=dom" -filter {grouptype
 -band 0x80000002} -properties * | ft name, grouptype, samaccounttype, groupcat
egory, groupscope

name                 grouptype  samaccounttype   groupcategory     groupscope
----                 ---------  --------------   -------------     ----------
GS                 -2147483646       268435456        Security         Global
```

Ezt a szűrőt még az LDAP emlékeim alapján raktam össze, amikor is a `GroupType` attribútum 2. bitje jelentette a csoport Globális jellegét, és a 32. bitje jelentette azt, hogy ez egy biztonsági csoport. Ez a két bit egyidejűleg kell, hogy érvényre jusson, ezért használtam a `-band` operátort. Azért, hogy ne a teljes címtár adatbázisban keressek, ezért a keresés kiindulópontját a `-SearchBase` paraméterrel korlátoztam.

Ugyanez az ActiveDirectory modul segítségével sokkal egyszerűbben és érthetőbben is megfogalmazható:

```
[5] PS C:\> Get-ADGroup -SearchBase "ou=groups,dc=r2,dc=dom" -filter {groupcate
gory -eq "Security" -and groupscope -eq "Global"} -properties * | ft name, grou
ptype, samaccounttype, groupcategory, groupscope

name                 grouptype  samaccounttype   groupcategory     groupscope
----                 ---------  --------------   -------------     ----------
GS                 -2147483646       268435456        Security         Global
```

Nézzük, hogyan lehet csoporttagság alapján keresni. Keresem azokat a felhasználókat, akik benne vannak az Oktatók csoportban:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[10] PS C:\> Get-ADUser -Filter {memberof -eq "cn=g-oktatók,ou=IT Admins,dc=r2,
dc=dom"} | Format-Table name

name
----
Kovács Tímea
Bakai Viktor
Dolák Tamás
Ács Viktor
```

Látható, hogy a csoportra annak Distinguished Name nevével kell hivatkozni, ami kicsit megnehezíti a dolgot. Próbáljuk meg egyszerűsíteni:

```
[12] PS C:\> $g = Get-ADGroup g-oktatók
[13] PS C:\> Get-ADUser -Filter {memberof -eq $g} | Format-Table name

name
----
Kovács Tímea
Bakai Viktor
Dolák Tamás
Ács Viktor
```

Látható, hogy két lépésben egyszerűbben megoldható volt a feladat, elsőként a csoportot ragadtam meg és tettem a `$g` változóba, a második lépésben kerestem ki azokat a felhasználókat, akiknek a `memberOf` tulajdonsága megegyezik ezzel a csoporttal. Fontos, hogy nem kellett a csoport distinguished name tulajdonságára hivatkozni, ezt a háttérben megtette helyettünk a PowerShell.

Nézzük azt a problémát, amikor nem közvetlenül tagja egy felhasználó egy csoportnak, hanem egy másik csoport tagságán keresztül. Erre használhatjuk majd a `-recursivematch` operátort, de először nézzük meg, hogy kik a `dl-fullcontrol` nevű csoportunk tagjai a `Get-ADGroupMember` cmdlet segítségével:

```
[26] PS C:\> Get-ADGroupMember dl-fullcontrol | Format-Table name

name
----
G-Oktatók
```

Látható, hogy a `DL-FullControl` csoportnak csak a `G-Oktatók` csoport a tagja. De vajon ezen keresztül melyik felhasználók?

```
[27] PS C:\> $g = Get-ADGroup dl-fullcontrol
[28] PS C:\> Get-ADuser -Filter {memberof -recursivematch $g} | Format-Table na
me

name
----
Kovács Tímea
Bakai Viktor
Dolák Tamás
Ács Viktor
```

Látható, hogy ugyanaz a négy ember, akiket már korábban láttunk.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

A `-recursivematch` operátor csak a Windows Server 2008, vagy afölötti verziójú tartományvezérlőknél érhető el.

Mivel ilyen jellegű rekurzív keresés nagyon gyakori, ezt már a `Get-ADGroupMember` cmdlet magától is tudja:

```
[29] PS C:\> Get-ADGroupMember dl-fullcontrol -Recursive | Format-Table name

name
----
Ács Viktor
Dolák Tamás
Bakai Viktor
Kovács Tímea
```

Hogyan lehet arra rákeresni, hogy egy tulajdonság ki van-e töltve, van-e értéke? Erre leggyakrabban a `-like` operátor használható a „*” paraméterrel:

```
[40] PS C:\> Get-ADUser -Filter {othertelephone -like "*"} -Properties othertel
ephone | Format-Table name, othertelephone

name                                    othertelephone
----                                    --------------
Saly Judit                              {tel2, tel1}
```

Ebben a példában mindazon személyeket kerestem, akiknek ki van töltve az `otherTelephone` tulajdonságuk. Ez egy un. „multivalued property”, azaz olyan tulajdonság, ami önmagában is egy tömb. Látszik, hogy a `-like` operátort ez nem zavarta, a „hasonlatosságot” elemenként nézte.

A leggyakrabban használatos AD objektumok lekérdezésére, kezelésére van külön főnévvel ellátott cmdlet, de általános lekérdező főnevünk is van, az `ADObject`. Ilyen általános lekérdezést a `Get-ADObject`-tel végezhetünk. Például keressük az AD sémában azokat az attribútumokat, amelyek a Globális Katalógusba replikálódnak:

```
[64] PS C:\> Get-ADObject -Filter {objectClass -eq "AttributeSchema" -and ismem
berofpartialattributeset -eq $true} -SearchBase "cn=schema,cn=configuration,dc=
r2,dc=dom" -properties ismemberofpartialattributeset | ft name

name
----
Alt-Security-Identities
CA-Certificate
CA-Certificate-DN
Certificate-Templates
Common-Name
Country-Name
Description
…
```

Ez az általános lekérdező cmdlet ugyanúgy működik, mint az objektumspecifikus változatai, a szűrőnél érdemes az `objectClass` objektumtípust is meghatározni, hogy a találati lista mindenképpen megfelelően

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

szűrt legyen és a `-SearchBase` paraméterrel olyan partíciókra is irányíthatjuk a keresést, ami nem az alaphelyzet szerinti tartományi partíció.

2.13.2 Keresés egyéb paraméterei

Az AD-ben történő keresés jelentősen terhelheti a tartományvezérlőnket és a hálózatot is, így fontos, hogy kellően „óvatosan” álljunk ennek neki. Az óvatosságot az AD kereső cmdletek paraméterekkel is támogatják. Ilyen paraméterek a következők:

Paraméternév	Használat
-ResultPageSize	Hány objektumonként küldje át a találatokat, alapérték 256
-ResultSetSize	Maximális találatszám, alapérték `$null`, korlátlan találatszám
-SearchBase	A keresés kiindulópontja
-SearchScope	Milyen mélységű legyen a keresés. Lehetséges értékek: Base: csak az adott objektum lekérdezése OneLevel: egy szint mélységben Subtree: teljes mélységben, ez az alapérték

Például előre megbecsülöm, hogy a keresés eredményeként mondjuk kb. 2 felhasználót kellene kapnom, ezért lekorlátozom a lehetséges találatszámot mondjuk 5-re. Ha 5 lesz a találatszám, akkor valószínű elrontottam valamit, így tovább kell finomítani a keresést:

```
[55] PS C:\> Get-ADUser -Filter * -ResultSetSize 5 | Format-Table name

name
----
Administrator
Guest
krbtgt
Helpdesk Hugó
Vegetári János
```

Van még egy kereső cmdlet, a `Search-ADAccount`, amellyel speciális állapotú fiókokat – felhasználói-, számítógép-, szolgáltatásfiókokat – lehet kikeresni. Például a felfüggesztett fiókok listája:

```
[57] PS C:\> Search-ADAccount -ResultSetSize 5 -AccountDisabled | Format-Table
name

name
----
Guest
krbtgt
```

Vagy lejárt jelszavú fiókok:

```
[58] PS C:\> Search-ADAccount -ResultSetSize 5 -AccountExpired | Format-Table n
ame
```

További lehetőségek a `Search-ADAccount` cmdlettel:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Paraméter	Használat
-AccountExpiring [-DateTime <DateTime>] [-TimeSpan <TimeSpan>]	lejáró fiókok
-AccountInactive [-DateTime <DateTime>] [-TimeSpan <TimeSpan>]	nem használt fiókok
-LockedOut	kizárt fiókok
-PasswordExpired	lejárt jelszavú fiókok
-PasswordNeverExpires	nem lejáró jelszavú fiókok

A Globális Katalógusban is lehet keresni, ehhez a keresés `-Server` paraméterével kell megadni az egyik globális katalógust és a portszámot:

```
[95] PS C:\> Get-ADUser -Filter {title -like "szkripter"} -SearchBase "" -Serve
r dc:3268 | Format-Table name
[96] PS C:\> Get-ADUser -Filter {title -like "szkripter"} | Format-Table name

name
----
Szabó Katalin
Kovács Róbert
Laár Katalin
Szirtes Tamás
```

A [95]-ös sorban a `DC` nevű GC-t kérdeztem le az alaphelyzet szerinti `3268`-as porton. Mivel a teljes GC adatbázisban szeretnék keresni, ezért nem adtam `-SearchBase` paraméterként kiinduló pontot a kereséshez. Ennek a keresésnek nem lett eredménye, mert a `Title` tulajdonság nem replikálódik a GC-be. A [96]-os sorban ugyanez a keresés már adott eredményt, hiszen ebben az esetben már a tartományi partícióban kerestem.

2.13.3 Objektumok létrehozása

Egy-egy AD objektum létrehozása esetén nem biztos, hogy optimális a PowerShellt használni, hiszen nagyon sok adatot meg kell adni, sokat kell gépelni. A PowerShell akkor előnyös, ha tömegesen akarunk létrehozni objektumokat, akár fájlban tárolt információk alapján.

Nézzünk pár egyszerűbb példát, különböző objektumok létrehozására:

```
New-ADOrganizationalUnit -Path "DC=r2,DC=dom" `
    -Name "Managed Users" `
    -DisplayName "Managed Users" `
    -Description "Demó" `
    -ProtectedFromAccidentalDeletion:$false

New-ADGroup -SamAccountName "Csoport" `
    -GroupScope Global `
    -Name "Csoport" `
    -DisplayName "Csoport" `
    -Path "OU=Managed Users,DC=r2,DC=dom" `
    -Description "Demó"
```

Nézzünk arra példát, hogy fájlból hogyan lehet felhasználókat létrehozni. A fájl szerkezete a következő:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
vn, kn, rn, beo, pw, OU
Vegetári, János, vj, üzletkötő, Pa$$w0rd, "Managed Users"
Fájdalom, Csilla, fcs, titkárnő, Pa$$w0rd, "Managed Users"
Főnök, Ferenc, ff, igazgató, Pa$$w0rd, "Managed Users"
Beosztott, Béla, bb, melós, Pa$$w0rd, "Managed Users"
Rendszer, Géza, rg, rendszergazda, Pa$$w0rd, "IT Admins"
```

Azaz ez egy CSV fájl, azaz vesszőkkel elválasztott értékekből képzett táblázat. Ez alapján hozom létre a felhasználókat, és még be is teszem őket egy csoportba:

```
Import-Csv C:\munka\users.txt | foreach-object {
    New-ADUser -Name "$($_.vn) $($_.kn)" `
        -DisplayName "$($_.vn) $($_.kn)" `
        -AccountPassword  (ConvertTo-SecureString -String $_.pw `
            -AsPlainText -Force) `
        -Path "OU=$($_.ou),dc=r2,dc=dom" `
        -Title $_.beo `
        -GivenName $_.kn `
        -Surname $_.vn `
        -SamAccountName $_.rn `
        -UserPrincipalName "$($_.rn)@r2.dom" `
        -Enabled $true `
        -Description "Demó"

    Add-ADGroupMember -Identity Csoport -Members $_.rn
}
```

Itt már megtérül a szkript összeállításába fektetett idő, hiszen sok felhasználó esetén is ez ugyanúgy működik.

2.13.4 Objektumok módosítása

Az objektumok módosításánál már jobban kihasználható a PowerShell előnye, azaz a csövezés. Ugyanis általában a módosítás csővezetéke egy szűrési szakaszból és egy módosítási szakaszból áll össze. Nézzük meg a módosítás néhány specialitását:

```
[1] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} | Set-ADUser -Description
 "Módosítás"
[2] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} -prop description| Format
-Table name, description

name                                    description
----                                    -----------
Nagy Ildikó                             Módosítás
Nagy István                             Módosítás
```

Ez az egyik fajta módosítási lehetőség. A `Get-ADUser` résszel szűrök, majd ezt a csővezetéken átadom a `Set-ADUser` cmdletnek. Fontos, hogy a `Set-ADUser` önmagában képes a csővezetékből felszedni a paramétereit, nem kell `Foreach-Object` cmdletet használni. A fenti példában a `-Description` tulajdonságát módosítottam a felhasználóknak. Ez olyan gyakran használatos tulajdonság, hogy maga a `Set-ADUser` cmdletnek létrehoztak ilyen paramétert. De mi van akkor, ha olyan tulajdonságot akarunk

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

módosítani, amelyik nincsen kivezetve paraméterként? Elsőként nézzük meg, hogy mi az amit „gyárilag” tud a `Set-ADUser`:

```
[5] PS C:\> (get-help Set-ADUser).parameters.parameter | ft name

name
----
AccountExpirationDate
AccountNotDelegated
Add
AllowReversiblePasswordEncryption
AuthType
CannotChangePassword
Certificates
ChangePasswordAtLogon
City
Clear
Company
Country
Credential
Department
Description
DisplayName
Division
EmailAddress
EmployeeID
EmployeeNumber
Enabled
Fax
GivenName
HomeDirectory
HomeDrive
HomePage
HomePhone
Identity
Initials
Instance
LogonWorkstations
Manager
MobilePhone
Office
OfficePhone
Organization
OtherName
Partition
PassThru
PasswordNeverExpires
PasswordNotRequired
POBox
PostalCode
ProfilePath
Remove
Replace
SamAccountName
ScriptPath
Server
ServicePrincipalNames
SmartcardLogonRequired
State
StreetAddress
```

```
Surname
Title
TrustedForDelegation
UserPrincipalName
Confirm
WhatIf
```

Ezek közül néhány nem AD tulajdonság módosítására való paraméter (vastagon szedtem ezeket), közülük a következők alkalmasak általánosan tulajdonságok módosítására: `Add`, `Clear`, `Remove`, `Replace`.

```
[13] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} | Set-ADUser -replace @{
othertelephone = 1234; description = "másik módszer"}
[14] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} -prop description, other
telephone| Format-Table name, description, othertelephone

name                        description              othertelephone
----                        -----------              --------------
Nagy Ildikó                 másik módszer            {1234}
Nagy István                 másik módszer            {1234}
```

Látható, hogy értékadás célzattal a `-Replace` paraméternek egy hashtáblát adtam meg, melyben az AD attribútumnevek vannak kulcsként és az értékük értékként. Az `-Add` paraméter un. „multivalued” tulajdonságoknál használható újabb érték hozzáadásakor. A `Description` az nem „multivalued”, de az `OtherTelephone` az igen. A `-Remove`-val ilyen „multivalued” tulajdonságokból lehet értékeket elvenni:

```
[15] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} | Set-ADUser -add @{othe
rtelephone = "újszám"}
[16] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} -prop description, other
telephone| Format-Table name, description, othertelephone

name                        description              othertelephone
----                        -----------              --------------
Nagy Ildikó                 másik módszer            {újszám, 1234}
Nagy István                 másik módszer            {újszám, 1234}


[17] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} | Set-ADUser -remove @{o
thertelephone = "1234"}
[18] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} -prop description, other
telephone| Format-Table name, description, othertelephone

name                        description              othertelephone
----                        -----------              --------------
Nagy Ildikó                 másik módszer            {újszám}
Nagy István                 másik módszer            {újszám}
```

A [15]-ös sorban a kiszűrt felhasználóimnak adtam egy-egy újabb telefonszámot, a [17]-es sorban meg elvettem tőlük a régebbit.

A `-Clear` paraméterrel lehet AD tulajdonságokat törölni, `$null` érték adásával ne próbálkozzunk, hibát fog adni:

```
[19] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} | Set-ADUser -clear othe
rtelephone
[20] PS C:\> Get-ADUser -Filter {name -like "nagy i*"} -prop description, other
telephone| Format-Table name, description, othertelephone
```

```
name                        description             othertelephone
----                        -----------             --------------
Nagy Ildikó                 másik módszer           {}
Nagy István                 másik módszer           {}
```

Itt természetesen nem hashtáblát kell megadni a `-Clear` értékeként, hanem csak az attribútumneveket.

Harmadik módja az AD objektumok tulajdonságértékeinek megváltoztatásának a közvetlen értékadás:

```
[26] PS C:\> $u = Get-ADUser acsbela -Properties department
[27] PS C:\> $u | Format-Table name, department

name                                    department
----                                    ----------
Ács Béla                                Beszerzés


[28] PS C:\> $u.Department = "Sóhivatal"
[29] PS C:\> Set-ADUser -Instance $u
[30] PS C:\> Get-ADUser acsbela -Properties department | Format-Table name, dep
artment

name                                    department
----                                    ----------
Ács Béla                                Sóhivatal
```

A [26]-os sorban kiolvasom a felhasználót a „`department`” tulajdonságával együtt és berakom a `$u` változóba. A [27]-es sorban látszik, hogy ő a „Beszerzés” nevű osztályon dolgozik. A [28]-as sorban a `$u` változó `Department` tulajdonságának új értéket adok. Ez a művelet a memóriában, a `$u` változóban jön csak létre, magában az AD adatbázisban nem. Ahhoz, hogy ez beíródjon a címtárba is, ehhez szükséges volt a [29]-es sorban a művelet „kommittálása”, amit a `Set-ADUser`-rel tettem meg.

Figyelem! Itt a paraméter az `-Instance` volt, nem az `-Identity`! Bár az `-Identity`-vel is lefut a parancs, de ekkor tényleges végrehajtás nem történik.

Ezzel a harmadik módszerrel a multivalued tulajdonságokat legkényelmesebben az `Update-List` cmdlet segítségével tudjuk módosítani:

```
[39] PS C:\> $u = Get-ADUser acsbela -Properties othertelephone
[40] PS C:\> $u


DistinguishedName : CN=Ács Béla,OU=csaba,DC=r2,DC=dom
Enabled           : True
GivenName         : Ács
Name              : Ács Béla
ObjectClass       : user
ObjectGUID        : 56419e7c-211c-41eb-80fb-c5be5d82b217
SamAccountName    : acsBela
SID               : S-1-5-21-3398938913-3940250523-927435294-1607
Surname           : Béla
UserPrincipalName : acsBela@r2.dom



[41] PS C:\> $u.othertelephone = "egy"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Álljunk meg egy szóra! Látható a [40]-es sor kimenetében, hogy alaphelyzetben nincs is az `$u` változónak `othertelephone` tulajdonsága, mégis sikeres az értékadás a [41]-es sorban!

Folytassuk akkor az `Update-List` alkalmazásával:

```
[42] PS C:\> $u | Update-List -Add "másik" -Property othertelephone


DistinguishedName  : CN=Ács Béla,OU=csaba,DC=r2,DC=dom
Enabled            : True
GivenName          : Ács
Name               : Ács Béla
ObjectClass        : user
ObjectGUID         : 56419e7c-211c-41eb-80fb-c5be5d82b217
othertelephone     : {egy, másik}
PSShowComputerName : {}
SamAccountName     : acsBela
SID                : S-1-5-21-3398938913-3940250523-927435294-1607
Surname            : Béla
UserPrincipalName  : acsBela@r2.dom
WriteErrorStream   : {}



[43] PS C:\> Set-ADUser -Instance $u
[44] PS C:\> Get-ADUser $u -Properties othertelephone | ft name,othertelephone

name                                    othertelephone
----                                    --------------
Ács Béla                                {másik, egy}
```

Természetesen az `Update-List` összes lehetőségét ki tudjuk itt használni, az `Add`-on kívül a `Replace` vagy `Remove`-ot is.

2.13.5 Az AD meghajtó

Az ActiveDirectory modul betöltésével kapunk még egy meglepetést is:

```
[1] PS C:\> Get-PSDrive

WARNING: column "CurrentLocation" does not fit into the display and was remove
d.

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
A                                      FileSystem    A:\
AD                                     ActiveDire... //RootDSE/
Alias                                  Alias
…
```

Kapunk egy új meghajtót `AD:` néven. Ezen keresztül is elérhetjük a címtár adatbázist és az ottani objektumokat. Bár erre a `Get-ADObject` és hasonló cmdletekkel is alkalmasak, de az objektumok hierarchiája és a hozzáférési listája ezen meghajtón keresztül férhető hozzá igazán.

Nézzük, hogy mi is van ezen a meghajtón:

```
[2] PS C:\> dir ad:
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Name                ObjectClass          DistinguishedName
----                -----------          -----------------
r2                  domainDNS            DC=r2,DC=dom
Configuration       configuration        CN=Configuration,DC=r2,DC=dom
Schema              dMD                  CN=Schema,CN=Configuration,DC=r2,...
DomainDnsZones      domainDNS            DC=DomainDnsZones,DC=r2,DC=dom
ForestDnsZones      domainDNS            DC=ForestDnsZones,DC=r2,DC=dom
```

Látható, hogy a legfelsőbb szinten maguk a címtárpartíciók láthatók. Az `r2` tartományi partíciómon kívül a `Configuration`, `Schema` és a DNS AD integrált zónapartíciók. Az aktuális könyvtárnak beállítom az `AD:` meghajtót:

```
[3] PS C:\> cd ad:
```

Ahhoz, hogy még beljebb lépjek és a TAB billentyűvel ki lehessen egészíttetni az elérési utakat, a `DistinguishedName` értékét kell elkezdeni begépelni:

```
[4] PS AD:\> cd dc=r<tab>
```

És ebből lesz a TAB lenyomása után:

```
[4] PS AD:\> cd '.\DC=r2,DC=dom'
```

Ezt egy picit szokni kell, sajnos a sima `Name` gépelése nem vezet eredményhez. Nézzük itt mi van:

```
[5] PS AD:\DC=r2,DC=dom> dir

Name                ObjectClass          DistinguishedName
----                -----------          -----------------
Builtin             builtinDomain        CN=Builtin,DC=r2,DC=dom
Computers           container            CN=Computers,DC=r2,DC=dom
Domain Controllers  organizationalUnit   OU=Domain Controllers,DC=r2,DC=dom
ForeignSecurityPr... container           CN=ForeignSecurityPrincipals,DC=r...
Infrastructure      infrastructureUpdate CN=Infrastructure,DC=r2,DC=dom
IT Admins           organizationalUnit   OU=IT Admins,DC=r2,DC=dom
LostAndFound        lostAndFound         CN=LostAndFound,DC=r2,DC=dom
Managed Computers   organizationalUnit   OU=Managed Computers,DC=r2,DC=dom
Managed Service A... container           CN=Managed Service Accounts,DC=r2...
Managed Users       organizationalUnit   OU=Managed Users,DC=r2,DC=dom
NTDS Quotas         msDS-QuotaContainer  CN=NTDS Quotas,DC=r2,DC=dom
Program Data        container            CN=Program Data,DC=r2,DC=dom
System              container            CN=System,DC=r2,DC=dom
Users               container            CN=Users,DC=r2,DC=dom
```

Ezek bizony a tartományi partíció legfelsőbb szintű konténerei és szervezeti egységei. Nézzünk bele a „Managed Users” szervezeti egységbe:

```
[6] PS AD:\DC=r2,DC=dom> cd '.\OU=Managed Users'
[7] PS AD:\OU=Managed Users,DC=r2,DC=dom> dir

Name                ObjectClass          DistinguishedName
----                -----------          -----------------
Beosztott Béla      user                 CN=Beosztott Béla,OU=Managed User...
CsoportTörölveLesz  group                CN=CsoportTörölveLesz,OU=Managed ...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Fájdalom Csilla      user                CN=Fájdalom Csilla,OU=Managed Use...
Főnök Ferenc         user                CN=Főnök Ferenc,OU=Managed Users,...
Vegetári János       user                CN=Vegetári János,OU=Managed User...
```

És nézzük meg, hogy milyen adatokhoz jutunk hozzá egy felhasználóról:

```
[8] PS AD:\OU=Managed Users,DC=r2,DC=dom> Get-Item '.\CN=Beosztott Béla' | Form
at-List *


PSPath            : ActiveDirectory:://RootDSE/CN=Beosztott Béla,OU=Managed Us
                    ers,DC=r2,DC=dom
PSParentPath      : ActiveDirectory:://RootDSE/OU=Managed Users,DC=r2,DC=dom
PSChildName       : CN=Beosztott Béla
PSDrive           : AD
PSProvider        : ActiveDirectory
PSIsContainer     : True
distinguishedName : CN=Beosztott Béla,OU=Managed Users,DC=r2,DC=dom
name              : Beosztott Béla
objectClass       : user
objectGUID        : 20114f76-357d-4055-b607-d609101371e7
PropertyNames     : {distinguishedName, name, objectClass, objectGUID}
PropertyCount     : 4
```

Látható, hogy itt sem férünk hozzá az összes AD tulajdonsághoz alaphelyzetben, így ha azokra van szükségünk, akkor meg kell adni a `-properties` paramétert is:

```
[24] PS AD:\OU=Managed Users,DC=r2,DC=dom> Get-Item '.\CN=Beosztott Béla' -prop
erties *


PSPath                : ActiveDirectory:://RootDSE/CN=Beosztott Béla,OU=Manage
                        d Users,DC=r2,DC=dom
PSParentPath          : ActiveDirectory:://RootDSE/OU=Managed Users,DC=r2,DC=d
                        om
PSChildName           : CN=Beosztott Béla
PSDrive               : AD
PSProvider            : ActiveDirectory
PSIsContainer         : True
accountExpires        : 9223372036854775807
badPasswordTime       : 0
badPwdCount           : 0
cn                    : Beosztott Béla
codePage              : 0
countryCode           : 0
description           : Demó
displayName           : Beosztott Béla
distinguishedName     : CN=Beosztott Béla,OU=Managed Users,DC=r2,DC=dom
dSCorePropagationData : {2009. 12. 07. 22:20:21, 2009. 12. 07. 22:16:42, 2009.
                         12. 07. 22:11:52, 2009. 12. 07. 21:41:10...}
givenName             : Béla
instanceType          : 4
lastKnownParent       : OU=Managed Users,DC=r2,DC=dom
lastLogoff            : 0
lastLogon             : 0
logonCount            : 0
memberOf              : {CN=CsoportTörölveLesz,OU=Managed Users,DC=r2,DC=dom}
msDS-LastKnownRDN     : Beosztott Béla
name                  : Beosztott Béla
```

```
nTSecurityDescriptor  : System.DirectoryServices.ActiveDirectorySecurity
objectCategory        : CN=Person,CN=Schema,CN=Configuration,DC=r2,DC=dom
objectClass           : user
objectGUID            : 20114f76-357d-4055-b607-d609101371e7
objectSid             : S-1-5-21-3398938913-3940250523-927435294-1124
primaryGroupID        : 513
pwdLastSet            : 129046772259930547
sAMAccountName        : bb
sAMAccountType        : 805306368
sn                    : Beosztott
title                 : melós
userAccountControl    : 512
userPrincipalName     : bb@r2.dom
uSNChanged            : 17055
uSNCreated            : 16959
whenChanged           : 2009. 12. 07. 22:20:21
whenCreated           : 2009. 12. 07. 17:33:45
PropertyNames         : {accountExpires, badPasswordTime, badPwdCount, cn...}
PropertyCount         : 37
```

Így már megkaptam az összes kitöltött AD tulajdonságát a felhasználónak.

Megjegyzés

A `-properties` tulajdonság un. dinamikus paraméter, azaz az aktuális elérési út PSProviderétől függ a jelenléte. Ha a fájlrendszerben vagyunk, akkor ilyen paramétere nincs a `Get-Item` cmdletnek. A jelenlegi PowerShell verzióban van egy kis hiányosság, ez a dinamikus paraméter TAB kiegészítéssel nem hívható elő sajnos. A help azért tud róla:

```
[25] PS AD:\OU=Managed Users,DC=r2,DC=dom> get-help Get-Item -Parameter propert
ies

-Properties <string[]>
    Specifies a comma-delimited list of properties to be retrieved for each it
    em (Active Directory object).  The * wildcard can be used to retrieve all
    properties.

    Required?                    false
    Position?                    named
    Default value
    Accept pipeline input?       false
    Accept wildcard characters?  false
```

További dinamikus paramétereink is vannak az ActiveDirectory provider környezetben:
`-Server`, `-GlobalCatalog`, `-SizeLimit`, `-PageSize`, `-FormatType`.

Ha már `-Server` és `-GlobalCatalog`, akkor nézzük ugyanezt a felhasználót a Globális Katalóguson keresztül:

```
[32] PS AD:\OU=Managed Users,DC=r2,DC=dom> Get-Item '.\CN=Beosztott Béla' -prop
erties * -globalcatalog -server dc:3268


PSPath                : ActiveDirectory::://RootDSE/CN=Beosztott Béla,OU=Manage
                        d Users,DC=r2,DC=dom
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PSParentPath          : ActiveDirectory::://RootDSE/OU=Managed Users,DC=r2,DC=d
                        om
PSChildName           : CN=Beosztott Béla
PSDrive               : AD
PSProvider            : ActiveDirectory
PSIsContainer         : True
cn                    : Beosztott Béla
description           : Demó
displayName           : Beosztott Béla
distinguishedName     : CN=Beosztott Béla,OU=Managed Users,DC=r2,DC=dom
dSCorePropagationData : {2009. 12. 07. 22:20:21, 2009. 12. 07. 22:16:42, 2009.
                         12. 07. 22:11:52, 2009. 12. 07. 21:41:10...}
givenName             : Béla
instanceType          : 4
memberOf              : {CN=CsoportTörölveLesz,OU=Managed Users,DC=r2,DC=dom}
name                  : Beosztott Béla
nTSecurityDescriptor  : System.DirectoryServices.ActiveDirectorySecurity
objectCategory        : CN=Person,CN=Schema,CN=Configuration,DC=r2,DC=dom
objectClass           : user
objectGUID            : 20114f76-357d-4055-b607-d609101371e7
objectSid             : S-1-5-21-3398938913-3940250523-927435294-1124
primaryGroupID        : 513
sAMAccountName        : bb
sAMAccountType        : 805306368
sn                    : Beosztott
userAccountControl    : 512
userPrincipalName     : bb@r2.dom
uSNChanged            : 17055
uSNCreated            : 16959
whenChanged           : 2009. 12. 07. 22:20:21
whenCreated           : 2009. 12. 07. 17:33:45
PropertyNames         : {cn, description, displayName, distinguishedName...}
PropertyCount         : 25
```

Látható, hogy az előző próbálkozás 37 AD tulajdonságával szemben itt csak 25 tulajdonságot kaptunk.

Nézzük akkor, hogy a hozzáférési lista hogyan érhető el ezen az `AD:` meghajtón keresztül a `Get-Acl` cmdlet segítségével:

```
[18] PS AD:\DC=r2,DC=dom> Get-ChildItem | get-acl
Get-Acl : Cannot validate argument on parameter 'Path'. The argument is null,
empty, or an element of the argument collection contains a null value. Supply
a collection that does not contain any null values and then try the command ag
ain.
At line:1 char:24
+ Get-ChildItem | get-acl <<<<
    + CategoryInfo          : InvalidData: (OU=_TemplateOU,DC=r2,DC=dom:PSObj
   ect) [Get-Acl], ParameterBindingValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationError,Microsoft.Powe
   rShell.Commands.GetAclCommand
```

Sajnos egy újabb apró hiba van itt a PowerShellben, az AD objektumok `PSPath` tulajdonsága nem adódik át megfelelően a `Get-ACL`-nek, így a csővezetéken nem történik meg a paraméterátadás. Egy kis trükkel azonban megkerülhető a probléma:

```
[20] PS AD:\DC=r2,DC=dom> Get-ChildItem | foreach-object {get-acl -path $_.pspa
th}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Path                        Owner                      Access
----                        -----                      ------
ActiveDirectory::///Root... R2\Domain Admins           Everyone Deny  ...
ActiveDirectory::///Root... BUILTIN\Administrators     Everyone Allow  ...
ActiveDirectory::///Root... R2\Domain Admins           NT AUTHORITY\Authentic...
ActiveDirectory::///Root... R2\Domain Admins           Everyone Deny  ...
…
```

Egy konkrét objektumra:

```
[21] PS AD:\DC=r2,DC=dom> Get-Item '.\OU=Managed Users' | ForEach-Object {get-a
cl -Path $_.pspath} | Format-List


Path   : ActiveDirectory::///RootDSE/OU=Managed Users,DC=r2,DC=dom
Owner  : R2\Domain Admins
Group  : R2\Domain Admins
Access : NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS Allow
         NT AUTHORITY\Authenticated Users Allow
         NT AUTHORITY\SYSTEM Allow
         R2\Domain Admins Allow
         BUILTIN\Account Operators Allow
         BUILTIN\Account Operators Allow
         BUILTIN\Account Operators Allow
         BUILTIN\Account Operators Allow
         BUILTIN\Print Operators Allow
         R2\PSHelpDesk Allow
         R2\PSHelpDesk Allow
         R2\PSHelpDesk Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS Allow
         NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS Allow
         NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         NT AUTHORITY\SELF Allow
         R2\Enterprise Admins Allow
         BUILTIN\Pre-Windows 2000 Compatible Access Allow
         BUILTIN\Administrators Allow
Audit  :
Sddl   : O:DAG:DAD:AI(A;;LCRPLORC;;;ED)(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPDTLOC
         RSDRCWDWO;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;DA)(OA;;CCDC;bf967a86
         -0de6-11d0-a285-00aa003049e2;;AO)(OA;;CCDC;bf967a9c-0de6-11d0-a285-00
         aa003049e2;;AO)(OA;;CCDC;4828cc14-1437-45bc-9b07-ad6f015e5f28;;AO)(OA
         ;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;AO)(OA;;CCDC;bf967aa8-0d
         e6-11d0-a285-00aa003049e2;;PO)(OA;CIIO;WP;;bf967aba-0de6-11d0-a285-00
         aa003049e2;S-1-5-21-3398938913-3940250523-927435294-1110)(OA;CIIO;CR;
         00299570-246d-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa0030
         49e2;S-1-5-21-3398938913-3940250523-927435294-1110)(OA;CIIO;CR;ab721a
         53-1e2f-11d0-9819-00aa0040529b;bf967aba-0de6-11d0-a285-00aa003049e2;S
```

```
        -1-5-21-3398938913-3940250523-927435294-1110)(OA;CIIOID;RP;4c164200-2
        0c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(O
        A;CIIOID;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a
        285-00aa003049e2;RU)(OA;CIIOID;RP;5f202010-79a5-11d0-9020-00c04fc2d4c
        f;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;CIIOID;RP;5f202010-79a5
        -11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;C
        IIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07
        -ad6f015e5f28;RU)(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;b
        f967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;CIIOID;RP;59ba2f42-79a2-11
        d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;CIIO
        ID;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00
        aa003049e2;RU)(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828
        cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;CIIOID;RP;037088f8-0ae1-11d2-
        b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;CIIOID;
        RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a86-0de6-11d0-a285-00aa0
        03049e2;ED)(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a9
        c-0de6-11d0-a285-00aa003049e2;ED)(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854
        e-00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)(OA;CIIOID;LCR
        PLORC;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;CIIOID;LCRPLORC;;b
        f967a9c-0de6-11d0-a285-00aa003049e2;RU)(OA;CIIOID;LCRPLORC;;bf967aba-
        0de6-11d0-a285-00aa003049e2;RU)(OA;CIID;RPWPCR;91e647de-d96f-4b70-955
        7-d63ff4f3ccd8;;PS)(A;CIID;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;EA)(A;CIID;LC
        ;;;RU)(A;CIID;CCLCSWRPWPLOCRSDRCWDWO;;;BA)
```

Gyakorlatilag egy ugyanolyan kimenetet kaptunk, mint a fájlrendszer esetében, persze az AD-ben egy kicsit bonyolultabbak ezek a biztonsági leírók, hiszen itt minden objektumnak a sémában definiált jó néhány explicit engedélybejegyzése is van.

2.13.6 Információk az AD erdőről, tartományról, tartományvezérlőkről

A korábban alkalmazott cmdletek többségénél nem határoztam meg a konkrét tartományvezérlőt, amellyel dolgozzon, ezek az AD cmdletek maguk is felderítik a tartományvezérlőket. A Globális Katalógusnál már segítenem kellett, így jól jön, ha tudunk információkat a tartományvezérlőinkről. Erre is vannak cmdletek. Az első a `Get-ADForest`:

```
[44] PS C:\> Get-ADForest


ApplicationPartitions : {DC=DomainDnsZones,DC=r2,DC=dom, DC=ForestDnsZones,DC=
                        r2,DC=dom}
CrossForestReferences : {}
DomainNamingMaster    : dc.r2.dom
Domains               : {r2.dom}
ForestMode            : Windows2008R2Forest
GlobalCatalogs        : {dc.r2.dom}
Name                  : r2.dom
PartitionsContainer   : CN=Partitions,CN=Configuration,DC=r2,DC=dom
RootDomain            : r2.dom
SchemaMaster          : dc.r2.dom
Sites                 : {Default-First-Site-Name}
SPNSuffixes           : {}
UPNSuffixes           : {}
```

folyami felhasználása nem

Ezzel a legfontosabb tartományi információkat megkapjuk: a tartományok listája, Globális Katalógusok, erdő szintű egyedi szerepek helye, gyökértartomány, telephelyek és a tartomány működési szintje.

Az aktuális tartományról is szerezhetünk információkat a `Get-ADDomain` cmdlettel:

```
[46] PS C:\> Get-ADDomain


AllowedDNSSuffixes                : {}
ChildDomains                      : {}
ComputersContainer                : CN=Computers,DC=r2,DC=dom
DeletedObjectsContainer           : CN=Deleted Objects,DC=r2,DC=dom
DistinguishedName                 : DC=r2,DC=dom
DNSRoot                           : r2.dom
DomainControllersContainer        : OU=Domain Controllers,DC=r2,DC=dom
DomainMode                        : Windows2008R2Domain
DomainSID                         : S-1-5-21-3398938913-3940250523-927435294
ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=r2,DC=dom
Forest                            : r2.dom
InfrastructureMaster              : dc.r2.dom
LastLogonReplicationInterval      :
LinkedGroupPolicyObjects          : {CN={31B2F340-016D-11D2-945F-00C04FB984F9
                                    },CN=Policies,CN=System,DC=r2,DC=dom}
LostAndFoundContainer             : CN=LostAndFound,DC=r2,DC=dom
ManagedBy                         :
Name                              : r2
NetBIOSName                       : R2
ObjectClass                       : domainDNS
ObjectGUID                        : 816e1b9d-1c30-4aba-9958-58fb0db9cec2
ParentDomain                      :
PDCEmulator                       : dc.r2.dom
QuotasContainer                   : CN=NTDS Quotas,DC=r2,DC=dom
ReadOnlyReplicaDirectoryServers   : {}
ReplicaDirectoryServers           : {dc.r2.dom}
RIDMaster                         : dc.r2.dom
SubordinateReferences             : {DC=ForestDnsZones,DC=r2,DC=dom, DC=Domai
                                    nDnsZones,DC=r2,DC=dom, CN=Configuration,
                                    DC=r2,DC=dom}
SystemsContainer                  : CN=System,DC=r2,DC=dom
UsersContainer                    : CN=Users,DC=r2,DC=dom
```

Innen is sok hasznos információhoz juthatunk. Sajnos a PowerShell 2.0-s verziója még nem támogatja az AD telephelyek kezelését, de a megfelelő .NET keretrendszerbeli objektumokkal lehet kezelni.

Még egy cmdlet kívánkozik még említésre, az `Get-ADDomainController`, mellyel a tartományvezérlőket lehet még több szempont szerint felderíteni. Egy tartományvezérlőről nagyon sok adatot megad:

```
[55] PS C:\> Get-ADDomainController


ComputerObjectDN           : CN=DC,OU=Domain Controllers,DC=r2,DC=dom
DefaultPartition           : DC=r2,DC=dom
Domain                     : r2.dom
Enabled                    : True
Forest                     : r2.dom
HostName                   : dc.r2.dom
InvocationId               : d450b8fa-1b93-4435-a113-5f268c0f0644
IPv4Address                : 192.168.1.10
```

könyv tanfolyam

```
IPv6Address                    :
IsGlobalCatalog                : True
IsReadOnly                     : False
LdapPort                       : 389
Name                           : DC
NTDSSettingsObjectDN           : CN=NTDS Settings,CN=DC,CN=Servers,CN=Default-Firs
                                 t-Site-Name,CN=Sites,CN=Configuration,DC=r2,DC=do
                                 m
OperatingSystem                : Windows Server 2008 R2 Enterprise
OperatingSystemHotfix          :
OperatingSystemServicePack     :
OperatingSystemVersion         : 6.1 (7600)
OperationMasterRoles           : {SchemaMaster, DomainNamingMaster, PDCEmulator, R
                                 IDMaster...}
Partitions                     : {DC=ForestDnsZones,DC=r2,DC=dom, DC=DomainDnsZone
                                 s,DC=r2,DC=dom, CN=Schema,CN=Configuration,DC=r2,
                                 DC=dom, CN=Configuration,DC=r2,DC=dom...}
ServerObjectDN                 : CN=DC,CN=Servers,CN=Default-First-Site-Name,CN=Si
                                 tes,CN=Configuration,DC=r2,DC=dom
ServerObjectGuid               : e67b1926-9731-4645-baef-479dc7046c9a
Site                           : Default-First-Site-Name
SslPort                        : 636
```

Keresem például azokat a tartományvezérlőket, amelyek az én telephelyen levő Globális Katalógusok (a kimenetet nem mutatom, mert a demórendszeremben csak egy DC van, így a kimenet nem túl változatos):

```
[58] PS C:\> Get-ADDomainController -Filter {isglobalcatalog -eq $true -and sit
e -eq "Default-First-Site-Name"}
```

Ez a kifejezés akár több DC-t is visszaadhat találatként. Ha mindenképpen csak egy DC-t akarunk, akkor használjuk a `-discover` kapcsolót. Ilyenkor azonban nem használható a `-filter` paraméter, helyette a `-service` paraméter használandó. Ennek a lehetséges értékei:

-Service paraméter értéke	Jelentése
PrimaryDC	Primari Domain Controller emulátor
GlobalCatalog	Globális katalógus
KDC	Key Distribution Center
TimeService	Időszinkron forrása
ReliableTimeService	Mérvadó időkiszolgáló
ADWS	AD Web Service

Ezek alapján keressünk egy optimális GC-t az AD telephelyemen:

```
[73] PS C:\> Get-ADDomainController -Discover -Service globalcatalog -SiteName
Default-First-Site-Name
```

2.13.7 Egyéb műveletek AD objektumokkal

Természetesen eltávolítani is lehet AD objektumokat, erre a `Remove-ADUser`, `Remove-ADGroup`, stb. cmdletek állnak rendelkezésünkre.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Mozgatni is lehet a címtáron belül objektumokat, erre a `Move-ADUser`, `Move-ADComputer`, stb. cmdleteket hívhatjuk segítségül.

Felhasználói és számítógép fiókokat lehet felfüggeszteni (`Disable-ADAccount`) és újra engedélyezni lehet ezeket (`Enable-ADAccount`), valamint a zárolást lehet feloldani (`Unlock-ADAccount`).

Természetesen még van jó néhány különböző AD cmdlet, de ezek ismertetése túlmutat ennek a könyvnek a keretein.

2.13.8 A Management Gateway

Windows Server 2008 és Windows Server 2003 tartományvezérlők esetében nem használható közvetlenül az Active Directory modul, szükséges az ilyen tartományokban telepíteni a Management Gateway nevű komponenst, ami lehetővé teszi az AD cmdletek működését. Maga az AD modul sem telepíthető a Windows Server 2008 R2 és Windows 7-nél régebbi Windows verziókra, de például egy Windows 7-re telepített Remote Server Administration Tools készlet telepítésével, és a Windows Server 2003-as tartományvezérlőre telepített Management Gateway együttesen már komplett, PowerShell 2.0-val menedzselhető környezetet biztosít számunkra.

A Management Gateway telepítésének számos előfeltétele van. A könyv írásának időpontjában ezeket a komponenseket kell telepíteni:

Komponens	Magyarázat
376193_ENU_i386_zip.exe	hotfix
dotnetfx35setup.exe	.NET Framework 3.5
NDP35SP1-KB969166-x86.exe	.NET Framework 3.5 SP 1
Windows5.2-KB968934-x86.exe	Management Gateway for Windows Server 2003
WindowsServer2003-KB969429-x86-ENU.exe	patch

Hasonló, de más verziószámú komponenseket kell telepíteni Windows Server 2008 tartományvezérlőkre.

A könyv tanfolyami felhasználása nem engedélyezett!

2.14 Távoli futtatási környezet testre szabása

A távoli futtatásban van egy forrás és egy célgép. A parancsok a célgépen hajtódnak végre, az eredmény XML adatok formájában jutnak a forrásgépre. Az egésznek az infrastrukturális alapját a WinRM szolgáltatás biztosítja speciális webszolgáltatás keretében. Jött az ötlet, hogy ne feltétlenül szolgaian lehessen csak végrehajtani a távoli parancsokat, hanem egy szűrőmechanizmussal testre lehessen szabni ezt a távoli környezetet. Ezen testre szabás során egyrészt jogosultságokat lehet rendelni a távoli környezethez, amelyekkel meghatározhatjuk, hogy ki az aki csatlakozhat oda és ott mit tehet, másrészt szűkíthetjük a rendelkezésre álló parancsokat, sőt, akár a parancsok egyes paramétereit is elrejthetjük. Ennek gyakorlati megvalósulását láthatjuk az Exchange Server 2010 esetében, ott ez a technológia segít kialakítani az un. Role Based Access Control-t, azaz a szerepalapú hozzáférés-vezérlést. Azonban ilyesmit mi is ki tudunk alakítani a rendelkezésre álló PowerShell cmdletek segítségével.

Az én példám egy helpdesk csapat számára kialakított speciális távoli hozzáférési felület a tartományvezérlőhöz, amin keresztül a helpdesk szakemberek az Active Directory csak lekérdező parancsaihoz férnek hozzá, ezen kívül csak a munkájukhoz szükséges `Set-ADAccountPassword` és `Set-ADUser` parancsokhoz. Nem ez jelenti a tényleges biztonsági beállítást, hiszen az AD-ben jogok delegálásval kell azt beállítani, de a PowerShell lehetőségek szűkítése hatékonyabbá teszi a munkavégzést, hiszen még véletlenül sem tévedhetnek a munkatársak olyan cmdletek közelébe, amelyek futtatása során úgyis jogosultsági hibajelzéseket kapnának.

Elsőként létrehozom a tartományvezérlőn azt a szkriptet, amelyik majd kialakítja a lekorlátozott környezetet. Ehhez Windows Server 2008 R2 tartományvezérlő vagy Management Gateway kell, hiszen ott áll rendelkezésre az ActiveDirectory modul:

```
Import-Module activedirectory

$commands = (Get-Command -Module activedirectory |
     where-object {$_.verb -eq "get"} | %{$_.name} )
$commands += "Set-ADUser"
$commands += "Set-ADAccountPassword"
"Get-Command", "Get-FormatData", "Select-Object",
     "Get-Help", "Measure-Object" | ForEach-Object {$commands += $_ }

$ExecutionContext.SessionState.Applications.Clear()
$ExecutionContext.SessionState.Scripts.Clear()

Get-Command | ?{$Commands -notcontains $_.name} |
     ForEach-Object {$_.Visibility="Private"}

$ExecutionContext.SessionState.LanguageMode="RestrictedLanguage"
```

Első lépésként importálom tehát az ActiveDirectory modult, hiszen a helpdeskeseknek biztosítani akarom a címtárral kapcsolatos cmdletek egy részét. Ezután pont a számukra szükséges parancskészletet alakítom ki azzal, hogy kiválogatom a számukra elérhető cmdleteket a `$commands` változóba. Ezek a cmdletek az AD modul összes `Get` igéjű cmdletje, plusz az ő tevékenységükhöz szükséges két `Set` cmdlet, a `Set-ADUser` és a `Set-ADAccountPassword`. Valamint van néhány kötelezően engedélyezendő cmdlet, ezeket látjuk a következő sorban. Ezekkel túl sok kárt nem lehet okozni, így ennek semmilyen biztonsági kockázata sincsen, viszont enélkül nem működik a távoli futtatási környezet.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ezután az `$ExecutionContext` automatikus változó `SessionState` tulajdonságának jellemzőit állítom be. Egyrészt letiltom az elérhető alkalmazásokat. Ilyen alkalmazás pl. a WinRM szolgáltatás használata, azaz letiltom, hogy az ide, a tartományvezérlőhöz kapcsolódó személy még távolabbi gépre kezdeményezhessen távoli kapcsolatot. Hasonló módon a szkripteket is letiltom, azaz például a helyi profilok szkriptjeit.

Ezután az összes elérhető cmdletet, amelyek nincsenek rajta az engedélyezett cmdletek listáján, láthatatlanná teszem azáltal, hogy a `Visibility` tulajdonságukat `Private`-ra állítom.

A legvégén az egész környezetet ebben a formában lezárom, hogy nehogy ezeket a beállításokat vissza tudja csinálni a leleményes helpdeskes. Ezt a `LanguageMode` tulajdonság beállításával tettem meg.

Ezután szintén a tartományvezérlőn létrehozok egy új távoli kapcsolódási környezetet, egy un. `PSSessionConfiguration` objektumot. Előtte, ha szükséges, engedélyezzük a távoli futtatás lehetőségét:

```
Enable-PSRemoting # ha még nem volt engedélyezve

register-pssessionConfiguration -name HelpDesk -startupScript `
C:\lurdy\startup.ps1 -ShowSecurityDescriptorUI
```

Ez létrehoz egy HelpDesk nevű „játszóteret” távoli futtatás céljára, ahol tehát eleve be lesz importálva az ActiveDirectory modul, viszont egy csomó cmdlet nem lesz elérhető. Mindezt a korábban látott `startup.ps1` szkript alakítja ki. Ezt a játszóteret még a jogosultság megfelelő beállításával is védjük, hogy ne csatlakozhasson ide bárki, mindjárt látjuk, hogy hogyan.

Ezt futtatva ilyen kimenetet kapunk:

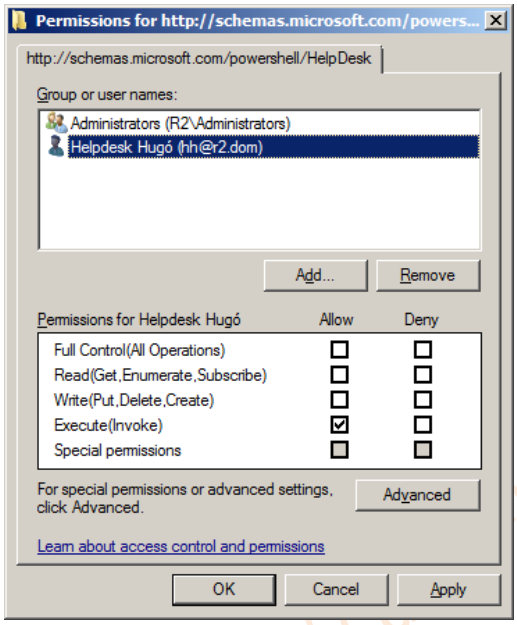
```
[62] PS C:\> register-pssessionConfiguration -name HelpDesk -startupScript `
C:\lurdy\startup.ps1 -ShowSecurityDescriptorUI

Confirm
Are you sure you want to perform this action?
Performing operation "Register-PSSessionConfiguration" on Target "Name:
HelpDesk. This will allow administrators to remotely run Windows PowerShell
commands on this computer".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin

Name                      Type                Keys
----                      ----                ----
HelpDesk                  Container           {Name=HelpDesk}

Confirm
Are you sure you want to perform this action?
Performing operation ""Restart-Service"" on Target "Name: WinRM".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
(default is "Y"):y

Confirm
Are you sure you want to perform this action?
Performing operation "Restart-Service" on Target "Windows Remote Management
(WS-Management) (winrm)".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
(default is "Y"):y
```

Néhány megerősítést kért tőlünk, és megjelent (lásd alább) a `-ShowSecurityDescriptorUI` kapcsoló hatására a következő grafikus dialógusablak:



72. ábra Grafikus - karakteres együttműködés

Ezen a felületen megadtam a Heldesk Hugó részére (a demó rendszeremen ő szimbolizálja a helpdesk csoportot) a szükséges `Execute (Invoke)` jogot. Itt ezt sokkal egyszerűbb beállítani, mint a karakteres konzolon, de ott is meg lehetett volna tenni.

Nézzük, hogy a távoli gépen hogyan lehet ehhez a környezethez csatlakozni:

```
PS C:\> $s = New-PSSession -ComputerName dc -ConfigurationName helpdesk
```

Ez nem túl bonyolult, de vajon hogyan tudja lekérdezni Hugó a felhasználókat az AD-ből?

```
PS C:\> Invoke-Command -Session $s -ScriptBlock {get-aduser -filter 'name -like
 "főn*"'}


PSComputerName     : dc
RunspaceId         : 81cb2f79-afa9-4e9a-8552-f8d296380932
PSShowComputerName : True
DistinguishedName  : CN=Főnök Ferenc,OU=Managed Users,DC=r2,DC=dom
Enabled            : True
GivenName          : Ferenc
Name               : Főnök Ferenc
ObjectClass        : user
ObjectGUID         : a067ce05-b902-4f26-812e-4bbb62280ea8
SamAccountName     : ff
SID                : S-1-5-21-3398938913-3940250523-927435294-1123
Surname            : Főnök
UserPrincipalName  : ff@r2.dom
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Itt már némi szomorúságot vélek felfedezni lelki szemeim előtt Hugó arcán. Elég macerás ez neki, főleg, hogy a „restricted” üzemmódú környezetben nem használhat scriptblock adattípust, így a `-Filter`-hez idézőjellel kell megadnia a feltételt. Szerencsére a PowerShell nagyon humánus, nem kárhoztatja a szegény helpdeskeseket állandó `Invoke-Command` parancsok használatára, mert be lehet importálni a távoli környezet lehetőségeit a lokális környezetbe az `Import-PSSession` cmdlet segítségével:

```
PS C:\> Import-PSSession -Session $s

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Script     tmp_e8a8b3de-e7fe-472a... {Get-ADUser, Get-ADServiceAccount, Get...
```

Ez a parancs „áthozza” a távoli környezet cmdleteit, valójában ugyanolyan nevű függvényeket hoz létre, mint a távoli cmdletek, és ebbe a függvénybe ágyazza be azt az `Invoke-Command` kifejezést, ami valójában a távoli parancsot végrehajtja. Nézzünk ezeket a függvényeket:

```
PS C:\> Get-Command -CommandType function | ?{$_.name -match "-AD"}

CommandType     Name                                Definition
-----------     ----                                ----------
Function        Get-ADAccountAuthorizationGroup     ...
Function        Get-ADAccountResultantPasswo...     ...
Function        Get-ADComputer                      ...
Function        Get-ADComputerServiceAccount        ...
Function        Get-ADDefaultDomainPasswordP...     ...
Function        Get-ADDomain                        ...
Function        Get-ADDomainController              ...
Function        Get-ADDomainControllerPasswo...     ...
Function        Get-ADDomainControllerPasswo...     ...
Function        Get-ADFineGrainedPasswordPolicy     ...
Function        Get-ADFineGrainedPasswordPol...     ...
Function        Get-ADForest                        ...
Function        Get-ADGroup                         ...
Function        Get-ADGroupMember                   ...
Function        Get-ADObject                        ...
Function        Get-ADOptionalFeature               ...
Function        Get-ADOrganizationalUnit            ...
Function        Get-ADPrincipalGroupMembership      ...
Function        Get-ADRootDSE                       ...
Function        Get-ADServiceAccount                ...
Function        Get-ADUser                          ...
Function        Get-ADUserResultantPasswordP...     ...
Function        Set-ADAccountPassword               ...
Function        Set-ADUser                          ...
```

Láthatjuk, hogy az elérhető AD cmdletek tényleg csak a `Get` igéjűeket tartalmazzák, plusz a `Set-ADAccountPassword` és a `Set-ADUser`. Pillantsunk bele egy függvénydefinícióba is (kicsit megvágtam a paraméterdefiníciós rész):

```
PS C:\> ${function:get-aduser}

    param(

    [Alias('db')]
    [Switch]
    ${Debug},
```

```
    [Alias('wv')]
    ${WarningVariable},

    [Alias('Property')]
    ${Properties},

    ${SearchBase},

    ${ResultPageSize},

    ${Credential},
…
begin {

        try {
            $positionalArguments = & $script:NewObject collections.arraylist
            foreach ($parameterName in $PSBoundParameters.BoundPositionally)
            {
                $null = $positionalArguments.Add( $PSBoundParameters[$paramete
rName] )
                $null = $PSBoundParameters.Remove($parameterName)
            }
            $positionalArguments.AddRange($args)

            $clientSideParameters = Get-PSImplicitRemotingClientSideParameters
 $PSBoundParameters $True

            $scriptCmd = { & $script:InvokeCommand `
                           @clientSideParameters `
                           -HideComputerName `
                           -Session (Get-PSImplicitRemotingSession -CommandNa
me 'Get-ADUser') `
                           -Arg ('Get-ADUser', $PSBoundParameters, $positiona
lArguments) `
                           -Script { param($name, $boundParams, $unboundParam
s) & $name @boundParams @unboundParams } `
                       }

            $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation
.CommandOrigin)
            $steppablePipeline.Begin($myInvocation.ExpectingInput, $ExecutionC
ontext)
…
}
```

Ez tulajdonképpen szerkezetileg egy olyan függvény, mint amit a *2.4.5 Meglevő cmdletek kiegészítése, átalakítása* fejezetben láttunk. Az elején ugyanolyan paramétereket definiál a függvény, mint ami az eredeti cmdletnek van, majd a paraméterátadáshoz szétválogatja a pozícionális paramétereket és a nevesítetteket. Nem mondom, hogy minden világos pontosan számomra, hogy mi is történik, a lényeg, hogy meg lesz hívva távoli módon az eredeti cmdlet.

Ráadásul így a TAB kiegészítés is működik, ezzel még kényelmesebb lesz a helpdeskes munkája. Nézzük, hogyan tudja a korábbi parancsot futtatni:

```
PS C:\> Get-ADUser -Filter {name -like "főn*"}
```

```
RunspaceId        : 81cb2f79-afa9-4e9a-8552-f8d296380932
DistinguishedName : CN=Főnök Ferenc,OU=Managed Users,DC=r2,DC=dom
Enabled           : True
GivenName         : Ferenc
Name              : Főnök Ferenc
ObjectClass       : user
ObjectGUID        : a067ce05-b902-4f26-812e-4bbb62280ea8
SamAccountName    : ff
SID               : S-1-5-21-3398938913-3940250523-927435294-1123
Surname           : Főnök
UserPrincipalName : ff@r2.dom
```

Láthatjuk, hogy minden ugyanúgy működik, mintha ott ülne az illető a tartományvezérlőre bejelentkezve! Mindezt úgy, hogy nem kellett telepíteni semmit sem a helpdeskes munkaállomására.

A távoli futtatási környezetek („játszóterek”) számos paraméterét be lehet állítani annak érdekében, hogy például túl sok, vagy néhány nagyon túlbuzgó helpdeskes nehogy túl nagy terhelést rakjon a tartományvezérlőnkre. Ezen lehetőségek feltérképezésére nézzünk bele az eddig még nem igazán feltérképezett `WSMAN:` meghajtónkra:

```
[1] PS C:\> cd wsman:
[2] PS WSMan:\> dir


   WSManConfig:

ComputerName                                Type
------------                                ----
localhost                                   Container
```

Itt nem sok minden van, menjünk még mélyebbre:

```
[3] PS WSMan:\> cd .\localhost
[4] PS WSMan:\localhost> dir | ft -AutoSize


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Name                Value      Type
----                -----      ----
MaxEnvelopeSizekb   150        System.String
MaxTimeoutms        60000      System.String
MaxBatchItems       32000      System.String
MaxProviderRequests 4294967295 System.String
Client                         Container
Service                        Container
Shell                          Container
Listener                       Container
Plugin                         Container
ClientCertificate              Container
```

Itt már van néhány paraméterünk, de ezek un. „globális” paraméterei a WinRM szolgáltatásnak. Láthatók ebben a listában „könyvtárszerű” elemek, amelyekbe be lehet lépni, és ott újabb elemek találhatók majd. Nézzük ezeknek a fő könyvtáraknak a szerepét:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Könyvtár neve	Szerepe
Client	A távoli futtatás kliensoldali jellemzői. Ez a tartományvezérlőnkön ebben a helyzetben nem érdekes, hiszen az a mi szempontunkból most nem kliens.
Service	A WinRM szolgáltatás globális paraméterei, a „szerver” oldal.
Shell	Ez a távoli futtatási környezet, és annak jellemzői.
Listener	A kiszolgáló „figyelő” komponensének beállításai, pl. milyen porton és melyik IP címére érkező kéréseket fogadja.
Plugin	Itt vannak a „játszóterek”, azaz a különböző távoli futtatási környezetek. Vannak itt már alaphelyzetben létrehozottak, de ide jött létre az én „helpdesk” környezetem is.
ClientCertificate	Ha tanúsítványalapú hitelesítés van megkövetelve, akkor itt lehet tárolni a kliens által felmutatott tanúsítványokat.

Nézzünk bele elsőként a `Plugin` könyvtárba:

```
[36] PS WSMan:\localhost\Plugin> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin

Name                      Type                Keys
----                      ----                ----
Event Forwarding Plugin   Container           {Name=Event Forwarding Plugin}
HelpDesk                  Container           {Name=HelpDesk}
microsoft.powershell      Container           {Name=microsoft.powershell}
Microsoft.PowerShell32    Container           {Name=Microsoft.PowerShell32}
microsoft.ServerManager   Container           {Name=microsoft.ServerManager}
SEL Plugin                Container           {Name=SEL Plugin}
WMI Provider              Container           {Name=WMI Provider}
```

Az általam létrehozott HelpDesk környezet mellett megtalálhatók a Windows által egyéb célokra használt WinRM futtatási környezetei, valamint az alaphelyzet szerinti PowerShell távoli futtatásoknak otthont adó `microsoft.powershell` és `Microsoft.PowerShell32` futtatási környezetek. Ezeket is testre lehet szabni, de inkább menjünk bele a `HelpDesk`-be:

```
[42] PS WSMan:\localhost\Plugin> cd .\HelpDesk
[43] PS WSMan:\localhost\Plugin\HelpDesk> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin\HelpDesk

WARNING: column "Type" does not fit into the display and was removed.

Name                      Value
----                      -----
xmlns                     http://schemas.microsoft.com/wbem/wsman/1/config/...
Name                      HelpDesk
Filename                  %windir%\system32\pwrshplugin.dll
SDKVersion                1
XmlRenderingType          text
lang                      en-US
InitializationParameters
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Resources
```

Látható, hogy ezen a szinten is van néhány beállítás, de nézzük a két alkönyvtárt:

```
[44] PS WSMan:\localhost\Plugin\HelpDesk> cd .\InitializationParameters
[45] PS WSMan:\localhost\Plugin\HelpDesk\InitializationParameters> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin\HelpDesk\In
itializationParameters

ParamName            ParamValue
---------            ----------
PSVersion            2.0
startupscript         C:\lurdy\startup...
```

Itt látható a környezet létrehozásánál megadott szkript, nézzük a `Resources` mappát, illetve egy kicsit mélyebbre is megyek:

```
[46] PS WSMan:\localhost\Plugin\HelpDesk\InitializationParameters> cd ..\resour
ces
[47] PS WSMan:\localhost\Plugin\HelpDesk\Resources> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin\HelpDesk\Re
sources

Name                      Type                Keys
----                      ----                ----
Resource_201271944        Container           {Uri=http://schemas.microsof...


[48] PS WSMan:\localhost\Plugin\HelpDesk\Resources> cd .\Resource_201271944
[49] PS WSMan:\localhost\Plugin\HelpDesk\Resources\Resource_201271944> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Plugin\HelpDesk\Re
sources\Resource_201271944

WARNING: column "Type" does not fit into the display and was removed.

Name                      Value
----                      -----
ResourceUri               http://schemas.microsoft.com/powershell/HelpDesk
SupportsOptions           true
ExactMatch                true
Capability                {Shell}
Security
```

Látható itt, majdnem legalul, hogy a `Capability` elemnél hivatkozik a `Shell`-re, mint alkalmazásra, azaz itt a háttérben egy konzol fut. Ezt a `Shell` elemet korábban már láttunk a `localhost` szint alatt. Menjünk oda most vissza:

```
[54] PS WSMan:\localhost> cd .\Shell
[55] PS WSMan:\localhost\Shell> dir
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Shell

WARNING: column "Type" does not fit into the display and was removed.

Name                        Value
----                        -----
AllowRemoteShellAccess      true
IdleTimeout                 180000
MaxConcurrentUsers          5
MaxShellRunTime             2147483647
MaxProcessesPerShell        15
MaxMemoryPerShellMB         150
MaxShellsPerUser            5
```

Na, itt már nagyon konkrét dolgok vannak a távoli futtatási környezetre vonatkozólag, például az, hogy engedélyezett-e ez egyáltalán. Vagy ha a helpdeskes nem csinál semmit, akkor mennyi idő múlva törölje a neki dinamikusan kialakított „játszóteret”. Ez itt 18000 másodperc, azaz 5 óra. Láthatjuk azt is, hogy egy időben 5 távoli csatlakozást enged és környezetenként 150 MB memória felhasználását engedélyezi és felhasználónként maximum 5 környezetet lehet nyitni.

Természetesen átírhatjuk ezeket a paramétereket:

```
[69] PS WSMan:\localhost\Shell> Set-Item .\AllowRemoteShellAccess -Value $false
```

Azaz ideiglenesen megtiltottam a távoli futtatás lehetőségét. Ezután a helpdeskes ezt tapasztalja:

```
PS C:\> Get-ADUser -Filter {name -like "főn*"}
Starting a command on remote server failed with the following error message :
The WS-Management service cannot process the request. The service is configure
d to not accept any remote shell requests. For more information, see the about
_Remote_Troubleshooting Help topic.
    + CategoryInfo          : OperationStopped: (System.Manageme...pressionSy
   ncJob:PSInvokeExpressionSyncJob) [], PSRemotingTransportException
    + FullyQualifiedErrorId : JobFailure
```

Azaz nem tud távoli parancsokat futtatni.

Ezekkel a lehetőségekkel tehát olyan speciális futtatási környezeteket tudunk kialakítani, amelyeket a célközönség számára testre tudunk szabni, így pont azokat a szolgáltatásokat tudják elérni, amelyek számukra a munkájuk végzéséhez szükségesek.

A könyv tanfolyami felhasználása nem engedélyezett!

2.15 .NET Framework hasznos osztályai

Az eddigiekben is láthattunk már jó néhány olyan problémát, amelyben a .NET keretrendszer osztályai nyújtottak segítséget, mert a PowerShell saját cmdletjei, típusai nem voltak elegendőek. Ebben a fejezetben néhány további fontos .NET osztályból szemezgetek, ezzel kívánok mindenkit arra bíztatni, hogy nyugodtan böngészgessen a .NET osztályok között a Reflector program vagy az MSDN weboldal segítségével, hátha olyanra bukkan, amely kész megoldást nyújt valamely feladatra.

2.15.1 Környezet ([environment])

Nézzük az első olyan egyszerű .NET osztályt, amelynek statikus metódusai és tulajdonságai hasznunkra válhatnak. Ez a `System.Environment`, vagy röviden `[Environment]`. Nézzük, miről is van szó:

```
[13] PS C:\> [environment] | get-member -Static


   TypeName: System.Environment

Name                        MemberType Definition
----                        ---------- ----------
Equals                      Method     static bool Equals(System.Object objA...
Exit                        Method     static System.Void Exit(int exitCode)
ExpandEnvironmentVariables Method     static string ExpandEnvironmentVariab...
FailFast                    Method     static System.Void FailFast(string me...
GetCommandLineArgs          Method     static string[] GetCommandLineArgs()
GetEnvironmentVariable      Method     static string GetEnvironmentVariable(...
GetEnvironmentVariables     Method     static System.Collections.IDictionary...
GetFolderPath               Method     static string GetFolderPath(System.En...
GetLogicalDrives            Method     static string[] GetLogicalDrives()
ReferenceEquals             Method     static bool ReferenceEquals(System.Ob...
SetEnvironmentVariable      Method     static System.Void SetEnvironmentVari...
CommandLine                 Property   static System.String CommandLine {get;}
CurrentDirectory            Property   static System.String CurrentDirectory...
ExitCode                    Property   static System.Int32 ExitCode {get;set;}
HasShutdownStarted          Property   static System.Boolean HasShutdownStar...
MachineName                 Property   static System.String MachineName {get;}
NewLine                     Property   static System.String NewLine {get;}
OSVersion                   Property   static System.OperatingSystem OSVersi...
ProcessorCount              Property   static System.Int32 ProcessorCount {g...
StackTrace                  Property   static System.String StackTrace {get;}
SystemDirectory             Property   static System.String SystemDirectory ...
TickCount                   Property   static System.Int32 TickCount {get;}
UserDomainName              Property   static System.String UserDomainName {...
UserInteractive             Property   static System.Boolean UserInteractive...
UserName                    Property   static System.String UserName {get;}
Version                     Property   static System.Version Version {get;}
WorkingSet                  Property   static System.Int64 WorkingSet {get;}
```

Osztályok statikus tulajdonságértékeinek felderítéséhez készítettem egy függvényt:

```
function get-staticprops ([type] $type)
{
    $h = @{}
    $type | Get-Member -Static -MemberType property |
        ForEach-Object {$h.($_.name) = $type::($_.name)}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    $h
}
```

Ezzel megvizsgálva az [environment] osztály tulajdonságait, a következőket kaphatjuk:

```
[17] PS C:\> get-staticprops environment

Name                            Value
----                            -----
NewLine                         ...
CommandLine                     "C:\WINDOWS\system32\WindowsPowerShell\v1.0\...
UserName                        Administrator
ExitCode                        0
TickCount                       173265703
ProcessorCount                  1
StackTrace                         at System.Environment.get_StackTrace()...
MachineName                     DC
OSVersion                       Microsoft Windows NT 6.1.7600.0
UserDomainName                  R2
CurrentDirectory                C:\Users\Administrator
UserInteractive                 True
Version                         2.0.50727.4927
HasShutdownStarted              False
SystemDirectory                 C:\Windows\system32
WorkingSet                      49762304
```

Ezek közül szkriptjeinkben haszonnal használhatunk nem egy tulajdonságot.

Nézzünk pár metódust is! Ugyan a környezeti változókhoz hozzáférünk közvetlenül PowerShellből is, de nézzük csak a $env:path változót:

```
[25] PS C:\> $env:path
%SystemRoot%\system32\WindowsPowerShell\v1.0\;C:\Windows\system32;C:\Windows;C
:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
```

Windows Server 2008 R2 esetében ez közvetlenül nem használható, hiszen az elején ott van a %SystemRoot% környezeti változó, de még a DOS-os szintaxissal, ami a PowerShell számára nem értelmezhető. Ahhoz, hogy ez behelyettesíthető legyen a PowerShell segítségével, egy viszonylag bonyolult regex kifejezést kellene használni, de ennél sokkal egyszerűbb a ExpandEnvironmentVariables metódus használata:

```
[26] PS C:\> [environment]::ExpandEnvironmentVariables($env:path)
C:\Windows\system32\WindowsPowerShell\v1.0\;C:\Windows\system32;C:\Windows;C:\
Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
```

Ezzel már tökéletes az eredmény!

Még egyértelműbb a speciális mappák tényleges elérési útját megmutató metódus haszna:

```
[34] PS C:\> [environment]::GetFolderPath("MyPictures")
C:\Users\Administrator\Pictures
[35] PS C:\> [environment]::GetFolderPath("ProgramFiles")
C:\Program Files
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Hiszen ezek az elérési utak eltérnek különböző Windows verziók esetében, ezzel viszont verzió-függetlenül jól működő szkripteket tudunk írni.

2.15.2 Konzol ([console])

A `$host` változó szolgáltatásaihoz hasonló dolgokat érünk el a `[console]` osztály (`System.Console`) statikus metódusain keresztül is. Nézzük a tagjellemzőit:

```
[12] PS C:\> [console] | Get-Member -Static


   TypeName: System.Console

Name                 MemberType Definition
----                 ---------- ----------
CancelKeyPress       Event      System.ConsoleCancelEventHandler CancelKeyP...
Beep                 Method     static System.Void Beep(), static System.Vo...
Clear                Method     static System.Void Clear()
Equals               Method     static bool Equals(System.Object objA, Syst...
MoveBufferArea       Method     static System.Void MoveBufferArea(int sourc...
OpenStandardError    Method     static System.IO.Stream OpenStandardError()...
OpenStandardInput    Method     static System.IO.Stream OpenStandardInput()...
OpenStandardOutput   Method     static System.IO.Stream OpenStandardOutput(...
Read                 Method     static int Read()
ReadKey              Method     static System.ConsoleKeyInfo ReadKey(), sta...
ReadLine             Method     static string ReadLine()
ReferenceEquals      Method     static bool ReferenceEquals(System.Object o...
ResetColor           Method     static System.Void ResetColor()
SetBufferSize        Method     static System.Void SetBufferSize(int width,...
SetCursorPosition    Method     static System.Void SetCursorPosition(int le...
SetError             Method     static System.Void SetError(System.IO.TextW...
SetIn                Method     static System.Void SetIn(System.IO.TextRead...
SetOut               Method     static System.Void SetOut(System.IO.TextWri...
SetWindowPosition    Method     static System.Void SetWindowPosition(int le...
SetWindowSize        Method     static System.Void SetWindowSize(int width,...
Write                Method     static System.Void Write(string format, Sys...
WriteLine            Method     static System.Void WriteLine(), static Syst...
BackgroundColor      Property   static System.ConsoleColor BackgroundColor ...
BufferHeight         Property   static System.Int32 BufferHeight {get;set;}
BufferWidth          Property   static System.Int32 BufferWidth {get;set;}
CapsLock             Property   static System.Boolean CapsLock {get;}
CursorLeft           Property   static System.Int32 CursorLeft {get;set;}
CursorSize           Property   static System.Int32 CursorSize {get;set;}
CursorTop            Property   static System.Int32 CursorTop {get;set;}
CursorVisible        Property   static System.Boolean CursorVisible {get;set;}
Error                Property   static System.IO.TextWriter Error {get;}
ForegroundColor      Property   static System.ConsoleColor ForegroundColor ...
In                   Property   static System.IO.TextReader In {get;}
InputEncoding        Property   static System.Text.Encoding InputEncoding {...
KeyAvailable         Property   static System.Boolean KeyAvailable {get;}
LargestWindowHeight  Property   static System.Int32 LargestWindowHeight {get;}
LargestWindowWidth   Property   static System.Int32 LargestWindowWidth {get;}
NumberLock           Property   static System.Boolean NumberLock {get;}
Out                  Property   static System.IO.TextWriter Out {get;}
OutputEncoding       Property   static System.Text.Encoding OutputEncoding ...
Title                Property   static System.String Title {get;set;}
TreatControlCAsInput Property   static System.Boolean TreatControlCAsInput ...
WindowHeight         Property   static System.Int32 WindowHeight {get;set;}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
WindowLeft              Property   static System.Int32 WindowLeft {get;set;}
WindowTop               Property   static System.Int32 WindowTop {get;set;}
WindowWidth             Property   static System.Int32 WindowWidth {get;set;}
```

Ezzel készíthetünk a régi ZX Spectrum és Commodore világába visszavezető, képernyő-pozícióba kiíró `PrintPos` függvényt:

```
function printpos ($char,$x,$y,
     [string] $bgc = [console]::BackgroundColor,
     [string] $fgc = [Console]::ForegroundColor)
{
     if($x -ge 0 -and $y -ge 0 -and $x -le [Console]::WindowWidth -and
           $y -le [Console]::WindowHeight)
     {
           $saveY = [console]::CursorTop
           $offY = [console]::WindowTop

         [console]::setcursorposition($x,$offY+$y)
         Write-Host -Object $char -BackgroundColor $bgc `
                 -ForegroundColor $fgc -NoNewline
         [console]::setcursorposition(0,$saveY)
     }
}
```

A paraméterei: a kiírandó karakter (lehet szöveg is), a koordináta x és y része, valamint a háttérszín és a betű színe. Itt a fő okosság, hogy a konzol ablak a koordinátákat nem a látható részhez, hanem a teljes képernyő pufferhez számolja, így ha én a látható képernyő megfelelő pozíciójába akarok írni, akkor a „eltolást” figyelembe kell venni, amit a `[console]::WinodwsTop` statikus tulajdonság ad meg.

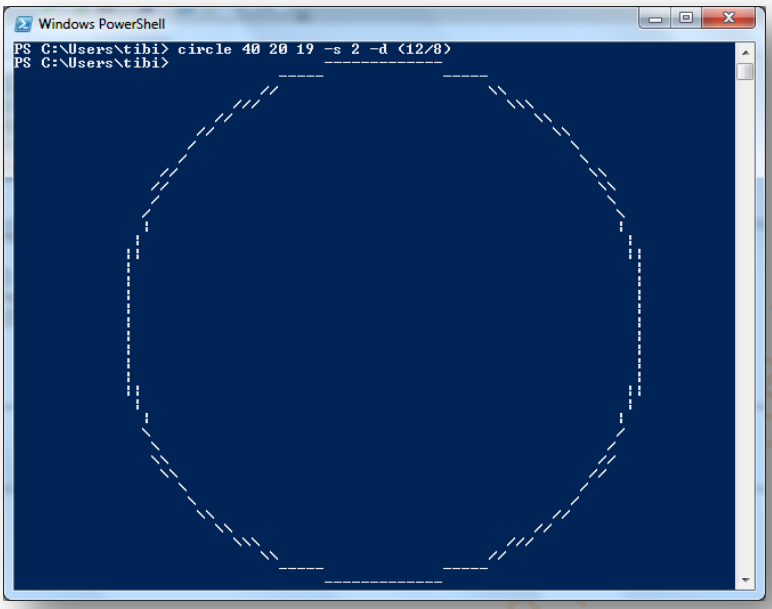
És ezzel akár körrajzoló függvényt is tudunk készíteni:

```
function circle ($x, $y, $r, $step = 5, $deform = 1)
{
     for($i =0 ; $i -lt 360; $i+=$step)
     {
           $oy = [int] ($y + $r*[math]::sin($i/180*[math]::pi))
           $ox = [int] ($x + $r*[math]::cos($i/180*[math]::pi)*$deform)
           switch($i)
           {
                 {$i -gt 337} {$c = "|"; break}
                 {$i -gt 292} {$c = "\"; break}
                 {$i -gt 247} {$c = "-"; break}
                 {$i -gt 202} {$c = "/"; break}
                 {$i -gt 157} {$c = "|"; break}
                 {$i -gt 112} {$c = "\"; break}
                 {$i -gt 67} {$c = "-"; break}
                 {$i -gt 22} {$c = "/"; break}
                 default {$c = "|"}
           }
           printpos $c $ox $oy
     }
}
```

A függvény definíciója nem sok magyarázatot igényel. Az `$x`, `$y` a kör közepének koordinátái, az `$r` a sugara, a `$step` a kör kirajzolásának a „felbontása”, alapérték 5 fokonként rajzol, de lehet kisebbre venni

nagyobb köröknél, a `$deform` meg a konzol fontjainak téglalapos formáját hivatott kompenzálni. Plusz még a fok függvényében más és más karaktert rak ki a képernyőre, amelyek jobban szimulálják a körívet.

Ezzel ilyen szépségeket lehet rajzolni:



73. ábra Körrajzolás a konzolra

Sőt! Ezzel az osztállyal megoldható a várakozás nélküli billentyűzetfigyelés is:

```
function getkeysilent
{
    if([console]::KeyAvailable){
          while([console]::KeyAvailable){$key = [console]::readkey().Key}}
    else{$key = "nokey"}
    $key
}
```

A `KeyAvailable` statikus tulajdonság akkor válik igazzá, ha van leütött billentyűzet. Ennek vizsgálatával csak akkor olvasom be a `ReadKey` metódussal a billentyűt, ha már tudom, hogy le volt nyomva, így nem várakozik. Viszont ilyenkor elképzelhető, hogy a billentyűzet pufferbe több leütés is feltorlódott, így egészen addig kell olvasni, amíg a `KeyAvailable` jelez megnyomást. Visszatérési értékként a puffer utolsó billentyűzetlenyomását adom vissza.

A konzol tartalmát is be lehet olvasni, erre írtam egy célfüggvényt, mellyel egyszerűbb lesz egy karaktert beolvasni az x-y koordinátáról:

```
function getpos ($x, $y)
{
     if($x -ge 0 -and $y -ge 0 -and $x -le [Console]::WindowWidth -and
           $y -le [Console]::WindowHeight) {
     $y += [console]::WindowTop
     $r = New-Object System.Management.Automation.Host.Rectangle $x,$y,$x,$y
     $host.UI.RawUI.GetBufferContents($r)[0,0].character
     }
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
}
```

Ehhez nem a `[console]` osztály jó nekünk, hanem a `$host` automatikus változó, mely egy téglalap típusú osztályt vár paraméterként.

2.15.3 Böngészés

A következő gyakori feladat a weben található információk letöltése. Erre a célra a .NET keretrendszerben a `System.Net.WebClient` osztály áll rendelkezésre. Itt nem egy emberi fogyasztásra szánt web böngészőt kell elképzelni, hanem egy olyan alapszolgáltatást, ami csatlakozik egy adott URL segítségével a webkiszolgálóhoz és kiolvassa az adott weboldalt:

```
[2] PS C:\> $client = New-Object System.Net.WebClient
[3] PS C:\> $client.DownloadString("http://www.geocaching.hu")
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">


<html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859
-2">
        <title>geocaching.hu</title>
        <meta name="description" content="A magyar geocaching központja, geo
ládák
koodinátái, leírások, fényképek, letöltés, fórum.">
        <meta name="keywords" content="kincskeresés, geocaching, geochaching
, geoc
ashing, geokesing, gps, kincs, kincsvadászat, keresés, láda, geoláda, koord
ináta, koordináták, letöltés, kincsesláda, cache">
<link rel="stylesheet" href="style.css" type="text/css">
...
```

A [3]-as sorban a `DownloadString` metódussal indítom a böngészést. Az eredmény – mivel külön nem rendelkeztem róla – a konzolra került ki. Látható, hogy a teljes HTML tartalmat megkaptam. Természetesen szkriptjeimben ezt nem a képernyőre fogom kifolyatni, hanem egy változóba töltök be és valamilyen elemzés, átalakítás (például a HTML címkéktől való megszabadítás) után adom csak vissza az engem ténylegesen érdeklő információkat.

A `DonwloadString` metódus olvasható karaktereket vár azon a weboldalon, amelyre ráirányítjuk. Ha azonban egy bináris állományt, például egy ZIP fájlt akarunk letölteni, akkor ez nem lesz nekünk jó. Ilyen letöltésekre egy másik metódust, a `DownloadFile`-t használhatjuk:

```
[5] PS I:\>$client = New-Object System.Net.WebClient
[6] PS I:\>$url= "http://www.xs4all.nl/~hneel/software/bytecount.zip"
[7] PS I:\>$filename = "c:\bytecount.zip"
[8] PS I:\>$client.DownloadFile($url, $filename)
[9] PS I:\>Get-ChildItem c:\b*


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
-a---     2008. 04. 22.      9:53     419660 bytecount.zip
```

A fenti példában látható, hogy ennek a metódusnak az URL-en kívül egy fájl elérési útja is paramétere, ahova majd letölti a bináris állományt.

2.15.4 Felhasználói információk

A felhasználói információkat is egyszerűen összegyűjthetjük az alábbi `WindowsIdentity` osztály `GetCurrent` metódusával:

```
[1] PS I:\>[System.Security.Principal.WindowsIdentity]::GetCurrent()


AuthenticationType : Kerberos
ImpersonationLevel : None
IsAuthenticated    : True
IsGuest            : False
IsSystem           : False
IsAnonymous        : False
Name               : IQJB\soostibor
Owner              : S-1-5-21-861567501-1202660629-1801674531-6051
User               : S-1-5-21-861567501-1202660629-1801674531-6051
Groups             : {S-1-5-21-861567501-1202660629-1801674531, , , ...}
Token              : 904
```

A kimenetben láthatóak a legfontosabb adatok: a felhasználói név, a csoporttagságra utaló SID-ek, az autentikáció módja, stb.

Miután SID-ekkel nem annyira könnyű dolgozni, ezért egy másik osztállyal, a `WindowsPrincipal`-lal további szolgáltatásokhoz jutunk. Nagyon egyszerűen le lehet kérdezni például, hogy vajon az éppen aktuális felhasználó rendszergazda jogosultságokkal bír-e, azaz tagja-e az `Administrators` csoportnak:

```
[6] PS C:\> $u = [System.Security.Principal.WindowsIdentity]::GetCurrent()
[7] PS C:\> $principal = New-Object Security.principal.windowsprincipal($u)
[8] PS C:\> $principal.IsInRole("Administrators")
True
```

Ezekkel nagyon hatékonyan lehet megvizsgálni az éppen aktuális felhasználó különböző jellemzőit akár tartományi, akár helyi gépes környezetben.

2.15.5 DNS adatok lekérdezése

A DNS adatokról volt már szó a *3.5.4 Active Directory információk lekérdezése* fejezetben, de ott az Active Directory szempontjait vettem előtérbe. Ha egy általános DNS névfeloldást szeretnénk végrehajtani, arra a `System.Net.DNS` osztály használható, annak is a `GetHostByName` metódusa:

```
PS C:\Users\Administrator> [System.Net.Dns]::GetHostByName("www.microsoft.co
m")

HostName                Aliases                 AddressList
--------                -------                 -----------
lb1.www.ms.akadns.net   {www.microsoft.com, t... {207.46.192.254, 207....
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\Users\Administrator> [System.Net.Dns]::GetHostByName("www.microsoft.co
m").aliases
www.microsoft.com
toggle.www.ms.akadns.net
g.www.ms.akadns.net
PS C:\Users\Administrator> [System.Net.Dns]::GetHostByName("www.microsoft.co
m").addresslist


IPAddressToString : 207.46.192.254
Address           : 4274007759
AddressFamily     : InterNetwork
ScopeId           :
IsIPv6Multicast   : False
IsIPv6LinkLocal   : False
IsIPv6SiteLocal   : False

IPAddressToString : 207.46.193.254
Address           : 4274073295
AddressFamily     : InterNetwork
ScopeId           :
IsIPv6Multicast   : False
IsIPv6LinkLocal   : False
IsIPv6SiteLocal   : False

IPAddressToString : 207.46.19.254
Address           : 4262670031
AddressFamily     : InterNetwork
ScopeId           :
IsIPv6Multicast   : False
IsIPv6LinkLocal   : False
IsIPv6SiteLocal   : False

IPAddressToString : 207.46.19.190
Address           : 3188928207
AddressFamily     : InterNetwork
ScopeId           :
IsIPv6Multicast   : False
IsIPv6LinkLocal   : False
IsIPv6SiteLocal   : False
```

Látható, hogy ez egyszerű IP cím lekérdezésen kívül számos egyéb információhoz hozzájuthatunk.

A könyv tanf

2.16 SQL adatelérés

Az adatbázisok adatai is elérhetők a PowerShell segítségével, itt is .NET osztályokat hívhatunk segítségül. Itt én a Microsoft platformon legkönnyebben elérhető adatbázis-kezelőt, a Microsoft Access-t fogom alapul venni, de kis módosítással más adatbázis-kezelőket is hasonlóan meg lehet szólítani.

2.16.1 Microsoft Access

Access adatbázisok eléréséhez Rachard Siddawaytől olvashatunk sokat (http://msmvps.com/blogs/richardsiddaway/). Az ő ötleteit fejlesztettem tovább és így egy olyan függvénykönyvtárat készítettem, amelyben az Access alapvető adatbázis-kezelési helyzeteire vannak függvények:

```
function New-AccessDatabase {
param (
     [string] $path,
      [switch] $close,
      [switch] $passthru
     )

     if (!(Test-Path (split-path $path))){Throw "Invaild Folder"}
     if (Test-Path $path){Throw "File Already Exists"}

      $cat = New-Object -ComObject 'ADOX.Catalog'
     [Void] $cat.Create("Provider=Microsoft.ACE.OLEDB.12.0; Data Source=$path")
      $cat.ActiveConnection.Close()
     if(!$close){
            $connection = New-Object System.Data.OleDb.OleDbConnection(
            "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=$path")
      $connection.Open()
            if($passthru){$connection}
            else{$global:AccessConnection = $connection}
      }
}
```

Az első függvény az adatbázisok létrehozását végzi. Egy paramétert kell neki kötelezően megadni, ez pedig az adatbázisfájl elérési útja. A függvény elején van egy kis hibakezelés, ami a nem létező szülőkönyvtárat és a már létező fájlt szűri ki.

Ha használjuk a `-Close` kapcsolót a függvény hívásakor, akkor a létrehozott adatbázist be is zárja, ha nem, akkor egy élő kapcsolati objektumot ad vissza a függvény, méghozzá `-Passthru` kapcsoló nélkül a globális `$AccessConnection` változóba, egyébként meg csak „sima” visszatérési értékként.

Az én tapasztalatom az, hogy könnyű elfelejtkezni ennek a kapcsolati objektumnak a változóba történő mentéséről, így ez a globális változó a legkényelmesebb módja a kezelésének. Persze, ha párhuzamosan több adatbázis-kapcsolatot akarunk kezelni, akkor külön változókban érdemes ezeket tárolni.

A második függvény az adatbázis megnyitása:

```
function Open-AccessDatabase {
param (
     [string]$path,
      [switch] $passthru
)
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    if (!(Test-Path $path)){Throw "File Does Not Exist"}

    $connection = New-Object System.Data.OleDb.OleDbConnection(
          "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=$path")
    $connection.Open()
    if($passthru){$connection }
     else {$global:AccessConnection = $connection}
}
```

Ez nem igényel túl sok magyarázatot, itt is a `-Passthru` kapcsoló segítségével tudjunk visszatérési értéket adni, egyébként a `$AccessConnection` változóba tölti a kapcsolati objektumot.

Ha szükséges, akkor az adatbázist be is zárhatjuk, azaz a kapcsolatot megszűntetjük:

```
function Close-AccessDatabase {
param (
    [System.Data.OleDb.OleDbConnection]$connection
)
     if($connection){$connection.Close()}
     elseif($global:AccessConnection){$global:AccessConnection.Close()}
     else{throw "Nothing to close"}
}
```

Ha adunk neki át kapcsolati paramétert, akkor azt zárja be, egyébként próbálja a globális `$AccessConnection` tartalma alapján bezárni a kapcsolatot, különben hibát jelez.

Új adatbázisba érdemes táblát is létrehozni:

```
function New-AccessTable {
param (
    [string] $name,
     [hashtable] $columns,
    [System.Data.OleDb.OleDbConnection] $connection = $global:AccessConnection
)
     $ofs=","
     $col = [string] ($columns.Keys | %{"$_ $($columns.$_)"})
     $sql = "CREATE TABLE $name ($col)"
    $cmd = New-Object System.Data.OleDb.OleDbCommand($sql, $connection)
    $cmd.ExecuteNonQuery()
}
```

Ennek a függvénynek meg kell adni az új tábla nevét (`$name`) és az oszlopokat hashtábla formátumban és opcionálisan a kapcsolati objektumot. Ha ilyet nem adok, akkor a globális `$AccessConnection` változó tartalma alapján csatlakozik az adatbázishoz. A függvény törzsében felépítem a hashtábla tartalma alapján azt az SQL kifejezést, amely a táblát hozza létre.

Nézzünk ennek futtatására egy példát:

```
PS C:\> New-AccessTable -name AlapAdatok -columns @{
>>      ID = "COUNTER CONSTRAINT PrimaryKey PRIMARY KEY";
>>      Név = "Text (250)";
>>      Ár = "CURRENCY";
>>      Kell = "YESNO";
>>      Bevitel = "DATE DEFAULT Now()";
>>      Jegyzet = "MEMO"}
>>
0
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

Vigyázzunk, az általam létrehozott egyszerű függvényekkel nem lehet olyan oszlopokat és táblákat létrehozni, amelyek neve egy SQL-ben használatos lefoglalt kulcsszó. Így például „Memo” nevű oszlop létrehozásakor hibát kapunk.

Ez a kifejezés létrehoz egy táblát `AlapAdatok` névvel, lesz egy `ID` oszlop, ami az Access *AutoNumber* típusát veszi fel, és egyedi azonosítóként szolgál, lesz egy `Név` oszlop, ami szöveges és max. 250 karakter hosszúságú adatot fogad, lesz egy `Ár` oszlop, amely pénznem adatokat fogad, lesz egy bool típusú `Kell` oszlop, egy `Bevitel` oszlop, ami dátum típusú és alapértékként az aktuális dátumot veszi fel értékként és végül egy `Jegyzet` oszlop, ami hosszú szövegeket fogad.

Töltsük fel a táblát adatokkal! Ehhez az `Add-AccessRecord` függvény ad segítséget:

```
function Add-AccessRecord {
param (
    [string]$table,
     [hashtable] $record,
    [System.Data.OleDb.OleDbConnection]$connection = $global:AccessConnection
)
     $ofs=", "
     $cols = "$($record.keys)"
     $ofs = """, """
     $vals = """"+[string] ($record.Keys |
           foreach-object {$record.$_}) + """"
     $sql = "INSERT INTO $table ($cols) VALUES ($vals)"
    $cmd = New-Object System.Data.OleDb.OleDbCommand($sql, $connection)
    $cmd.ExecuteNonQuery()
}
```

A filozófia hasonló az előző függvényéhez, azaz meg kell adni a tábla nevét és hashtábla formátumban a beillesztendő adatsort `oszlopnév = érték` módon. Nézzünk erre is példát:

```
PS C:\> Add-AccessRecord -table AlapAdatok -record @{név = "Bizgentyű"; Ár = 15
26; Kell = -1; Jegyzet = "Ez az első teszt adat"}
1
PS C:\> Add-AccessRecord -table AlapAdatok -record @{név = "Herkentyű"; Ár = 98
76; Kell = 0; Jegyzet = "Ez a második adat"; Bevitel = "2010.01.02"}
1
```

Természetesen érdemes az adatokat visszaolvasni tudni, ehhez a következő függvényt készítettem:

```
function Get-AccessTableRecords {
param (
    [string[]] $columns ="*",
     [string] $table,
     [string] $where,
    [System.Data.OleDb.OleDbConnection] $connection = $global:AccessConnection
)
     $ofs = ", "
     $cols = [string] $columns

     $wh = if($where){" WHERE $where"} else {""}
    $sql = "SELECT $cols FROM $table" + $wh
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
    $cmd = New-Object System.Data.OleDb.OleDbCommand($sql, $connection)
    $reader = $cmd.ExecuteReader()

    $dt = New-Object System.Data.DataTable
    $dt.Load($reader)
    $dt
}
```

Ez nem egy általános lekérdező függvény, ez kifejezetten egy tábla adatsorait tudja lekérdezni. Az első paramétere a `-Columns`, ami egy sztringtömböt vár. Ez adja meg, hogy a tábla mely oszlopait szeretném látni, ha nem adok meg semmit, akkor mindegyiket. Paraméter még, hogy melyik tábla sorait kérdezzük le, és meg lehet még adni egy szűrő feltételt is az SQL WHERE klauzulájának megfelelően.

Nézzük ennek is a kimenetét, az első esetben oszlopok megadásával:

```
PS C:\> Get-AccessTableRecords -columns név, jegyzet -table AlapAdatok

név                                    jegyzet
---                                    -------
Bizgentyű                              Ez az első teszt adat
Herkentyű                              Ez a második adat
```

A második esetben oszlopok nélkül. Itt látható, hogy a PowerShell 5 tulajdonság felett alaphelyzetben a listanézetet preferálja:

```
PS C:\> Get-AccessTableRecords -table AlapAdatok


ID      : 2
Bevitel : 2010.02.22. 15:43:42
Kell    : True
Jegyzet : Ez az első teszt adat
Név     : Bizgentyű
Ár      : 1526

ID      : 3
Bevitel : 2010.01.02. 0:00:00
Kell    : False
Jegyzet : Ez a második adat
Név     : Herkentyű
Ár      : 9876
```

Végül egy példa a szűrésre:

```
PS C:\> Get-AccessTableRecords -table AlapAdatok -where "név = 'Bizgentyű'"


ID      : 2
Bevitel : 2010.02.22. 15:43:42
Kell    : True
Jegyzet : Ez az első teszt adat
Név     : Bizgentyű
Ár      : 1526
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

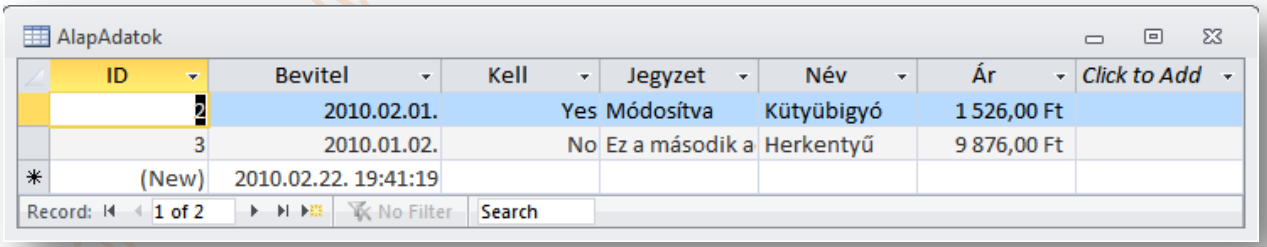
Utolsóként nézzünk egy adatmódosító függvényt! Itt már a fejlett függvények lehetőségeit is kihasználtam, hogy lehessen `-WhatIf` kapcsolóval „óvatosan” is futtatni:

```
function Set-AccessData {
[CmdletBinding(SupportsShouldProcess=$true)]
param (
    [string]$table,
    [string]$filter,
    [hashtable] $value,
    [System.Data.OleDb.OleDbConnection]$connection = $global:AccessConnection
)
    $ofs = ", "
    $set = [string] ($value.keys | ForEach-Object {"$_ = '$($value.$_)'"})
    $sql = "UPDATE $table SET $set WHERE $filter"
    $cmd = New-Object System.Data.OleDb.OleDbCommand($sql, $connection)

    if ($psCmdlet.ShouldProcess("$($connection.DataSource)", "$sql"))
        {$cmd.ExecuteNonQuery()}
}
```

A működés logikája a korábbiakhoz hasonló, a `-Table` paraméternek kell megadni, hogy melyik tábla adatát szeretnénk módosítani, a `-Filter` paraméterhez egy SQL szintaxisú „WHERE” klauzulát kell megadni. A `-Value` paraméterhez megint csak egy hashtábla formátumban kell megadni az oszlopnév – új érték párokat a következő példa alapján:

```
PS C:\> Set-AccessData -table AlapAdatok -filter 'név = "Bizgentyű"' -value @{
>> név = "Kütyübigyó";
>> Jegyzet = "Módosítva";
>> Bevitel = "2010.02.01"
>> }
>>
1
```

Visszaolvasva az adatokat már az újakat látjuk:



74. ábra A PowerShellel létrehozott Access tábla adatokkal

Vagy ugyanez a függvényeim segítségével:

```
PS C:\> Get-AccessTableRecords -table AlapAdatok | Format-Table -AutoSize

ID Bevitel              Kell Jegyzet           Név          Ár
-- -------              ---- -------           ---          --
 2 2010.02.01. 0:00:00  True Módosítva         Kütyübigyó 1526
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
 3 2010.01.02. 0:00:00 False Ez a második adat Herkentyű  9876
```

Természetesen ezt még tovább lehet fejleszteni, de itt csak annyi volt a célom, hogy olyan adatkezelési igényekre adjak PowerShelles választ, amelyekhez nem kell mély SQL tudás.

Megjegyzés

A 64-bites Windows XP-n a fenti SQL-es PowerShell kifejezések hibát adnak a „normál” PowerShell ablakban futtatva. Ennek oka az, hogy a szükséges ODBC driver 64-bites változata nincsen benne az operációs rendszerben, csak a 32-bites változat, amit csak 32 bites alkalmazásból szólíthatunk meg. Ha ilyen hibát tapasztalunk, akkor futtassuk a fenti kódot 32-bites PowerShell ablakban, ami szintén elérhető a 64-bites gépen is, az már működni fog minden baj nélkül.

2.16.2 Windows Search

Az SQL-hez nagyon hasonlóan lehet megszólítani a Desktop Search, új nevén Windows Search keresőt:

```
$connectionString = "Provider=Search.CollatorDSO;Extended `
    Properties='Application=Windows';"
$sqlCommand = "SELECT TOP 20 `
    System.itemurl, system.document.pagecount, system.datecreated, `
    system.author, system.size, system.itemname`
    FROM SYSTEMINDEX `
    WHERE Freetext('PowerShell')"
$connection = New-Object System.Data.OleDb.OleDbConnection $connectionString
$command = New-Object System.Data.OleDb.OleDbCommand $sqlCommand, $connection
$connection.Open()
$adapter = New-Object System.Data.OleDb.OleDbDataAdapter $command
$dataset = New-Object System.Data.DataSet
[void] $adapter.Fill($dataSet)
$connection.Close()
$records = $dataSet.Tables | Select-Object -Expand Rows
$records | format-table
```

Annyi a különbség az „igazi” SQL lekérdezéshez képest, hogy itt a tábla adott, a `SYSTEMINDEX`. Speciális kulcsszavakat használhatunk, például a `Freetext`-et, mellyel bármelyik tulajdonságában keres.

```
SYSTEM.ITEMUR SYSTEM.DOCUM SYSTEM.DATEC SYSTEM.AUTHO  SYSTEM.SIZE SYSTEM.ITEMN
L             ENT.PAGECOUN REATED       R                       AME
              T
------------- ------------ ------------ ------------  ----------- ------------
mapi://{S-...              2010.02.1... {Osama Sa...        19661 MVP Summi...
file:C:/_m... 1            2010.01.0... {soostibor}       3604752 Microsoft...
mapi://{S-...              2010.02.1... {Nathan B...      1982842 HardwareM...
file:C:/_m... 5            2010.02.1... {tibi}              82653 hwsw.docx
mapi://{S-...              2010.02.0... {Soós Tibor}         7396 Windows S...
mapi://{S-...              2010.02.1... {Michael ...        30499 [PSMVP] A...
file:C:/_m...              2010.01.2...                    3160344 adhelp.txt
mapi://{S-...              2010.02.1... {Max Trin...       3785098 [PSMVP] M...
mapi://{S-...              2010.01.2... {Susan Br...        15564 [PSMVP] F...
mapi://{S-...              2010.01.2... {Thomas Lee}        11024 [PSMVP] F...
mapi://{S-...              2009.12.1... {Manning ...        11343 Download ...
mapi://{S-...              2009.12.1... {manning....        11974 order yhs...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
mapi://{S-...              2009.12.1... {halr9000...        21249 [PSMVP] C...
mapi://{S-...              2010.01.2... {support@...        14436 Windows P...
mapi://{S-...              2009.12.1... {Keith Hill}      1069228 [PSMVP] P...
mapi://{S-...              2009.12.1... {Joel Ben...        13404 [PSMVP] P...
mapi://{S-...              2009.12.1... {karl pro...        11110 [PSMVP] i...
mapi://{S-...              2010.01.1... {Shay Levy}         12219 [PSMVP] B...
mapi://{S-...              2009.12.1... {Soós Tibor}        14384 rev2: Újé...
mapi://{S-...              2009.12.1... {Soós Tibor}        13714 Újévi ajá...
```

Látható, hogy ezzel mind a fájlrendszerben, mind az Outlook elemek között keres a kifejezés.

A könyv tanfolyami felhasználása nem engedélyezett!

2.17 COM objektumok kezelése

A *Component Object Model (COM)* a Microsoft által 1993-ban kifejlesztett interfész szabvány szoftverkomponensek közti kommunikációra. Segítségével minden ilyen programnyelven, amely támogatja ezt a szabványt, lehet készíteni olyan szoftverkomponenseket, amelyek képesek egymással kommunikálni és dinamikusan egymás objektumait létrehozni, kezelni. Sőt! Akár szkriptnyelvekből (VBScript, PowerShell) is meg lehet szólítani ezeket a komponenseket, mint ahogy az alábbiakban látható lesz.

A COM-ba számos más „altechnológia” tartozik: OLE, OLE Automation, ActiveX, COM+ és DCOM.

2.17.1 A Windows shell kezelése

Maga a Windows grafikus keretprogramja, shellje is COM objektum, azaz megszólítható, a publikus metódusai és tulajdonságai meghívhatók, lekérdezhetőek. Ehhez először meg kell hívni az shellt, ezt a PowerShellben nagyon egyszerűen, a `new-object -com` commandlettel tehetjük meg:

```
[2] PS C:\> $sh = new-object -com Shell.Application
```

Miután ezt az objektumot még többször akarom használni, ezért egy `$sh` változóba tettem. Nézzük meg ennek tagjellemzőit:

```
[3] PS C:\> $sh | gm


   TypeName: System.__ComObject#{efd84b2d-4bcf-4298-be25-eb542a59fbda}

Name                 MemberType Definition
----                 ---------- ----------
AddToRecent          Method     void AddToRecent (Variant, string)
BrowseForFolder      Method     Folder BrowseForFolder (int, string, int...
CanStartStopService  Method     Variant CanStartStopService (string)
CascadeWindows       Method     void CascadeWindows ()
ControlPanelItem     Method     void ControlPanelItem (string)
EjectPC              Method     void EjectPC ()
Explore              Method     void Explore (Variant)
ExplorerPolicy       Method     Variant ExplorerPolicy (string)
FileRun              Method     void FileRun ()
FindComputer         Method     void FindComputer ()
FindFiles            Method     void FindFiles ()
FindPrinter          Method     void FindPrinter (string, string, string)
GetSetting           Method     bool GetSetting (int)
GetSystemInformation Method     Variant GetSystemInformation (string)
Help                 Method     void Help ()
IsRestricted         Method     int IsRestricted (string, string)
IsServiceRunning     Method     Variant IsServiceRunning (string)
MinimizeAll          Method     void MinimizeAll ()
NameSpace            Method     Folder NameSpace (Variant)
Open                 Method     void Open (Variant)
RefreshMenu          Method     void RefreshMenu ()
ServiceStart         Method     Variant ServiceStart (string, Variant)
ServiceStop          Method     Variant ServiceStop (string, Variant)
SetTime              Method     void SetTime ()
ShellExecute         Method     void ShellExecute (string, Variant, Vari...
ShowBrowserBar       Method     Variant ShowBrowserBar (string, Variant)
ShutdownWindows      Method     void ShutdownWindows ()
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Suspend              Method     void Suspend ()
TileHorizontally     Method     void TileHorizontally ()
TileVertically       Method     void TileVertically ()
ToggleDesktop        Method     void ToggleDesktop ()
TrayProperties       Method     void TrayProperties ()
UndoMinimizeALL      Method     void UndoMinimizeALL ()
Windows              Method     IDispatch Windows ()
WindowsSecurity      Method     void WindowsSecurity ()
Application          Property   IDispatch Application () {get}
Parent               Property   IDispatch Parent () {get}
```

Ezzel láthatóvá váltak a `Shell.Application` tagjellemzői, azaz a Windows szkriptből is könnyen elérhető szolgáltatásai, mint például a súgó megnyitása vagy akár egy mappa megnyitása a Windows Intézőben:

```
[4] PS C:\> $sh.help()
[5] PS C:\> $sh.explore("C:\scripts")
```

Természetesen az így megnyitott Súgóval vagy az Intézővel olyan sok mindent nem tudunk PowerShellel továbbiakban kezdeni, ez inkább csak a COM objektumok kezelésének a legegyszerűbb és látványos demonstrációja akart lenni. Sokkal praktikusabb lett volna például a `ShutdownWindows` metódus meghívása, de ezt a tisztelt olvasóra bízom ☺.

2.17.2 WScript osztály használata

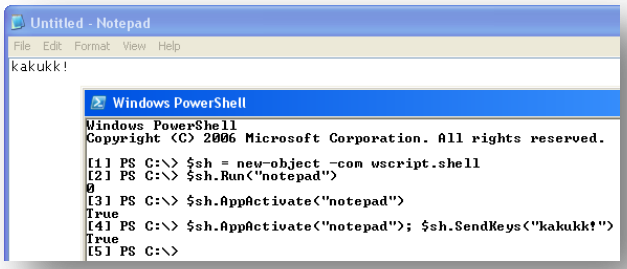
Az előzőekben láttuk, hogy a Windows Shellt hogyan lehet szkriptből megszólítani. Van egy másik COM objektum, a `WScript`, amelynek `Shell` alosztálya is hasonló célokat, de még mélyebb szinten valósít meg. Például szeretnénk egy olyan alkalmazással kommunikálni, amely nem COM alkalmazás, például a Notepaddal. Jobb híján billentyűzetleütéseket tudunk neki küldeni a `WScript.Shell` osztály `SendKeys` metódusával:

```
[1] PS C:\> $sh = new-object -com wscript.shell
[2] PS C:\> $sh.Run("notepad")
0
[3] PS C:\> $sh.AppActivate("notepad")
True
[4] PS C:\> $sh.AppActivate("notepad"); $sh.SendKeys("kakukk!")
True
```

Az [1]-es sorban elindítom a Notepad-et a `Run` metódussal. Aztán előtérbe helyezem az ablakát, hogy ő kapja a billentyűleütéseket, majd kiküldöm a billentyűzetütéseket. Az alábbi képernyőfotón látszik, hogy a kiadott karakterek tényleg beíródtak a Notepad szerkesztőfelületére.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



75. ábra Nem COM alkalmazás vezérlése WScrip.Shell objektummal

Nézzük ennek az objektumnak a tagjellemzőit:

```
[5] PS C:\> $sh | gm


   TypeName: System.__ComObject#{41904400-be18-11d3-a28b-00104bd35090}

Name                      MemberType            Definition
----                      ----------            ----------
AppActivate               Method                bool AppActivate (Variant...
CreateShortcut            Method                IDispatch CreateShortcut ...
Exec                      Method                IWshExec Exec (string)
ExpandEnvironmentStrings  Method                string ExpandEnvironmentS...
LogEvent                  Method                bool LogEvent (Variant, s...
Popup                     Method                int Popup (string, Varian...
RegDelete                 Method                void RegDelete (string)
RegRead                   Method                Variant RegRead (string)
RegWrite                  Method                void RegWrite (string, Va...
Run                       Method                int Run (string, Variant,...
SendKeys                  Method                void SendKeys (string, Va...
Environment               ParameterizedProperty IWshEnvironment Environme...
CurrentDirectory          Property              string CurrentDirectory (...
SpecialFolders            Property              IWshCollection SpecialFol...
```

Látható, hogy a billentyűleütések mellet többek között registry-szerkesztő, shortcut-létrehozó és dialógusablak-kirakó metódusai is vannak.

A WScriptnek nem csak Shell alosztálya van, hanem Network is. Ezzel számtalan hasznos, hálózattal kapcsolatos műveletet tudunk végrehajtani:

```
[10] PS I:\>$ws = New-Object -com WScript.Network
[11] PS I:\>$ws | gm


   TypeName: System.  ComObject#{24be5a31-edfe-11d2-b933-00104b365c9f}

Name                        MemberType Definition
----                        ---------- ----------
AddPrinterConnection        Method     void AddPrinterConnection (string...
AddWindowsPrinterConnection Method     void AddWindowsPrinterConnection ...
EnumNetworkDrives           Method     IWshCollection EnumNetworkDrives ()
EnumPrinterConnections      Method     IWshCollection EnumPrinterConnect...
MapNetworkDrive             Method     void MapNetworkDrive (string, str...
RemoveNetworkDrive          Method     void RemoveNetworkDrive (string, ...
RemovePrinterConnection     Method     void RemovePrinterConnection (str...
```

```
SetDefaultPrinter           Method     void SetDefaultPrinter (string)
ComputerName                Property   string ComputerName () {get}
Organization                Property   string Organization () {get}
Site                        Property   string Site () {get}
UserDomain                  Property   string UserDomain () {get}
UserName                    Property   string UserName () {get}
UserProfile                 Property   string UserProfile () {get}
```

Nézzünk például egy hálózati meghajtók felsorolását:

```
[12] PS C:\> $ws = New-Object -com WScript.Network
[13] PS C:\> $ws.EnumNetworkDrives()
I:
\\K-FILE1\soostibor$
```

2.17.3 Alkalmazások kezelése

A COM által nyújtott interfész lehetővé teszi, hogy nem csak a Windows különböző funkcióit tudjuk elérni, hanem a legtöbb alkalmazás is elérhetővé tesz olyan objektumokat, amelyek hasznos szolgáltatásait felhasználhatjuk szkriptjeinkben.

2.17.3.1 Internet Explorer

A legegyszerűbb vonatkozó példa az Internet Explorer megszólítása. Elsőként látható módon elindítom az alkalmazást:

```
[17] PS C:\> $ie = New-Object -ComObject InternetExplorer.Application
[18] PS C:\> $ie.visible=$true
```

Nézzük meg, hogy mit tartalmaz még a `$ie` változó:

```
PS C:\> $ie | Get-Member


   TypeName: System.__ComObject#{d30c1661-cdaf-11d0-8a3e-00c04fc9e26e}

Name                 MemberType Definition
----                 ---------- ----------
ClientToWindow       Method     void ClientToWindow (int, int)
ExecWB               Method     void ExecWB (OLECMDID, OLECMDEXECOPT, Varia...
GetProperty          Method     Variant GetProperty (string)
GoBack               Method     void GoBack ()
GoForward            Method     void GoForward ()
GoHome               Method     void GoHome ()
GoSearch             Method     void GoSearch ()
Navigate             Method     void Navigate (string, Variant, Variant, Va...
Navigate2            Method     void Navigate2 (Variant, Variant, Variant, ...
PutProperty          Method     void PutProperty (string, Variant)
QueryStatusWB        Method     OLECMDF QueryStatusWB (OLECMDID)
Quit                 Method     void Quit ()
Refresh              Method     void Refresh ()
Refresh2             Method     void Refresh2 (Variant)
ShowBrowserBar       Method     void ShowBrowserBar (Variant, Variant, Vari...
Stop                 Method     void Stop ()
AddressBar           Property   bool AddressBar () {get} {set}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Application           Property   IDispatch Application () {get}
Busy                  Property   bool Busy () {get}
Container             Property   IDispatch Container () {get}
Document              Property   IDispatch Document () {get}
FullName              Property   string FullName () {get}
FullScreen            Property   bool FullScreen () {get} {set}
Height                Property   int Height () {get} {set}
HWND                  Property   int64 HWND () {get}
Left                  Property   int Left () {get} {set}
LocationName          Property   string LocationName () {get}
LocationURL           Property   string LocationURL () {get}
MenuBar               Property   bool MenuBar () {get} {set}
Name                  Property   string Name () {get}
Offline               Property   bool Offline () {get} {set}
Parent                Property   IDispatch Parent () {get}
Path                  Property   string Path () {get}
ReadyState            Property   tagREADYSTATE ReadyState () {get}
RegisterAsBrowser     Property   bool RegisterAsBrowser () {get} {set}
RegisterAsDropTarget  Property   bool RegisterAsDropTarget () {get} {set}
Resizable             Property   bool Resizable () {get} {set}
Silent                Property   bool Silent () {get} {set}
StatusBar             Property   bool StatusBar () {get} {set}
StatusText            Property   string StatusText () {get} {set}
TheaterMode           Property   bool TheaterMode () {get} {set}
ToolBar               Property   int ToolBar () {get} {set}
Top                   Property   int Top () {get} {set}
TopLevelContainer     Property   bool TopLevelContainer () {get}
Type                  Property   string Type () {get}
Visible               Property   bool Visible () {get} {set}
Width                 Property   int Width () {get} {set}
```

Navigáljunk el egy weboldalra:

```
PS C:\> $ie.Navigate("http://www.iqjb.hu")
PS C:\> $ie.Busy
False
```

Érdemes vizsgálni a `Busy` tulajdonságot, mert amikor annak értéke `$False`-ra vált, akkor fejeződött be az oldal letöltése. Nézzünk bele a letöltött oldalba:

```
PS C:\> $ie.Document


Script                              :
all                                 : System.__ComObject
body                                : System.__ComObject
activeElement                       : System.__ComObject
images                              : System.__ComObject
applets                             : System.__ComObject
links                               : System.__ComObject
forms                               : System.__ComObject
anchors                             : System.__ComObject
title                               : IQSOFT - John Bryce Oktatóközpont
scripts                             : System.__ComObject
designMode                          : Inherit
selection                           : System.__ComObject
readyState                          : complete
frames                              :
embeds                              : System.__ComObject
```

```
plugins                           : System.__ComObject
alinkColor                        : #0000ff
bgColor                           : #ffffff
fgColor                           : #17426a
linkColor                         : #0000ff
vlinkColor                        : #800080
…
```

Nagyon sok tulajdonsága van, így csak egy részét másoltam ide. Ezek közül a weboldal HTML tartalma kiolvasható a következő tulajdonságon keresztül:

```
PS C:\> $ie.Document.body.innerHTML
<TABLE border=0 cellSpacing=0 cellPadding=0 width="99%">
<TBODY>
<TR>
<TD width="99%">
<TABLE border=0 cellSpacing=0 cellPadding=0 width="100%">
<TBODY>
<TR>
<TD background=/Images/Page2/fejlechatter.gif align=left><A href="http://www.i
qjb.hu" target=_parent><IMG border=0 src="/Images/Page2/fejlec.gif" useMap=#fe
jlecMap></A></TD></TR></TBODY></TABLE></TD></TR></TBODY></TABLE>
<TABLE border=0 cellSpacing=0 cellPadding=0 width="99%">
…
```

Az emberi szemnek olvasható nyers szöveg pedig kinyerhető a következő tulajdonsággal:

```
PS C:\> $ie.Document.body.innerText | ForEach-Object {$_ -replace "\r\n", ""}
Microsoft Tesztelés tanfolyamok Biztonságtechnikai tanfolyamok Java, C++, Corb
a, XML Eclipse Cisco, hálózati technológiák Oracle (BEA) IBM Sybase Agilis, UM
L, RUP modellezési, tervezési módszertanok SuSE Linux Check Point ITIL, ITC, P
rojektirányítás, Minőségbiztosítás Szemináriumok Irodai szoftverek Pénzügy, HR
…
```

(A sok soremelés miatt ezeket egy `-replace` kifejezéssel kiszedtem.)

2.17.3.2 Microsoft Word

Nézzünk egy kicsit bonyolultabb példát, olvastassunk be egy Word dokumentumot, de csak az emberi fogyasztásra szánt értelmes szöveget! Ez nem olyan egyszerű feladat, hiszen ha egyszerű fájlművelettel próbálkoznánk, akkor a Word dokumentum mindenféle formázó információját is beolvassuk, és abból elég nehéz kinyerni a tényleges szöveget. Szerencsére a Word is rendelkezik COM felülettel, így az alábbi néhány soros szkripttel könnyen felolvastathatjuk az „igazi” szöveget a dokumentumból:

```
[25] PS I:\>$wordApp = New-Object -COM Word.Application
[26] PS I:\>$file = (Get-Item C:\_docs\tematikák.docx).FullName
[27] PS I:\>$doc = $wordApp.Documents.Open($file)
[28] PS I:\>$text = $doc.Content.Text
[29] PS I:\>$text
Microsoft PowerShell for Administrators Who Should AttendAnyone Who Scrip
ts For Windows - this course will help you build scripting skills in PowerS
hell when you are coming from a background in scripting on Windows operatin
...
[30] PS I:\>$wordApp.Quit()
```

könyv tanfolyami felhasználása nem engedélyezett!

A szkript elején megszólítom a `Word.Application` COM objektumot és betöltöm a `$wordApp` változóba, majd a megnyitandó dokumentum elérési útját berakom egy változóba, majd a `$wordApp` segítségével megnyitom ezt a fájlt. Mivel nem rendelkeztem arról, hogy a Word látható legyen, mindez csak a háttérben történik. A fájl megnyitásával egy új objektumhoz jutok, magához a dokumentumhoz, majd ennek veszem a nyers szöveges részét a `$doc.Content.Text` kifejezés segítségével, amit a [29]-es sorban ki is írattam a konzolra.

Ilyen jellegű alkalmazások kezelésekor illik azokat a végén bezárni, hogy ne foglalja feleslegesen a memóriát. Ezt a [30]-as sorban tettem meg.

2.17.3.3 Microsoft Excel

A Microsoft Excel is elérhető COM objektumként. Sajnos az Excel 2010 előtti verziók esetében, ha nem amerikai angol a Windows területi beállítása, akkor egy hiba miatt a cellák elérésénél hibával „elszállt” a művelet. Az alábbi példában Excel 2010-et használtam, itt már nem jött elő ez a hiba.

```
$excelapp = New-Object -comobject Excel.Application

$excelapp.Visible = $True

$workbook = $excelapp.Workbooks.Add()
$sheet = $workbook.Worksheets.Item(1)

$sheet.Cells.Item(1,1) = 7
$sheet.Cells.Item(1,2) = "=fact(a1)"
```

A fenti példában megnyitottam az Excel alkalmazást COM objektumként, láthatóvá tettem, majd nyitottam egy munkafüzetet és annak első lapját a `$sheet` változóhoz rendeltem. Ennek a munkalapnak az „A1”-es cellájába 7-et töltöttem, a „B1”-es cellájába egy Excel függvényt, „=fact(a1)”-et.

Visszaolvasni ezeket a következő módon lehet:

```
PS C:\> $sheet.Cells.Item(1,2).formula
=FACT(A1)
PS C:\> $sheet.Cells.Item(1,2).value2
5040
```

Látható, hogy ki lehet olvasni mind a képletet (`formula` tulajdonság), mind az értéket (`value2` tulajdonság).

Munkafüzeteket elmenteni a legfrissebb Excel verziókban nem egyszerű, mert kötelező megadni a formátumot, nincs alaphelyzet szerinti érték. Az alábbi példában a normál módon történő mentés látható, majd zárom a munkalapot:

```
$workbook.SaveAs("C:\_munka\Test.xlsx",-4143)
$workbook.Close()
```

Az alábbi táblázat a lehetséges mentési formátumokat tartalmazza:

Név	Érték	Jelentés
xlAddIn	18	Microsoft Excel 97-2003 Add-In
xlAddIn8	18	Microsoft Excel 97-2003 Add-In

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

xlCSV	6	CSV
xlCSVMac	22	Macintosh CSV
xlCSVMSDOS	24	MSDOS CSV
xlCSVWindows	23	Windows CSV
xlCurrentPlatformText	-4158	Current Platform Text
xlDBF2	7	DBF2
xlDBF3	8	DBF3
xlDBF4	11	DBF4
xlDIF	9	DIF
xlExcel12	50	Excel12
xlExcel2	16	Excel2
xlExcel2FarEast	27	Excel2 FarEast
xlExcel3	29	Excel3
xlExcel4	33	Excel4
xlExcel4Workbook	35	Excel4 Workbook
xlExcel5	39	Excel5
xlExcel7	39	Excel7
xlExcel8	56	Excel8
xlExcel9795	43	Excel9795
xlHtml	44	HTML format
xlIntlAddIn	26	International Add-In
xlIntlMacro	25	International Macro
xlOpenDocumentSpreadsheet	60	OpenDocument Spreadsheet
xlOpenXMLAddIn	55	Open XML Add-In
xlOpenXMLTemplate	54	Open XML Template
xlOpenXMLTemplateMacroEnabled	53	Open XML Template Macro Enabled
xlOpenXMLWorkbook	51	Open XML Workbook
xlOpenXMLWorkbookMacroEnabled	52	Open XML Workbook Macro Enabled
xlSYLK	2	SYLK
xlTemplate	17	Template
xlTemplate8	17	Template 8
xlTextMac	19	Macintosh Text
xlTextMSDOS	21	MSDOS Text
xlTextPrinter	36	Printer Text
xlTextWindows	20	Windows Text
xlUnicodeText	42	Unicode Text
xlWebArchive	45	Web Archive
xlWJ2WD1	14	WJ2WD1

A könyv tanfolyami felhasználása nem engedélyezett!

xlWJ3	40	WJ3
xlWJ3FJ3	41	WJ3FJ3
xlWK1	5	WK1
xlWK1ALL	31	WK1ALL
xlWK1FMT	30	WK1FMT
xlWK3	15	WK3
xlWK3FM3	32	WK3FM3
xlWK4	38	WK4
xlWKS	4	Worksheet
xlWorkbookDefault	51	Workbook default
xlWorkbookNormal	-4143	Workbook normal
xlWorks2FarEast	28	Works2 FarEast
xlWQ1	34	WQ1
xlXMLSpreadsheet	46	XML Spreadsheet

Ha egy meglevő Excel táblát akarunk megnyitni, akkor ezt a megnyitott Excel com objektum birtokában a következőképpen tehetjük ezt meg:

```
$excelapp.Workbooks.Open("C:\_munka\Test.xlsx")
$workbook = $excelapp.Workbooks.Item(1)
$sheet = $workbook.Worksheets.item(1)
$sheet.Cells.item(1,3) = "Save után"
$workbook.Save()

$excelapp.Quit()
```

Ilyenkor az „újramentés” már egyszerűbb, a `Save` metódus nem igényel paramétert.

2.17.4 Windows Update

Egy rendszerüzemeltető számára nagyon fontos a Windows operációs rendszerek frissítésének állapota. Ezt a Windows Update kliens szolgáltatás végzi, ennek állapotát a `Microsoft.Update.Autoupdate` COM objektumon keresztül tudjuk lekérdezni.

```
[22] PS C:\> $o = New-Object -ComObject Microsoft.Update.Autoupdate
[23] PS C:\> $o | gm


   TypeName: System.__ComObject#{4a2f5c31-cfd9-410e-b7fb-29a653973a0f}

Name                MemberType Definition
----                ---------- ----------
DetectNow           Method     void DetectNow ()
EnableService       Method     void EnableService ()
Pause               Method     void Pause ()
Resume              Method     void Resume ()
ShowSettingsDialog Method     void ShowSettingsDialog ()
Results             Property   IAutomaticUpdatesResults Results () {get}
ServiceEnabled      Property   bool ServiceEnabled () {get}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Settings           Property   IAutomaticUpdatesSettings Settings () {get}
```

Sőt! Nem csak az állapotát tudjuk lekérdezni, hanem vezérelni is tudjuk bizonyos mértékig. Például elindíthatunk egy frissességi vizsgálatot, azaz azt a folyamatot, hogy ellenőrizze a Windows Update kliens, hogy a rendelkezésére álló javítócsomagok közül mindegyik telepítve van-e már, és a hiányzókat telepíti.

```
[24] PS C:\> $o.detectnow()
[25] PS C:\> $o.results

LastSearchSuccessDate                LastInstallationSuccessDate
---------------------                ---------------------------
2009. 12. 05. 15:56:46               2009. 11. 25. 8:13:07
```

Illetve lekérdezhetjük az aktuális beállításait:

```
[26] PS C:\> $o.settings


NotificationLevel         : 4
ReadOnly                  : False
Required                  : False
ScheduledInstallationDay  : 0
ScheduledInstallationTime : 3
IncludeRecommendedUpdates : True
NonAdministratorsElevated : True
FeaturedUpdatesEnabled    : False
```

Természetesen az előző példák csak ízelítők próbáltak lenni a COM objektumok kezeléséből, nem kívántam egy komplett COM szakkönyvet írni, hiszen ennek már széles a szakirodalma. A lényeg az, hogy bármilyen COM objektum nagyon egyszerűen megszólítható PowerShellből, így egy meglevő, például VBScriptben írt példa nagyon egyszerűen átemelhető PowerShellbe.

A könyv tanfolyami felhasználása nem engedélyezett!

2.18 Eseményvezérelt futtatás

Előfordulnak olyan helyzetek, amikor a szkriptünknek valamilyen eseményre kellene reagálnia: eltelik valamennyi idő, létrejön egy új meghajtó, beesik egy speciális eseménynapló-bejegyzés, új fájl vagy könyvtár jön létre egy mappában, stb. Ilyen helyzetekben készíthetünk egy ciklust, ami az esemény bekövetkezéséig fut, és periodikusan ellenőrzi, hogy az esemény bekövetkezett-e. Ez nem annyira profi megoldás, mert nehéz eldönteni, hogy milyen rendszerességgel ellenőrizzük az esemény bekövetkeztét? Ha túl gyakran ellenőrizzük, akkor szegény számítógépünk értékes processzorteljesítményét erre pazaroljuk. Ha túl ritkásan, akkor meg esetleg túl sokára reagálunk az eseményre. A profi megoldás az, hogy felhasználjuk a Windows eseménykezelő rendszerét, merthogy az egész Windows működésének alapja az eseménykezelés: a kattintások, billentyűleütések mind-mind eseményeket generálnak, amelyeket az egyes ablakok lekezelnek. Szerencsére ehhez az infrastruktúrához a PowerShell 2.0-ban már hozzáférünk.

A PowerShell 2.0-ban háromfajta eseményre tudunk reagálni: EngineEvent, ObjectEvent és WMIEvent. A reagálás akkor történhet meg, ha előtte „előfizetünk” valamely eseményre. De nézzük ezeket egyesével!

2.18.1 EngineEvent kezelése

Ez az `EngineEvent` eseménytípus a legegyszerűbb. Olyan események ezek, amelyek a PowerShell processzel történhetnek, a processz elindulhat és befejeződhet. Sajnos az elindulást nem tudjuk figyelni, hiszen ahhoz még az elindulása előtt kellene előfizetnünk rá, ami fizikai képtelenség. Így marad annak az eseménynek a figyelése, amikor a PowerShell alkalmazást, processzt bezárjuk. Azért, hogy ne legyen ez ilyen egyhangú, még van egy egyedi eseménygenerálási lehetőség a `new-event` cmdlet segítségével, amikor is bármilyen, általunk kitalált esemény bekövetkeztét jelezhetjük.

Nézzük a folyamatot! Először is elő kell fizetni valamely esemény megfigyelésére a `Register-EngineEvent` cmdlet segítségével. Elsőként a PowerShell bezárását nézzük meg:

```
[1] PS C:\> Register-EngineEvent -SourceIdentifier PowerShell.Exiting -Action {
write-host "Minden jót!"; Start-Sleep 20; "kiléptem..."}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
1               PowerShell.E... NotStarted False


[2] PS C:\> exit
Minden jót!
```

Regisztráltam tehát egy eseményfigyelőt, ami a PowerShell kilépésére végrehajtja az `-Action` paraméternek átadott szkriptblokkot. Itt egy kis várakozást kellett beépítenem a folyamatba, különben nem lett volna időm kimásolni a kimenetet. Láthatjuk tehát, hogy az `exit` parancsra a szkriptblokk tényleg lefutott.

A másik legyen egy általam kitalált esemény! Az eseményeket forrásuk, származási helyük alapján különböztetjük meg, így az általam kitalált esemény forrása szintén általam kitalált lesz: `Saját.Esemény`:

```
[8] PS C:\> Register-EngineEvent -SourceIdentifier Saját.Esemény -Action {write
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
-host "Most bekövetkezett a saját eseményem!"}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
2               Saját.Esemény   NotStarted False


[9] PS C:\> New-Event Saját.Esemény


ComputerName     :
RunspaceId       : 01526b1c-155b-408d-aa7c-23fe54ae22f3
EventIdentifier  : 3
Sender           :
SourceEventArgs  :
SourceArgs       : {}
SourceIdentifier : Saját.Esemény
TimeGenerated    : 2010. 02. 24. 22:24:06
MessageData      :



Most bekövetkezett a saját eseményem!
```

Lekérdezni az eseményfigyeléseket a `Get-EventSubscriber` cmdlettel lehet:

```
[45] PS C:\> Get-EventSubscriber


SubscriptionId   : 3
SourceObject     :
EventName        :
SourceIdentifier : Saját.Esemény
Action           : System.Management.Automation.PSEventJob
HandlerDelegate  :
SupportEvent     : False
ForwardEvent     : False
```

Megszűntetni az `Unregister-Event` cmdlettel.

```
[46] PS C:\> Unregister-Event -SubscriptionId 3
```

Az esemény-előfizetések valójában PowerShell Job-okként futnak:

```
[1] PS C:\> Register-EngineEvent -SourceIdentifier Saját.Esemény -Action {"Most
 bekövetkezett a saját eseményem!"}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
1               Saját.Esemény   NotStarted False


[2] PS C:\> Get-Job
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
1               Saját.Esemény   NotStarted False
```

Látható, hogy létrejött egy Job. Most „bekövetkeztetem” az eseményt:

```
[3] PS C:\> New-Event -SourceIdentifier saját.esemény


ComputerName     :
RunspaceId       : 55d7819c-650b-4fb2-9553-ce395f0ab348
EventIdentifier  : 1
Sender           :
SourceEventArgs  :
SourceArgs       : {}
SourceIdentifier : saját.esemény
TimeGenerated    : 2010. 02. 25. 21:55:56
MessageData      :
```

A fenti eseményregisztráció `-Action` részében most nem `Write-Host`-tal jeleztem az esemény bekövetkeztét, hanem csak „simán” kiküldtem a szöveget az Outputra, de mivel háttérben futó folyamatként futott ez le, ezért nem látok a konzolon semmit. Ahhoz, hogy ezt meglássam, a kimenetet el kell kérnem a Jobtól:

```
[5] PS C:\> Receive-Job -Id 1
Most bekövetkezett a saját eseményem!
```

Ha eltávolítom az eseményregisztrációt, attól még a befejezett Job benne marad a rendszerben, így érdemes lehet azt eltávolítani:

```
[6] PS C:\> Unregister-Event -SourceIdentifier saját.esemény
[7] PS C:\> Get-Job

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
1               Saját.Esemény   Stopped    False
[8] PS C:\> Remove-Job -Id 1
```

Van még további lehetőség: távoli gépen generálódó események helyi gépre továbbítása. Első lépésként regisztrálni kell az eseményt a távoli gépen egy `PSSession` kapcsolaton keresztül:

```
[47] PS C:\> $s = New-PSSession -ComputerName member
[48] PS C:\> Invoke-Command -Session $s -ScriptBlock {Register-EngineEvent -Sou
rceIdentifier saját.event -Forward}
```

A helyi gépen is regisztrálni kell az eseményt, hogy a továbbított esemény bekövetkeztéről értesülhessünk:

```
[49] PS C:\> Register-EngineEvent -SourceIdentifier saját.event -Action {write-
host "Valami történt!"}
```

```
WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
6               saját.event     NotStarted False
```

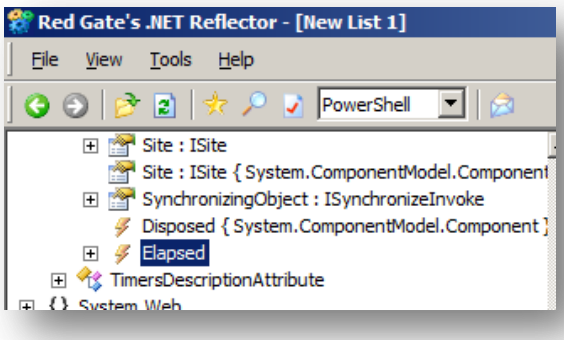
Ezután már jöhet a távoli gépen az esemény. Sajnos az eseménykezelésnek is „scope”-ja van, azaz nem úgy általában kell generálni az eseményt, hanem abban a scope-ban, ahol az eseményre előfizettünk, jelen esetben az `$s` változóban tárolt PSSession-ben:

```
[51] PS C:\> Invoke-Command -Session $s -ScriptBlock {new-event saját.event}


PSComputerName      : member
RunspaceId          : 976df52b-7aa1-4ad5-adf0-75e0ec51e98b
PSShowComputerName  : True
ComputerName        :
EventIdentifier     : 2
Sender              :
SourceEventArgs     :
SourceArgs          : {}
SourceIdentifier    : saját.event
TimeGenerated       : 2010. 02. 25. 10:13:36
MessageData         :



Valami történt!
```

Látható, hogy meg is érkezett a reagálás a gépemen a távoli eseményre. Megjegyzendő még, hogy a továbbított eseményeknél nem lehet szkriptet definiálni, tehát választanunk kell, hogy vagy továbbítjuk az eseményt és a másik gépen reagálunk rá, vagy pedig nem továbbítjuk és akkor tudunk a távoli gépen reagálni.

2.18.2 ObjectEvent kezelése

A következő eseménytípus a .NET osztályok és objektumok által generált események. Ha belenézünk a Reflector programmal a .NET keretrendszerbe, akkor itt-ott találunk esemény típusú tagjellemzőket:



76. ábra Az esemény-tagjellemzők villámmal vannak jelezve

A könyv tanfolyami felhasz ett!

Ezt persze PowerShellből is fel tudjuk deríteni, a fenti példában szereplő `System.Timers.Timer` osztály esetében például így:

```
[4] PS C:\> $timer = new-object System.Timers.Timer
[5] PS C:\> Get-Member -InputObject $timer -MemberType event


   TypeName: System.Timers.Timer

Name     MemberType Definition
----     ---------- ----------
Disposed Event      System.EventHandler Disposed(System.Object, System.Even...
Elapsed  Event      System.Timers.ElapsedEventHandler Elapsed(System.Object...
```

Ha jól beállítjuk a `$timer` változóban tárolt objektumot, azaz megadjuk, hogy mennyi időt számláljon vissza és elindítjuk a visszaszámlálást, akkor az idő elteltekor ez az esemény fog bekövetkezni, amire PowerShell segítségével is tudunk reagálni. Első lépés tehát a számláló beállítása:

```
[7] PS C:\> $timer | Format-List *


AutoReset           : True
Enabled             : False
Interval            : 100
Site                :
SynchronizingObject :
Container           :



[8] PS C:\> $timer.Interval = 1000
[9] PS C:\> $timer.Enabled = $true
```

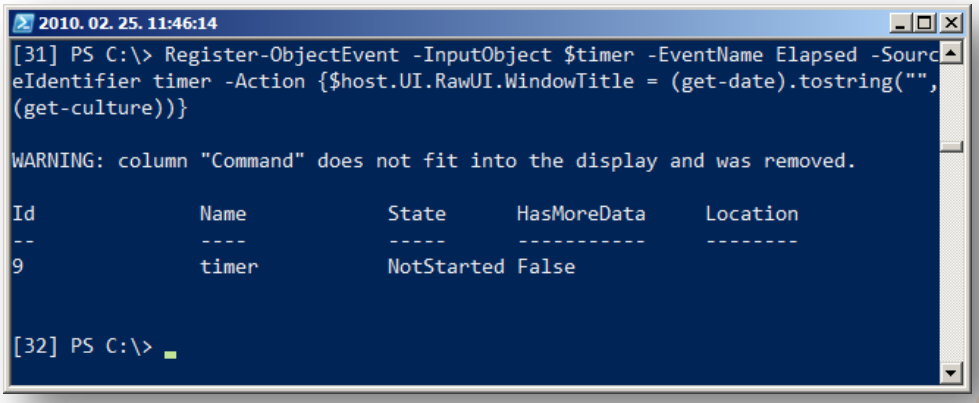
Megnéztem először, hogy milyen tulajdonságai is vannak ennek a `$timer`-nek, utána beállítottam 1000 ms-os visszaszámlálást, majd bekapcsoltam a számlálót. Regisztráljuk az eseménykezelőt is erre a `Register-ObjectEvent` cmdlettel:

```
[31] PS C:\> Register-ObjectEvent -InputObject $timer -EventName Elapsed -Sourc
eIdentifier timer -Action {$host.UI.RawUI.WindowTitle = (get-date).tostring("",
(get-culture))}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
9               timer           NotStarted False
```

Az `-Action` részben az aktuális időt teszem be a PowerShell ablak fejlécébe, szépen a területi beállításoknak megfelelő formátumban. Az eredmény ilyen szép (az ablak fejlécét kell figyelni):

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



77. ábra Az idő kiíratása az ablakfejlécbe eseményvezérelten

Az eseményfigyelést ugyanúgy az `Unregister-Event` cmdlettel tudjuk megszűntetni.

A másik gyakori lekövetendő esemény a fájlok születése, megszűnése és módosulása. Erre a .NET keretrendszer `System.IO.FileSystemWatcher` osztálya áll rendelkezésünkre, amit hasonlóan kell kezelni, felparaméterezni, mint az előbb látott időzítőt:

```
[32] PS C:\> $watcher = New-Object System.IO.FileSystemWatcher
[33] PS C:\> Get-Member -InputObject $watcher -MemberType event, property


   TypeName: System.IO.FileSystemWatcher

Name                  MemberType Definition
----                  ---------- ----------
Changed               Event      System.IO.FileSystemEventHandler Changed(S...
Created               Event      System.IO.FileSystemEventHandler Created(S...
Deleted               Event      System.IO.FileSystemEventHandler Deleted(S...
Disposed              Event      System.EventHandler Disposed(System.Object...
Error                 Event      System.IO.ErrorEventHandler Error(System.O...
Renamed               Event      System.IO.RenamedEventHandler Renamed(Syst...
Container             Property   System.ComponentModel.IContainer Container...
EnableRaisingEvents   Property   System.Boolean EnableRaisingEvents {get;set;}
Filter                Property   System.String Filter {get;set;}
IncludeSubdirectories Property   System.Boolean IncludeSubdirectories {get;...
InternalBufferSize    Property   System.Int32 InternalBufferSize {get;set;}
NotifyFilter          Property   System.IO.NotifyFilters NotifyFilter {get;...
Path                  Property   System.String Path {get;set;}
Site                  Property   System.ComponentModel.ISite Site {get;set;}
SynchronizingObject   Property   System.ComponentModel.ISynchronizeInvoke S...

[35] PS C:\> $watcher | Format-List


NotifyFilter          : FileName, DirectoryName, LastWrite
EnableRaisingEvents   : False
Filter                : *.*
IncludeSubdirectories : False
InternalBufferSize    : 8192
Path                  :
Site                  :
SynchronizingObject   :
```

nem engedélyezett!

```
Container             :
```

Látható, hogy milyen paraméterekből épül fel egy ilyen fájlfigyelő: ki kell választani, hogy milyen eseményt figyelünk; lehet a fájlnévre vonatkozó szűrőt beállítani; alkönyvtárakban is figyeljen-e; elérési utat meg lehet adni. Nézzük, hogy milyen értesítések létezhetnek egyáltalán:

```
[41] PS C:\> [system.enum]::getnames([system.io.notifyfilters])
FileName
DirectoryName
Attributes
Size
LastWrite
LastAccess
CreationTime
Security
```

Ezt összevetve a lehetséges eseményekkel, érdemes lehet például a következőket beállítani figyelésre:

```
[71] PS C:\> $watcher.NotifyFilter = "FileName", "DirectoryName", "LastWrite",
"Attributes", "Security"
```

A `FileName`, `DirectoryName` figyelésével lehetővé válik a `Rename`, `Create` és `Delete` esemény észlelése. A `LastWrite` figyelésével észrevesszük a tartalmi módosítást, azaz a `Change` eseményt. Ugyancsak `Change` esemény következik be, ha a fájl hozzáférését és ha attribútumait módosítjuk. A `FileName` és `DirectoryName` figyelése még `Change` eseményt is generál, ha a fájl átmozgatjuk.

Egyéb paraméterek beállítása:

```
[72] PS C:\> $watcher.IncludeSubdirectories = $true
[73] PS C:\> $watcher.Path = "C:\munka"
```

Miután itt több fajta esemény is bekövetkeztethet és egy esemény-előfizetésnél csak egy eseményt lehet megadni, ezért készítettem egy kifejezést, ami az összes, a [33]-as sornál látható eseményre regisztrál egy-egy előfizetést:

```
[74] PS C:\> $watcher | Get-Member -MemberType event | ForEach-Object {Register
-ObjectEvent -InputObject $watcher -EventName $_.name -SourceIdentifier "watche
r.$($_.name)" -Action {$event.timegenerated, $eventargs | out-host}}

WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
44              watcher.Changed NotStarted False
45              watcher.Created NotStarted False
46              watcher.Deleted NotStarted False
47              watcher.Disp... NotStarted False
48              watcher.Error   NotStarted False
49              watcher.Renamed NotStarted False
```

Ez a kifejezést tehát kiszedi a `$watcher` tagjellemzői közül csak az esemény típusúakat, majd ezek mindegyikére beregisztrál egy `ObjectEvent` eseménykezelést. Az `-Action` résznél látható, hogy létrejönnek ilyen esetben újabb automatikus változók, ezek közül a legfontosabbakat kiírom a konzolra.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Akkor most kicsit stimulálom a megfigyelt könyvtáram fájljait, nézzük a kimenetet:

```
2010. február 25. 16:41:23

ChangeType : Changed
FullPath   : C:\munka\Ez egy fájl.txt
Name       : Ez egy fájl.txt


2010. február 25. 16:41:37

ChangeType : Deleted
FullPath   : C:\munka\Ez egy fájl.txt
Name       : Ez egy fájl.txt


2010. február 25. 16:41:59

OldFullPath : C:\munka\aa2.txt
OldName     : aa2.txt
ChangeType  : Renamed
FullPath    : C:\munka\újnév.txt
Name        : újnév.txt


2010. február 25. 16:41:59

ChangeType : Changed
FullPath   : C:\munka\újnév.txt
Name       : újnév.txt


2010. február 25. 16:42:27

ChangeType : Created
FullPath   : C:\munka\New Text Document (4).txt
Name       : New Text Document (4).txt
```

A fenti listában levő értesítések rendre a következő tevékenységem hatására generálódtak:

- kivettem egy meglevő fájl Read-Only attribútumát
- töröltem ezt a fájlt
- átneveztem egy fájlt
- beleírtam a fájlba
- létrehoztam egy fájlt

Az előfizetések törlése ugyanúgy történik, mint az `EngineEvent`-ek esetében.

2.18.3 WMIEvent kezelése

A harmadik típusú események, amelyeket a PowerShell képes kezelni és reagálni rájuk, a WMI események. Ilyen eseményekkel a WMI objektumok létrejöttére, megszűnésére és változására lehet reagálni. Ilyen lehet például az IP-címzéssel, videó beállításokkal, megosztások és processzek megszűnésével vagy létrejöttével, és szolgáltatások leállásával vagy indításával kapcsolatot események.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A WMI események alapját egy lekérdezés adja, ami kiszűri a lehetséges több ezer WMI objektum közül azokat, amelyek valamilyen jellegű változására kíváncsiak vagyunk. Ennek a lekérdezésnek az általános szerkezete:

```
SELECT <Properties> FROM <EventClass> WITHIN <másodperc> WHERE TargetInstance
    ISA '<WmiClass>' AND '<OtherCriteria>'…
```

A fenti kifejezésben a lehetséges `EventClass` kategóriák a következők lehetnek:

- `__InstanceCreationEvent` – akkor következik be, ha egy WMI osztály egy példánya létrejön
- `__InstanceDeletionEvent` – akkor következik be, ha egy WMI osztály egy példánya megszűnik
- `__InstanceModificationEvent` – akkor következik be, ha egy WMI osztály egy példányának valamely tulajdonsága megváltozik
- `__InstanceOperationEvent` – akkor következik be, ha a fentiek közül bármelyik bekövetkezik

(Az eseményosztályok nevében fontos a két aláhúzás az elején!)

Ezek alapján első nekifutásra összeraktam egy olyan lekérdezést, ami a szolgáltatások körében bekövetkező bármilyen változásra reagál:

```
[9] PS C:\> $wmiquery = "select * from __InstanceModificationEvent within 1 whe
re TargetInstance ISA 'Win32_Service'"
```

Ezzel regisztrálom is a PowerShell eseményfigyelőt, méghozzá olyan tevékenységgel, hogy adja át az általa érzékelt eseménnyel kapcsolatos paramétereket egy globális változóba, és jelezze a képernyőn, hogy valami történt:

```
[42] PS C:\> Register-WmiEvent -Query $wmiquery -Action {$global:wmievent = $ev
entargs; write-host "Esemény történt!"}
```

Ezután leállítom az egyik szolgáltatást, nézzük mi történik, illetve mi lesz a globális `$wmievent` változóm tartalma:

```
Esemény történt!

[46] PS C:\> $wmievent

NewEvent                                  Context
--------                                  -------
System.Management.ManagementBaseObject    {}

[47] PS C:\> $wmievent.NewEvent


__GENUS             : 2
__CLASS             : __InstanceModificationEvent
__SUPERCLASS        : __InstanceOperationEvent
__DYNASTY           : __SystemClass
__RELPATH           :
__PROPERTY_COUNT    : 4
__DERIVATION        : {__InstanceOperationEvent, __Event, __IndicationRelated,
                      __SystemClass}
__SERVER            : DC
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
__NAMESPACE          : //./root/CIMV2
__PATH               :
PreviousInstance     : System.Management.ManagementBaseObject
SECURITY_DESCRIPTOR  :
TargetInstance       : System.Management.ManagementBaseObject
TIME_CREATED         : 129116550251633750
```

Ennek a `$wmievent` objektumnak minket leginkább érdeklő tulajdonságai a `TargetInstance`, ami azt adja meg, hogy a változás után mi lett az eseményt kiváltó WMI példány, és a `PreviousInstance`, ami azt adja meg, hogy mi volt a változás előtti WMI példány. Ezen két objektum tulajdonságainak összevetésével ki lehet mutatni, hogy pontosan mi is változott:

```
[48] PS C:\> $wmievent.NewEvent.TargetInstance


__GENUS                 : 2
__CLASS                 : Win32_Service
__SUPERCLASS            : Win32_BaseService
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Service.Name="msiserver"
__PROPERTY_COUNT        : 25
__DERIVATION            : {Win32_BaseService, CIM_Service, CIM_LogicalElement,
                           CIM_ManagedSystemElement}
__SERVER                : DC
__NAMESPACE             : root\CIMV2
__PATH                  : \\DC\root\CIMV2:Win32_Service.Name="msiserver"
AcceptPause             : False
AcceptStop              : False
Caption                 : Windows Installer
CheckPoint              : 0
CreationClassName       : Win32_Service
Description             : Adds, modifies, and removes applications provided as
                           a Windows Installer (*.msi) package. If this servic
                          e is disabled, any services that explicitly depend o
                          n it will fail to start.
DesktopInteract         : False
DisplayName             : Windows Installer
ErrorControl            : Normal
ExitCode                : 0
InstallDate             :
Name                    : msiserver
PathName                : C:\Windows\system32\msiexec.exe /V
ProcessId               : 0
ServiceSpecificExitCode : 0
ServiceType             : Own Process
Started                 : False
StartMode               : Manual
StartName               : LocalSystem
State                   : Stopped
Status                  : OK
SystemCreationClassName : Win32_ComputerSystem
SystemName              : DC
TagId                   : 0
WaitHint                : 0



[49] PS C:\> $wmievent.NewEvent.PreviousInstance
```

```
__GENUS                 : 2
__CLASS                 : Win32_Service
__SUPERCLASS            : Win32_BaseService
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Service.Name="msiserver"
__PROPERTY_COUNT        : 25
__DERIVATION            : {Win32_BaseService, CIM_Service, CIM_LogicalElement,
                           CIM_ManagedSystemElement}
__SERVER                : DC
__NAMESPACE             : root\CIMV2
__PATH                  : \\DC\root\CIMV2:Win32_Service.Name="msiserver"
AcceptPause             : False
AcceptStop              : True
Caption                 : Windows Installer
CheckPoint              : 0
CreationClassName       : Win32_Service
Description             : Adds, modifies, and removes applications provided as
                           a Windows Installer (*.msi) package. If this servic
                          e is disabled, any services that explicitly depend o
                          n it will fail to start.
DesktopInteract         : False
DisplayName             : Windows Installer
ErrorControl            : Normal
ExitCode                : 0
InstallDate             :
Name                    : msiserver
PathName                : C:\Windows\system32\msiexec.exe /V
ProcessId               : 1632
ServiceSpecificExitCode : 0
ServiceType             : Own Process
Started                 : True
StartMode               : Manual
StartName               : LocalSystem
State                   : Running
Status                  : OK
SystemCreationClassName : Win32_ComputerSystem
SystemName              : DC
TagId                   : 0
WaitHint                : 0
```

Sajnos ezt így szemre áttekinteni elég nehéz, ezért készítettem egy objektumtulajdonság összehasonlító `Compare-ObjectProps` függvényt. Ez paraméterként két objektumot vár, meg lehet adni összehasonlító üzemmódot (`eq`: csak az egyforma tulajdonságokat jeleníti meg, `ne`: eltérő tulajdonságokat jeleníti meg, `all`: minden tulajdonságot megjelenít), és `Include` paraméterként meg lehet adni, hogy mik azok a tulajdonságok, amiket mindenképpen szeretnénk látni:

```
function Compare-Objectprops ($o1,$o2,$mode = "all",[string[]] $include)
{
     $props = @($o1 | Get-Member -MemberType Properties | % {$_.name})
     $o2 | Get-Member -MemberType Properties |
           % {$_.name} | ?{$props -notcontains $_} | %{$props += $_}

     $mat = @{}
     $props | %{$mat.$_ = @($o1.$_,$o2.$_)}
     foreach($key in $mat.keys)
     {
         if($include -contains $key)
```

könyv tanfolyami

```
        {
            New-Object -TypeName psobject -Property @{
                        name = $key;
                        Object1 = $mat.$key[0];
                        Object2 = $mat.$key[1]}
        }
        else
           {
            switch($mode)
               {
                 "eq" {if($mat.$key[0] -eq $mat.$key[1]){
                              New-Object -TypeName psobject -Property @{
                                    name = $key;
                                    Object1 = $mat.$key[0];
                                    Object2 = $mat.$key[1]}
                              break}}
                  "ne" {if($mat.$key[0] -ne $mat.$key[1]){
                              New-Object -TypeName psobject -Property @{
                                    name = $key;
                                    Object1 = $mat.$key[0];
                                    Object2 = $mat.$key[1]}
                              break}}
                  "all" {New-Object -TypeName psobject -Property @{
                              name = $key;
                              Object1 = $mat.$key[0];
                              Object2 = $mat.$key[1]}}
               }
            }
     }
}
```

Nézzük ezzel mit láthatunk:

```
[54] PS C:\> Compare-Objectprops $wmievent.NewEvent.PreviousInstance $wmievent.
NewEvent.TargetInstance -mode ne -include displayname

name                        Object2                   Object1
----                        -------                   -------
DisplayName                 Windows Installer         Windows Installer
AcceptStop                  False                     True
State                       Stopped                   Running
ProcessId                   0                         3140
Started                     False                     True
__DERIVATION                {Win32_BaseService, CI... {Win32_BaseService, CI...
```

Látható, hogy a `Windows Installer` szolgáltatást állítottam le, ez okozta az eseményt, hiszen a korábbi állapot (`Object1` oszlop) volt a „`Running`”, most meg „`Stopped`”.

Az előbb látott objektumok (`TargetInstance` és `PreviousInstance`) a WMI eseménylekérdezésben is felhasználható. Azaz ha én csak olyan jellegű szolgáltatás-változásokra vagyok kíváncsi, amikor a szolgáltatás leállt, és arra nem vagyok kíváncsi, amikor egy szolgáltatás elindult, akkor a lekérdezést így lehet pontosítani:

```
[62] PS C:\> $wmiquery = "select * from   InstanceModificationEvent within 1 wh
ere TargetInstance ISA 'Win32_Service' AND targetinstance.State = 'Stopped'"
[63] PS C:\> Register-WmiEvent -Query $wmiquery -Action {$global:wmievent = $ev
entargs; write-host "Esemény történt!"}
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
WARNING: column "Command" does not fit into the display and was removed.

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
5               c2fbcf0d-af2... NotStarted False


[64] PS C:\> Esemény történt!
Compare-Objectprops $wmievent.NewEvent.PreviousInstance $wmievent.NewEvent.Targ
etInstance -mode ne -include displayname | ft name, object1, object2

name                        Object1                  Object2
----                        -------                  -------
DisplayName                 Windows Installer        Windows Installer
AcceptStop                  True                     False
State                       Running                  Stopped
Status                      Degraded                 OK
ProcessId                   3608                     0
Started                     True                     False
__DERIVATION                {Win32_BaseService, CI... {Win32_BaseService, CI...
```

A fenti példa úgy született, hogy töröltem a korábbi eseményfigyelést, és a módosított WMI lekérdezés létrehozása és az eseményfigyelés regisztrálása után elindítottam a korábban leállított Windows Installer szolgáltatást. Erre nem történt semmi, ellenben amikor újra leállítottam, akkor megkaptam a fenti eseményjelzést.

A könyv tanfolyami felhasználása nem engedélyezett!

3.Függelékek

A könyv tanfolyami felhasználása nem engedélyezett!

3.1 OOP alapok

Az előzőekben már találkozhattunk azzal a kijelentéssel, hogy a PowerShell egy objektumorientált shell. Első hallásra talán furcsának tűnhet ez a szókapcsolat, már csak azért is, mert ilyen egyszerűen nem létezett korábban; a PowerShell az első, és pillanatnyilag az egyetlen ilyen tulajdonságú parancsfeldolgozó. A következőkben áttekintjük, mit is jelent, és milyen következményekkel jár az objektumorientált felépítés.

Nézzük meg részletesebben az objektumorientált programozáshoz kapcsolódó legfontosabb fogalmakat, amelyek pontos ismeretére a későbbiekben feltétlenül szükségünk lesz.

3.1.1 Osztály (típus)

Az osztályok az objektumok tervrajzai; tartalmazzák a különféle funkciókat megvalósító programkódot és deklarálják az objektumok adatszerkezeteit. Az osztályok és objektumok viszonya megegyezik a változótípus és konkrét változó viszonyával. Ahogyan a változóhoz tartozó típus meghatározza a változó lehetséges állapotait, és rajta végezhető műveleteket, úgy határozza meg az objektum osztálya a benne tárolható adatokat (tulajdonságokat), és az általa elvégezhető műveleteket (metódusokat). Az osztály definiálja tehát a majdani objektumok adatait, és azokat a műveleteket (eljárások és függvények), amelyek elvégzésére az osztály alapján létrehozott objektum képes.

3.1.2 Példány (objektum)

Az adott osztály (tervrajz) alapján létrehozott objektumpéldányok képesek az osztályban definiált változóknak megfelelő információ tárolására, és az osztály által meghatározott műveletek (eljárások és függvények) végrehajtására.

Minden objektum egy meghatározott memóriaterületet foglal el, itt tárolja adatait. Az adatok pillanatnyi értékét az objektum állapotának nevezzük. Két objektum azonban akkor sem azonos, ha állapotuk megegyezik (vagyis valamennyi adatuk rendre egyenlő), mivel az objektumot nem az állapota, hanem az általa elfoglalt memóriaterület kezdőcíme azonosítja. Programjainkban az objektumok tehát memóriacímként (objektumreferencia) jelennek meg.

3.1.3 Példányosítás

Az objektum osztálya tehát az objektum viselkedését, képességeit meghatározó kódból, és adatainak típusdefinícióiból áll. Példányosításnak azt a műveletet nevezzük, amelynek során a szükséges memóriaterületet lefoglalva, egy osztály alapján objektumot hozunk létre. Az objektum létrehozásakor elvégzendő műveleteket az adott osztályban definiált speciális függvény, az osztály konstruktora határozza meg. Az osztály alapján létrehozott objektum adatszegmense a konstruktornak átadott paraméterlista, vagy más adatforrás alapján a példányosítás során, vagyis az objektum létrehozása közben töltődik fel. Programunk szempontjából a példányosítás „végterméke” egy memóriacímet tartalmazó változó (objektumreferencia, mutató, pointer); a következőkben ennek segítségével érhetjük el az adott példány adatait, és hívhatjuk meg az osztályban definiált eljárásokat és függvényeket.

A könyv tanfolyami felhasználása nem engedélyezett!

Ha egyszerre több azonos osztályú objektumot is használunk, az objektumok kódja csak egyetlen példányban kerül a memóriába (hiszen ez minden azonos osztályba tartozó objektum esetén feltétlenül egyforma), de természetesen az adatszegmens minden objektum esetében önállóan létezik. Minden objektum tartalmaz egy referenciát (`this`), amely az osztályát azonosítja, ennek felhasználásával hívhatja meg az osztályban tárolt metódusokat.

3.1.4 Metódusok és változók

Azokat a változókat, amelyek objektum-példányonként külön memóriaterületre kerülnek példányváltozóknak, a példányváltozókat felhasználó metódusokat pedig példánymetódusoknak nevezzük. Ha meghívjuk egy objektum példánymetódusát, akkor a metódus kódja az osztályból származik ugyan, az elvégzett műveletek viszont általában az objektum példány saját adatait fogják felhasználni, vagyis megváltoztatják az objektum állapotát.

3.1.5 Példányváltozó, példánymetódus

Azokat a változókat, amelyek objektum-példányonként külön memóriaterületre kerülnek példányváltozóknak, a példányváltozókat felhasználó metódusokat pedig példánymetódusoknak nevezzük. Ha meghívjuk egy objektum példánymetódusát, akkor a metódus kódja az osztályból származik ugyan, az elvégzett műveletek viszont általában az objektum példány saját adatait fogják felhasználni, vagyis megváltoztatják az objektum állapotát.

3.1.6 Statikus változó, statikus metódus

Bizonyos változók nem egy konkrét objektum-példányra, hanem az egész osztályra jellemzők. Az ilyen közös, az egész osztályra jellemző változókat osztályváltozóknak, vagy statikus változóknak nevezzük. A statikus változók értéke minden példány esetén megegyezik, ezért csak egy helyen kell tárolni, minden példány ezt az egy memóriaterületet éri el.

Osztálymetódusnak, vagy statikus metódusnak nevezzük azokat a metódusokat, amelyek objektum-példány nélkül, közvetlenül az osztályra való hivatkozással futtathatók. A statikus metódusok csak a statikus változókat érik el a példányváltozókat nem. Tehát a példányváltozókat csak a példánymetódusok érik el, míg a statikus változókat a statikus- és példánymetódusok egyaránt használhatják.

3.1.7 Változók elrejtése

Az objektumok egyik fontos tulajdonsága, hogy a külvilág számára csak azokat a metódusokat (és ritkán adatokat) teszi elérhetővé, amelyek feltétlenül szükségesek az objektum használatához. Az objektum tehát egy jól meghatározott interfészen keresztül érhető el, amelyet természetesen az osztály készítői igyekeznek a lehető legkisebbre készíteni.

Az osztályokban definiált változók általában csak metódusokon keresztül, vagyis ellenőrzött módon érhetők el. Ezzel megakadályozható az objektum állapotának „elrontása”, azaz minden adatmezőnek csak olyan érték adható, amelynek tárolására azt a programozó szánta. Előnyös továbbá az is, hogy ilyen módon az

A könyv tanfolyami felhasználása nem engedélyezett!

osztály teljes belső adatszerkezete lecserélhető anélkül, hogy az osztályt használó komponenseknek erről tudniuk kellene.

3.1.8 Overloaded metódusok

A metódus szignatúrája a nevén kívül tartalmazza paramétereinek számát és az egyes paraméterek típusát is. A metódusokat a fordítóprogram nem pusztán a nevük, hanem a szignatúrájuk alapján azonosítja, vagyis egy osztályon belül lehet több azonos nevű, de eltérő paraméterlistájú metódus is. A metódus meghívásakor a fordító a név és az aktuálisan átadott paraméterek alapján választja ki azt a metódust, amelyik le fog futni. Ilyen metódusokkal igen gyakran fogunk találkozni a .NET keretrendszer osztályaiban is. A jelenséget „method overloading[13]”-nak, azaz metódus-újratöltésnek, -felültöltésnek, -felülbírálásnak nevezzük.

3.1.9 Öröklődés

Az öröklődés két osztály között értelmezett kapcsolat, azt fejezik ki, hogy az egyik osztály (az utód) specializált változata a másiknak, az ősosztálynak. A specializálás során egy már meglévő objektum leírásához, tervrajzához, új, egyedi jellemzőket és képességeket adunk hozzá. A specializált osztály tehát örökli az ősosztály adatait és metódusait, de az öröklés során ezekhez újabbakat is hozzáadhatunk, illetve módosíthatjuk a meglévőket.

Egy osztály örökítésekor három lehetőséget használhatunk:

- Új változókat adhatunk hozzá az ősosztályhoz.
- Új metódusokat adhatunk hozzá az ősosztályhoz.
- Felülírhatjuk az ősosztály metódusait.[14]

[13] Sajnos erre a kifejezésre nincsen jó magyar szó, a szokásosan használt „túlterhelt metódus” kifejezés inkább valamiféle elmegyógyászati diagnózisra emlékeztet.

[14] Az ősosztály adatait nem lehet felülírni.

A könyv tanfolyami felhasználása nem engedélyezett!

A könyv tanfolyami felhasználása nem engedélyezett! © Soós Tibor – soost@ IQJB.hu

3.2 Mi is az a .NET keretrendszer

A .NET már sok éve velünk van, és a három betű valószínűleg mindenkinek ismerősen cseng, de talán nem fölösleges röviden áttekinteni, hogy pontosan miről is van szó, és miért jó nekünk ez az egész.

A .NET Framework a Windows operációs rendszerekbe egyre szorosabban integrálódó komponens, amely az alkalmazások új generációjának fejlesztését és futtatását teszi lehetővé. A .NET egyszerűen az infrastruktúra szintjére emeli számos általános feladat megoldását, amelyekkel korábban minden programozónak magának kellett jól-rosszul megküzdenie. Aki ismeri és betartja a szabályokat, az használhatja az infrastruktúrát.

A .NET alapú programok már nem közvetlenül veszik igénybe az operációs rendszer szolgáltatásait és nem is közvetlenül a processzor futtatja őket; egy újabb absztrakciós réteg (ez maga a .NET keretrendszer) kapcsolódott be a játékba. Ez természetesen némi sebességcsökkenést okoz a programfuttatás szintjén, de igen jelentős sebességnövekedéssel jár, ha a fejlesztésre fordított időt vesszük figyelembe.

A keretrendszer alapvetően két komponensből áll, ezek a *CLR* (*Common Language Runtime*) és a mögötte álló osztályhierarchia, a *Class Library*.

3.2.1 Futtatókörnyezet (Common Language Runtime, CLR)

A .NET keretrendszer felhasználásával készített programok, egy szoftveres virtuális környezetben (CLR) futnak, amely biztosítja a programfuttatáshoz szükséges feltételeket. A CLR egységes futási környezetet biztosít a .NET alapú programok számára, függetlenül attól, hogy azokat milyen programnyelven készítették. Elvégzi a memóriakezelést és biztosítja a programfutáshoz szükséges egyéb alapvető szolgáltatásokat, kezeli a programszálakat, biztonságos futási környezetet ad a programkódnak, és megakadályozza, hogy a futtatott programok bármiféle általa szabálytalannak ítélt műveletet végezzenek. A CLR fennhatósága alatt futó programokat felügyelt kódnak (managed code) nevezzük.

A CLR biztosítja a programnyelvek közötti teljes együttműködést, így lehetővé válik a különböző nyelveken megírt komponensek problémamentes együttműködése is. A CLR szigorú típusrendszerre épül, amelyhez minden CLR-kompatibilis programnyelvnek alkalmazkodnia kell. Ez azt jelenti, hogy az adott nyelv minden elemének (típusok, struktúrák, elemi adattípusok) a CLR által ismert típusokká konvertálhatónak kell lennie. További feltétel, hogy a fordítóprogramoknak a kódban lévő típusokat és hivatkozásokat leíró metaadatokat kell elhelyezniük a lefordított állományokban. A CLR ezek felhasználásával felügyeli a folyamatokat, megkeresi és betölti a megfelelő osztályokat, elhelyezi az objektum-példányokat a memóriában, stb.

A CLR minden erőforrást az adott folyamat számára létrehozott felügyelt heap-en (halom) helyez el. A felügyelt heap hasonló a hagyományos programnyelvek által használt heap-hez, de az itt létrehozott objektumokat nem a programnak kell megszüntetnie, a memória felszabadítása automatikusan történik, ha az adott objektumra már nincs többé szükség. A .NET egyik fontos szolgáltatása a szemétgyűjtő (Garbage Collector, GC), amely képes a hivatkozás nélkül maradt objektumok felkutatására, és az általuk lefoglalt memóriaterület felszabadítására.

3.2.2 Class Library (osztálykönyvtár)

A Class Library egy több ezer(!) osztályból álló gyűjtemény, amelyek segítségével szinte bármilyen feladatot megoldhatunk, lehetővé teszi parancssori, grafikus, vagy webes felületet, hálózati és biztonsági

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

szolgáltatásokat használó alkalmazások fejlesztését. Segítségével gyakorlatilag a Windows rendszerek valamennyi szolgáltatása a korábbinál lényegesen egyszerűbb formában elérhetővé válik. A Class Library lehetővé teszi szinte az összes, eddig csak a Win32 API segítségével megvalósítható szolgáltatás használatát, és még több ezer más feladatra is megoldást ad. Megtalálhatjuk benne a klasszikus algoritmusok (rendezések, keresések, stb.) megvalósítását, valamint rengeteg gyakori feladat szinte kész megoldását is.

A Class Library-t természetesen bármely .NET-képes programnyelvből használhatjuk, a nyelvek közötti különbség így tulajdonképpen szinte jelentéktelen szintaktikai különbséggé válik, a lényeg, az osztálykönyvtár azonos, bármelyik nyelvet (vagy akár a PowerShellt) is használjuk.

3.2.3 Programnyelvek

A CLR által nyújtott szolgáltatásokat a Visual Basic, a C#, a Visual C++, és a Jscript programnyelvekből, néhány külső gyártó által fejlesztett nyelvből (például Eiffel, Perl, COBOL stb.), illetve most már a PowerShellből is elérhetjük. A szükséges fordítóprogramok parancssori változatai megtalálhatók a .NET Framework SDK csomagban különféle egyéb eszközökkel együtt (debugger, disassembler stb.). Ilyen módon a .NET alapú programok készítéséhez nincs feltétlenül szükség integrált fejlesztői környezetre (például a Visual Studiora), elvben bármilyen alkalmazást elkészíthetünk a notepad.exe és a megfelelő fordítóprogram felhasználásával.

3.2.4 Programfordítás

A .NET fordítóprogramjai forráskódunkat egy köztes nyelvre (Intermediate Language, IL) fordítják. Az IL olyan processzorfüggetlen kód, amely igen hatékonyan fordítható tovább egy adott platform gépi kódjává. A compilerek kimenete tehát az IL-kód, amit a fordítóprogram exe, vagy dll állományba csomagol. Az elkészített exe állományok a felhasználó szempontjából a szokásos módon futtathatók, a háttérben azonban természetesen egészen más történik.

3.2.5 Programfuttatás

Mivel a processzor csak natív programok futtatására képes, az IL állományt végrehajtás előtt gépi kóddá kell fordítani. A gépi kód előállítását a futásidejű fordító (just-in-time compiler, JIT) végzi el, de csak akkor, ha az adott kód valóban le is fog futni. Az IL állományba irányuló minden metódushívás az első alkalommal meghívja a JIT-fordítót, ami gépi kóddá alakítja az adott metódust, ezt futtatja majd a processzor. Ha újra meghívjuk ugyanazt a metódust, már nincs szükség fordításra, közvetlenül az eltárolt gépi kódú változat fog lefutni. Így az egyes metódusokat csak egy alkalommal kell lefordítani, ami jelentősen gyorsíthatja a programok futását. További időmegtakarítást jelent, hogy a nem használt IL-kódot a JIT egyáltalán nem fordítja le. Amikor az alkalmazás leáll, a generált gépi kód automatikusan törlődik, újraindítás után tehát újra szükség van a metódusok futásidőben történő fordítására.

A JIT fordító természetesen minden támogatott processzor-architektúrára rendelkezésére áll, így a fordítóprogramok által készített IL-kód változtatás nélkül futtatható a különböző architektúrájú számítógépeken.[15]

[15] Persze csak akkor, ha nincs benne olyan API hívás, amely csak az adott platformon létezik.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

3.2.6 Assemblyk (kódkészletek)

A .NET-ben az assemblyk jelentik az alapvető telepítési, verziókövetési és biztonsági egységet. Az assembly több fizikai dll-t és esetleg más fájlokat egy önálló logikai egységben gyűjt össze. Az assemblykben lévő fizikai dll-ek a modulok, amelyekben minden esetben IL-kód található. Az assembly ezen felül önleíró adatokat (manifest) is tartalmaz, ami metaadatok formájában írja le az assemblyben található kódot és egyéb erőforrásokat.

Ha egy assemblyt több alkalmazás is használ, akkor azt egy speciális helyre, a .NET globális assembly gyorsítótárába (Global Assembly Cache, GAC) kell helyeznünk az SDK gacutil.exe programjának segítségével. Az osztott használatú assemblyk nyilvános kulcsú titkosítással biztosítják nevük egyediségét.

3.2.7 Érték- és referenciatípusok

A .NET keretrendszerben két alapvetően eltérő típussal találkozhatunk. Az érték típusok (ilyen például az int, a byte, vagy a char) a veremmemóriában jönnek létre, és helyük automatikusan felszabadul, amikor a deklaráló kódblokk véget ér, vagyis nincs szükségük a szemétgyűjtő szolgáltatásaira. Ha értékadásban használjuk őket, akkor nem egy rájuk mutató referencia, hanem tényleges értékük másolódik át. Minden egyes érték típusú változóhoz saját, önálló memóriaterület tartozik, vagyis az egyik változón végzett művelet egyetlen másik változó tartalmát sem változtathatja meg. Saját magunk is létrehozhatunk érték szerinti típusokat a „struct” kulcsszóval. Ilyen struktúra például az int típushoz tartozó `System.Int32` is, amely az osztályokhoz hasonlóan adatmezőket és metódusokat is tartalmazhat.

A referenciatípusok nem magát az adatot, hanem egy memóriacímre való hivatkozást tárolnak. Egy memóriacímre több hivatkozás is mutathat, tehát egy referencián (vagyis az általa mutatott objektumon) végzett műveletek a többi referenciát is érintik. A referenciatípusokat a `new` operátorral hozzuk létre, és azok a szemétgyűjtő (GC) által kezelt memóriába kerülnek (vagyis a felügyelt heap-re). Amikor az értékadásokban referenciatípusokat használunk, mindig csak a referencia értéke másolódik át, a valódi objektumhoz nem nyúlunk hozzá.

Valamennyi érték és referenciatípus elérhető `System.Object` osztályú referencia segítségével. Ha az érték szerinti típust referencián keresztül érjük el, akkor a fordító olyan kódot hoz létre, amely a típusnak megfelelő memóriát a heap-en foglalja le, és átmásolja ide a változó tartalmát a veremből. Ezt az eljárást „Boxing”-nak (dobozolásnak) nevezzük.

A könyv tanfolyami felhasználása nem engedélyezett!

3.3 A WMI áttekintése

Mielőtt belemerülnénk a WMI technológia részleteibe, érdemes tisztázni, hogy mire is jó ezek ismerete, hiszen a szoftvercégek jó pénzért megírják nekünk a kiváló felügyeleti alkalmazásokat, mi kattintgathatunk a remek felületen. A WMI technológia pedig legyen csak a szoftvercégek fejlesztőinek problémája, ők azért kapják a fizetésüket, hogy ilyesmit megtanuljanak és használjanak.

Nagyjából három olyan ok van, ami miatt mégsem kerülhetjük el a WMI megismerését:

Egyedi igények: – Bár a rendszerfelügyeleti szoftverek meglehetősen sokféle feladat megoldására képesek, speciális, egyedi funkciók mégis hiányozhatnak belőlük. Ilyenkor két dolgot tehetünk: kivárjuk azt a néhány évet, amíg a következő verzió megjelenik (talán abban benne lesz), vagy előkapjuk a Notepadot, és néhány sorban megírjuk magunk a hiányzó funkciót, WMI-t használó PowerShell script segítségével.

Költségek: A szoftvercégek kiváló termékei sajnos pénzbe kerülnek, méghozzá rendszer-felügyeleti szoftver esetében (nagyon) sok pénzbe. Egy Microsoft Systems Center Configuration Manager vagy IBM Tivoli megvásárlása néhány tucat gép esetében szinte reménytelen (és teljesen fölösleges is), de még pár száz gép esetén sem biztos, hogy kifizetődő. A néhány valóban szükséges funkció (például hardver és szoftverleltár, számítógépek monitorozása, riasztások, stb.) WMI segítségével egészen könnyen megvalósítható.

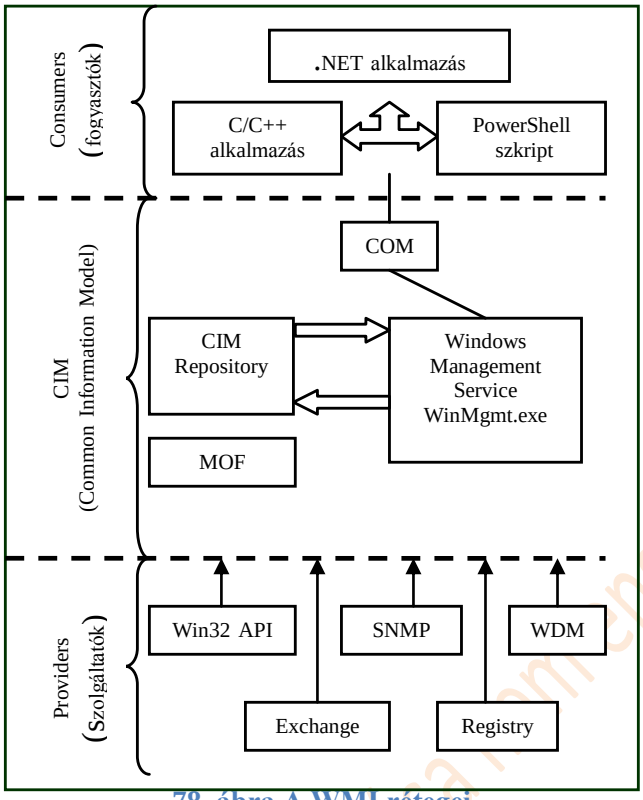
WMI-szűrők: A Windows Server 2003 újdonsága, hogy a csoportházirend (Group Policy) hatókörét WMI-szűrők segítségével módosíthatjuk. Mit is jelent ez? Tegyük fel, hogy csoportházirend segítségével szoftvert terítünk a hálózaton (300 különféle számítógép). A telepítendő szoftvernek viszont megvan az az eléggé el nem ítélhető tulajdonsága, hogy csak olyan gépen működik megfelelően, amelyben legalább 128 MB RAM van. Ha ezek a gépek nem egy külön OU-ban vannak (miért is lennének?), akkor vagy körbejárjuk a 300 gépet a manuális telepítéshez, vagy felkészülünk rá, hogy az automatikus telepítés valamilyen hibaüzenettel megszakad. A harmadik megoldás a WMI-szűrő használata. Ekkor az adott számítógép a szűrő hatására nyilatkozik a benne lévő memória mennyiségéről, és a telepítés csak megfelelő eredmény esetén indul el.

3.3.1 A WMI felépítése

A WMI a CIM (Common Information Model) segítségével jeleníti meg az operációs rendszer felügyelt objektumait. Felügyelt objektum lehet bármelyik szoftver, hardvereszköz, logikai vagy fizikai komponens, amelynek állapota a rendszergazdát érdekelheti.

A WMI három elkülöníthető rétegből áll, a következőkben ezekről lesz szó.

A könyv tanfolyami felhasználása nem engedélyezett!



78. ábra A WMI rétegei

Consumers (fogyasztók) – Fogyasztóknak nevezzük azokat az alkalmazásokat, amelyek felhasználják a WMI által biztosított adatokat. Fogyasztók lehetnek például szkriptek, Active X vezérlők, .NET alapú programok, vagy vállalati rendszerfelügyeleti eszközök (MOM, SMS, stb.). Valamennyi fogyasztó a Windows Management Service (WinMgmt.exe) által megvalósított COM csatolófelületen keresztül fér hozzá az adatokhoz. Az alkalmazásoknak természetesen nem kell tudniuk, hogy az egyes rendszerkomponensekre vonatkozó adatok valójában honnan és milyen módon származnak; nekik csak a COM csatolófelület által nyújtott lehetőségeket kell felhasználniuk.

CIM – A CIM rétegben találhatók a WMI központi komponensei. A CIM Repository tartalmazza azokat az osztálydefiníciókat, amelyekre a rendszer felügyelt objektumainak megjelenítéséhez szükség van, a Windows Management Service pedig a CIM Repository alapján továbbítja a providerektől kapott adatokat a fogyasztó alkalmazások felé. A MOF (Management Object Format) fájlok a CIM Repository bővítését teszik lehetővé. Ilyen fájlokat a WMI részeként kapott MOF compiler (mofcomp.exe) segítségével készíthetünk. Maga a CIM Repository is több ilyen módon lefordított .mof fájlból tevődik össze.

Providers (szolgáltatók) – a szolgáltatók feladata a felügyelt objektumokkal való közvetlen kommunikáció, azok saját API-jának felhasználásával. A különféle rendszerkomponens-csoportok adatainak lekérdezését önálló szolgáltatók végzik. A WMI csomag maga is számos szolgáltatót tartalmaz (a későbbi Windows verziók egyre többet), de természetesen sok alkalmazás hozza a saját szolgáltatóját, amelynek segítségével az alkalmazás adatai elérhetővé válnak WMI-n keresztül

Az adatok továbbítása a következő módon történik: az alkalmazás a COM felület használatával bármely felügyelt objektum tetszőleges adatára rákérdezhet. A WinMgmt.exe a CIM Repository adatainak felhasználásával meghatározza, hogy az adott információt melyik providertől, és milyen módon kell elkérnie. Ezután megszólítja a providert, az pedig lekérdezi az objektum megfelelő adatát. Ezt azután a WinMgmt.exe továbbítja a fogyasztó alkalmazás felé.

A könyv tanfolyami felhasználása nem engedélyezett!

Vagyis mondhatjuk azt, hogy a CIM Repository tulajdonképpen egy egységes nyilvántartás, amelynek segítségével a különböző providerek kezelése azonos módon történhet. Minden adatot, amelyet a WMI-n keresztül el szeretnénk érni, tartalmaznia kell a CIM Repository objektummodelljének, és regisztrálnunk kell a használni kívánt providereket is.

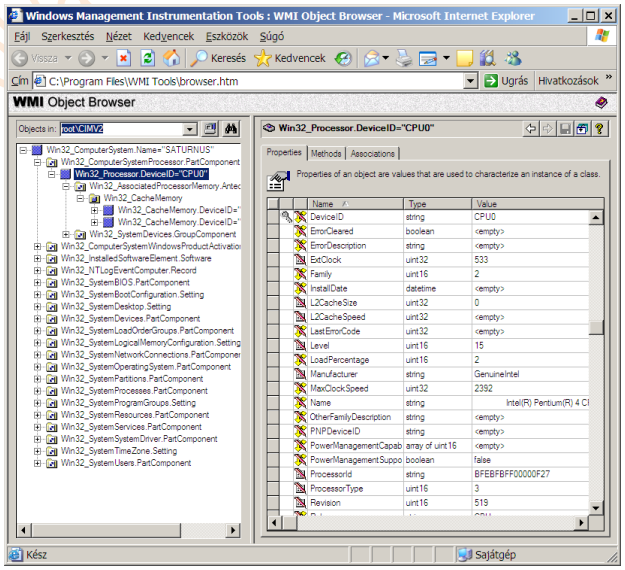
3.3.2 A WMI objektummodellje

A WMI működésének megértéséhez mindenképpen szükséges az objektummodell, és az ezzel kapcsolatos fogalmak ismerete; a következőkben erről lesz szó.

Osztályok (Classes) – A WMI objektummodellje osztályokon alapul. Az osztály a rendszer valamely felügyelt objektumának típusleírása, amely tartalmazza az adott objektum tulajdonságait és az általa támogatott metódusokat. A Win32_NetworkAdapter osztály például a számítógépben lévő hálózati adapterek típusleírását tartalmazza. Az osztályok között természetesen öröklődés is létezik. A szokásos felügyeleti szkriptek (programok) általában az öröklési lánc legvégén lévő (levél) objektumokkal foglalkoznak, de természetesen lehetőség van arra is, hogy feljebb menjünk a hierarchiában, és egy adott kezdőpont alatt lévő osztályok hasznos adatait nyerjük ki, anélkül, hogy pontosan tudnánk azok nevét. Absztrakt osztálynak nevezzük azokat az osztályokat, amelyekből nem lehet példányt létrehozni; kizárólag örökítési célokat szolgálnak.

Tulajdonságok (Properties) – Az osztályokhoz tulajdonságok tartoznak, amelyek az osztály által meghatározott objektumok leírására szolgálnak. Természetesen minden osztályban olyan tulajdonságok vannak definiálva, amelyek az adott típusú objektum leírásához szükségesek. Számos osztályban találkozhatunk kulcs (key) tulajdonsággal, ezek (az adatbázisokhoz hasonlóan) az adott osztály példányainak egyedi azonosítására használhatók. A Win32_NetworkAdapter osztály kulcstulajdonsága például a „DeviceID”.

Példányok (Instances) – Míg a Win32_NetworkAdapter osztály bármilyen hálózati adapter leírására képes, az osztály példánya egy bizonyos, fizikailag is létező adapter reprezentációja. Az egyes példányokat az adott osztály kulcs tulajdonságának ismeretében szólíthatjuk meg. Az objektum példányokat a WMI Object Browser segítségével jeleníthetjük meg (az eszköz a WMI Tools csomag része).



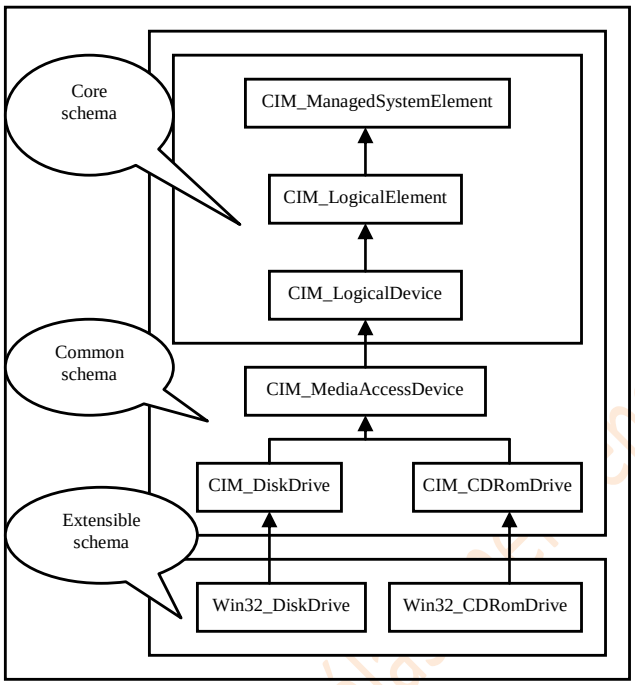
79. ábra WMI Object Browser

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

3.3.3 Sémák

A Common Information Model sémákból épül fel, amelyek egymással kapcsolatban álló osztályokat tartalmaznak. Jelenleg a CIM három sémát tartalmaz, az alábbi ábrának megfelelően:



80. ábra A CIM sémák

A „core” sémához tartozó absztrakt osztályok kevés konkrétumot tartalmaznak, céljuk az, hogy újabb osztályokat örökíthessünk belőlük. Ezen a szinten tulajdonképpen még az sem biztos, hogy az örökített osztályoknak bármi köze is lesz a számítógépes rendszerekhez; modellezhetnek akár épületeket, vagy berendezési tárgyakat is. A „common” séma viszont már határozottan számítógép-rendszerek modellezésére szolgál.

Az említett két sémát a WBEM szabvány definiálja, így azok gyártó- és platformfüggetlenek.

Az „extensible” sémák viszont már az egyes szoftvergyártók hatáskörébe tartoznak; ide kerülhetnek az adott platformra vonatkozó speciális osztályok. A WMI a Win32_ sémát használja a felügyelt objektumok modellezésére; az itt megtalálható osztályok a „common” séma osztályainak bővített (örökített) változatai, így megjeleníthetik a Windows operációs rendszerekre jellemző speciális tulajdonságokat is.

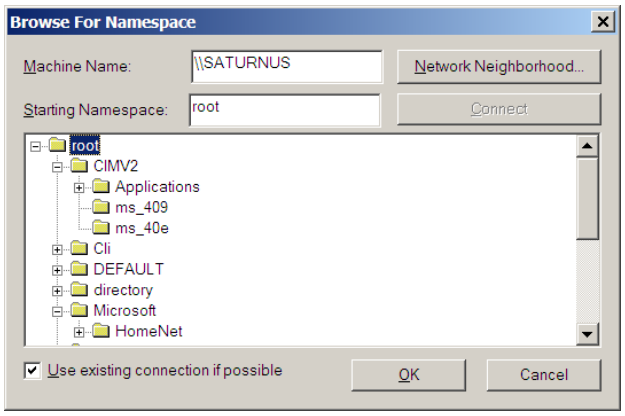
3.3.4 Névterek

A WMI osztályai különböző névterekhez tartoznak annak megfelelően, hogy melyik rendszerterületet jelenítik meg. A névterek szervezése hierarchikus, a fa gyökere a „root” névtér. Az egyes osztályok útvonalának megadásakor először is meg kell határoznunk azt a számítógépet, amelyik a CIM Repositoryt tartalmazza, majd sorban meg kell adnunk a hozzá vezető névtér-hierarchia elemeit:

```
\\Gep\Root\Cimv2:Win32_LogicalDisk.DeviceID='C:'
```

A könyv tanfolyami felhasználása nem engedélyezett!

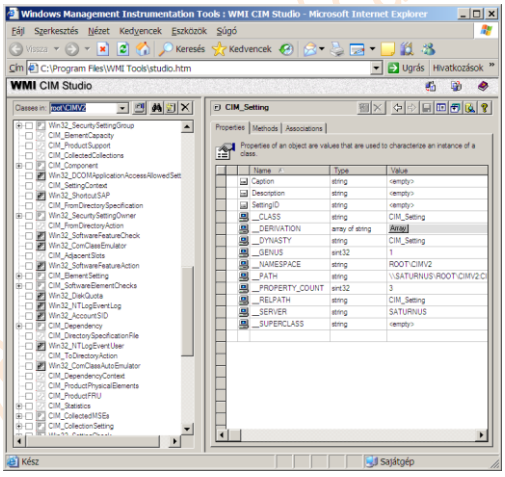
© Soós Tibor – soost@ IQJB.hu



81. ábra A névterek szervezése hierarchikus

Amint az ábrán is látható, minden számítógépen számos különböző névteret találhatunk, de a számítógép hardverelemeivel és az operációs rendszerrel kapcsolatos objektumokat leíró osztályok szinte kivétel nélkül a Cimv2 névtérben találhatók, így természetesen a rendszerfelügyeleti szkriptekben ez lesz a legtöbbet használt névtér.

A WMI CIM Studio (WMI Tools) segítségével a sémát és a névtereket jeleníthetjük meg. Felhasználhatjuk osztályok, illetve ezek tulajdonságainak és metódusainak megkeresésére (erre gyakran lesz szükség), és az osztályok közötti kapcsolatok feltérképezésére is.



82. ábra WMI CIM Studio

3.3.5 A legfontosabb providerek

A WMI providerek két forrásból származhatnak; vannak alkalmazás specifikus (ezeket egyes alkalmazások telepítik), és vannak beépített (ezeket a Windows részeként kapjuk) providerek. A következőkben áttekintjük az operációs rendszer részeként érkező legfontosabb providereket:

Win32 – a Win32 provider a számítógép hardver elemeihez és az operációs rendszer legfontosabb komponenseihez tartozó osztályok kiszolgálását végzi. A provider az adatok összegyűjtéséhez a Win32 API-t és különféle registry értékeket használ. A root/cimv2 névtér osztályainak döntő többségét ez a provider szolgálja ki.

SNMP – A provider a meglévő SNMP infrastruktúra és a WMI lehetőségeinek együttes használatát teszi lehetővé. A Windows Server 2003-ból ez a provider már hiányzik.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Performance Counter – A WMI legújabb verziójában a Windows Management Service a Performance Monitor által is használt adatfájlok alapján felépíti a teljesítményobjektumokat reprezentáló osztályokat a CIM Repositoryban. Az alkalmazások a többi WMI osztályhoz hasonlóan kérdezhetik le a teljesítményadatokat.

Registry – A registry providert felhasználó osztályok lehetővé teszik, hogy az alkalmazások írhassák és olvashassák a registryben szereplő értékeket. A RegistryEvent provider (regevent.mof) segítségével pedig alkalmazásunk értesítést kaphat a kiválasztott registry értékek módosulásáról.

Windows Driver Model – A Windows Driver Model lehetővé teszi, hogy az eszközvezérlő programok adatokat szolgáltassanak az általuk vezérelt eszközzel kapcsolatban .A provider ezekhez az adatokhoz biztosít hozzáférést a root/wmi névtérben létrehozott osztályok segítségével.

Directory Services – a DS provider az Active Directory osztályait és objektumait teszi elérhetővé a WMI-t használó alkalmazások számára. A provider az AD sémát képezi le a WMI sémába. A DS provider ADSI segítségével csatlakozik az Active Directoryhoz.

Event Log – a provider a Windows Eseménynapló szolgáltatáshoz biztosít hozzáférést, és lehetővé teszi azt is, hogy programunk értesítést kapjon az új naplóbejegyzések keletkezéséről.

Windows Installer – a provider a root/cimv2 névtérben létrehozott osztályok segítségével biztosítja a hozzáférést a Windows Installer szolgáltatás által telepített csomagokkal kapcsolatos adatokhoz. A provider lehetővé teszi az MSI csomagok telepítését, eltávolítását és beállítását is.

Security – A provider lehetővé teszi a Windows rendszer biztonsági beállításainak kiolvasását és módosítását. Beállíthatjuk a fájlok és mappák tulajdonosát, naplózását, és a hozzáférési jogokat is.

A könyv tanfolyami felhasználása nem engedélyezett!

3.4 COM-objektumok

A Component Object Model (COM) a komponens alapú rendszerek fejlesztésének Microsoft féle szabványa. Lehetővé teszi, hogy a COM-ügyfelek bináris szinten hívjanak meg olyan függvényeket, amelyeket a COM-objektumok számukra elérhetővé tesznek. A COM-objektumok mindegyike egyedileg azonosítható, önálló komponens, a különféle alkalmazások, és más komponensek egy jól meghatározott csatolófelületen keresztül vehetik igénybe az adott komponens szolgáltatásait.

A Microsoft is COM-komponensekkel tette lehetővé a hozzáférést az operációs rendszer függvényeihez, így a rendszerfelügyelettel kapcsolatos szkriptek igen jelentős része használ COM-objektumokat; segítségükkel érhetjük el a rendszer különféle elemeit, újrahasznosíthatjuk a már létező, előre megírt funkciókat. PowerShell esetén gyakran még akkor is COM-objektumot használunk, amikor látszólag nem, mivel a .NET rendszerfelügyelettel kapcsolatos osztályainak legtöbbje titokban szintén a megfelelő COM-objektum metódusait hívogatja (ADSI, WMI, stb.).

A szkriptjeinkben használt COM-objektumok mindegyike tehát olyan funkcionalitás-gyűjtemény, amelyet gondos kezek előre elkészítettek, és jól felhasználható formában összecsomagoltak számunkra. Ezek az objektumok szinte minden esetben bináris formában léteznek, gépünkön egy dinamikusan csatolt könyvtár (dll), vagy Active-X kontroll (ocx) képében találhatók meg. Szerencsére a COM-objektumok belső felépítéséről, adatszerkezeteiről semmit nem kell tudnunk ahhoz, hogy megírhassuk a velük kommunikáló szkriptet, és felhasználhassuk a bennük rejtőző képességeket. Az objektumokkal való kommunikáció metódusaik meghívását és tulajdonságaik beállítását jelenti; a rendszerfelügyeleti szkriptek legfontosabb funkciói ilyen módon valósíthatók meg.

3.4.1 Típuskönyvtárak

Típuskönyvtárnak az olyan csomagokat (dll vagy ocx fájlok) nevezzük, amelyek több, általában egymással szoros kapcsolatban álló osztályból állnak. A típuskönyvtár használatával az abban szereplő osztályok mindegyike alapján létrehozhatjuk a megfelelő objektumot, azaz elvégezhetjük az osztály példányosítását.

A könyv tanfolyami felhasználása nem engedélyezett!

3.5 Active Directory kezelése PowerShell 1.0-val

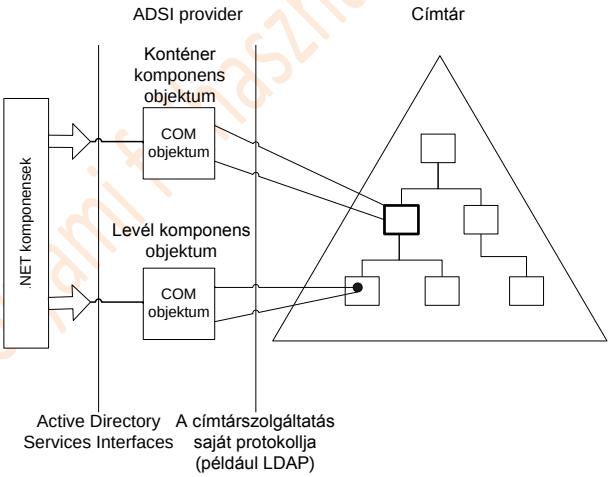
3.5.1 Mi is az ADSI?

Az ADSI segítségével a különféle címtárakat egységes formában kezelhetjük, mivel az általa biztosított csatolófelületek felhasználásával alkalmazásunk a különféle szerkezetű címtárakat is azonos módon érheti el. Az ADSI hasonló szerepet tölt be a címtárak elérésében, mint az ODBC a különféle adatbázis-kiszolgálók esetében. Segítségével a címtárakkal kapcsolatos gyakori feladatok – felhasználók kezelése, különféle erőforrások keresése – az elosztott, több címtármegoldást használó környezetekben is könnyen elvégezhetőek.

Az ADSI objektumai valójában COM objektumok, amelyek a kezelni kívánt címtár objektumait jelenítik meg, és COM csatolófelületeken keresztül érhetőek el. Az objektumok két csoportba sorolhatók; a konténer típusú objektumok más konténereket, és levél típusú objektumokat tartalmazhatnak.

3.5.2 ADSI providerek

Az ADSI providerek valósítják meg a konkrét címtártípus eléréséhez szükséges funkciókat. Amint az alábbi ábrán is látható, az ADSI-t használó ügyfeleknek (így szkriptjeinknek, vagy a .NET Framework komponenseinek is) csak az ADSI provider által biztosított csatolófelület megfelelő használatával kell törődniük, a háttérben lévő címtár egyedi adottságainak megfelelő műveletek kezdeményezése már a provider feladata.



83. ábra Címtár elérése ADSI provideren keresztül

Az ADSI által biztosított legfontosabb providerek a következők:

WinNT:// – a provider segítségével a Windows NT 4.0 tartományokhoz férhetünk hozzá, és ez teszi lehetővé az önálló (Windows NT, 2000, XP) számítógépeken levő helyi címtár-adatbázisokhoz (SAM) való hozzáférést is. Meg kell jegyeznünk, hogy a WinNT provider kompatibilitási okokból használható Active Directory esetében is, de ez számos korlátozással jár (a keresés nem támogatott, szervezeti egységek megadása nem lehetséges, stb.) ezért használata nem célszerű.

LDAP:// – ez a provider az LDAP protokoll segítségével elérhető címtárak (például az Active Directory) kezelésére szolgál.

NDS:// – segítségével a Novell Directory Services szolgáltatáshoz férhetünk hozzá.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

3.5.3 Az ADSI gyorsítótár

Minden ADSI objektum ügyféloldali attribútum-gyorsítótárral rendelkezik, amelyben ideiglenesen tárolódnak az objektumhoz tartozó tulajdonságok nevei és értékei. A gyorsítótár jelentősen növeli az attribútumokkal végzett különféle műveletek teljesítményét, mivel nélküle programunknak minden egyes attribútum értékének beállítása után a címtárhoz kéne fordulnia.

3.5.4 Active Directory információk lekérdezése

Az elméleti áttekintés után nézzük élesben a címtárak elérését PowerShell segítségével, mellyel az Active Directory-val kapcsolatos felügyeleti tevékenységek is hatékonyan automatizálhatók. De még mielőtt ebbe beleásnánk magunkat, nézzünk pár előkészítő tevékenységet. Miután a PowerShell alatt a .NET keretrendszer található, így fontos ismerni a címtárak elérésének problémáit diagnosztizálni képes fontosabb osztályokat.

Miután az Active Directory hibák 110%-a (☺) valamilyen DNS hibára vezethető vissza, így elsőként ellenőrizzük a névfeloldást. Az ezzel kapcsolatos .NET osztály a `System.Net.Dns`, melynek `GetHostEntry` statikus metódusával tudjuk ellenőrizni például a tartományvezérlőnk nevének feloldását:

```
PS C:\Users\Administrator> [System.Net.Dns]::GetHostEntry("adds.iqjb.w08")

HostName                    Aliases                     AddressList
--------                    -------                     -----------
adds.iqjb.w08               {}                          {192.168.1.2}
```

Ha ez helyes eredményt ad, akkor folytathatjuk az AD felderítését, ellenőrzését az erdő legfontosabb objektumaival. Erre a célra a `System.DirectoryServices.ActiveDirectory.Forest` osztály alkalmas, annak is a `GetCurrentForest` statikus metódusa:

```
PS C:\Users\Administrator> [System.DirectoryServices.ActiveDirectory.Forest]
::getcurrentforest()


Name                    : iqjb.w08
Sites                   : {Default-First-Site-Name}
Domains                 : {iqjb.w08}
GlobalCatalogs          : {adds.iqjb.w08}
ApplicationPartitions   : {DC=ForestDnsZones,DC=iqjb,DC=w08, DC=DomainDnsZone
                          s,DC=iqjb,DC=w08}
ForestMode              : Windows2008Forest
RootDomain              : iqjb.w08
Schema                  : CN=Schema,CN=Configuration,DC=iqjb,DC=w08
SchemaRoleOwner         : adds.iqjb.w08
NamingRoleOwner         : adds.iqjb.w08
```

Láthatjuk, hogy ez a metódus a legfontosabb adatokat megadja az erdőről: a *root* tartomány és a többi tartomány nevét, a globális katalógusok listáját, az erdő működési szintjét és az erdőszintű egyedi szerepeket hordozó tartományvezérlők neveit.

Hasonló módon megvizsgálhatjuk az aktuális tartomány adatait is a `System.DirectoryServices.ActiveDirectory.Domain` osztály `getcurrentdomain` statikus metódusa segítségével:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\Users\Administrator> [System.DirectoryServices.ActiveDirectory.Domain]
::getcurrentdomain()


Forest                    : iqjb.w08
DomainControllers         : {adds.iqjb.w08}
Children                  : {}
DomainMode                : Windows2008Domain
Parent                    :
PdcRoleOwner              : adds.iqjb.w08
RidRoleOwner              : adds.iqjb.w08
InfrastructureRoleOwner : adds.iqjb.w08
Name                      : iqjb.w08
```

Itt a legfontosabb plusz információ a tartományi szintű egyedi szerepeket hordozó tartományvezérlők nevei.

Nem mellékes, hogy milyen AD site (telephely) beállításokkal dolgozunk, hiszen ez befolyásolja az ügyfél gépek tartományvezérlő választását és a címtár replikációt. A telephely információk kiolvasására a `System.DirectoryServices.ActiveDirectory.ActiveDirectorySite` osztály `GetComputerSite` statikus metódusa használható:

```
PS C:\> [System.DirectoryServices.ActiveDirectory.ActiveDirectorySite]::GetC
omputerSite()


Name                            : Default-First-Site-Name
Domains                         : {iqjb.w08}
Subnets                         : {192.168.112.0/24}
Servers                         : {w2k8.iqjb.w08}
AdjacentSites                   : {}
SiteLinks                       : {DEFAULTIPSITELINK}
InterSiteTopologyGenerator      : w2k8.iqjb.w08
Options                         : None
Location                        : Budapest
BridgeheadServers               : {}
PreferredSmtpBridgeheadServers : {}
PreferredRpcBridgeheadServers   : {}
IntraSiteReplicationSchedule    : System.DirectoryServices.ActiveDirectory.A
                                  ctiveDirectorySchedule
```

Ezzel a néhány kifejezéssel tehát elég jól át lehet tekinteni, hogy milyen az AD infrastruktúránk, fájlba történő átirányítással akár az AD rendszerünk dokumentálásához is segítséget kapunk.

3.5.5 Csatlakozás az Active Directory-hoz

Az előzőekben a .NET keretrendszer osztályainak statikus metódusaival dolgoztam, melyek segítségével általános információkat lehetett kinyerni az Active Directory környezetről. Ha konkrét, adott tartományra vagy címtárelemre vonatkozó információkhoz akarunk hozzájutni, akkor csatlakozni kell az adott címtár objektumhoz. Az előzőekben leírtuk, hogy a címtár kezelésében fontos szerepe van a gyorsítótárnak, azaz egy ilyen csatlakoztatás előkészíti a memóriában az adott címtár objektum egy reprezentációját. Ha ezek után változtatunk az objektum valamely tulajdonságán, akkor ez csak a memóriában hajtódik végre, külön metódussal kell ezt a változást a címtárba visszaírni, mint ahogy ezt látni fogjuk.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Elsőként azonban nézzük meg a legegyszerűbb csatlakoztatást:

```
PS C:\> $domain = [ADSI] ""
```

Az `[ADSI]` típusjelölővel hivatkozunk a címtáras elérésre, és ha egy üres sztringet adunk meg a „konstruktor” paramétereként, akkor az aktuális tartományi objektumhoz csatlakozunk. Ez a típusjelölő a `System.DirectoryServices.DirectoryEntry` .NET osztály rövidített neve. Ezt akár ki is írhatjuk, és ekkor további paramétereket is megadhatunk, ha szükséges:

```
PS C:\> $domain = new-object DirectoryServices.DirectoryEntry("","iqjb\Admin
istrator", "Password1")
```

Olvassuk ki, hogy mi került a `$domain` változónkba:

```
PS C:\> $domain

distinguishedName
-----------------
{DC=iqjb,DC=w08}
```

Ez még nem túl sok, nézzük meg a tagjellemzőit:

```
PS C:\> $domain | Get-Member


   TypeName: System.DirectoryServices.DirectoryEntry

Name                                MemberType Definition
----                                ---------- ----------
auditingPolicy                      Property   System.DirectoryServices.Pro...
creationTime                        Property   System.DirectoryServices.Pro...
dc                                  Property   System.DirectoryServices.Pro...
distinguishedName                   Property   System.DirectoryServices.Pro...
dSCorePropagationData               Property   System.DirectoryServices.Pro...
forceLogoff                         Property   System.DirectoryServices.Pro...
fSMORoleOwner                       Property   System.DirectoryServices.Pro...
gPLink                              Property   System.DirectoryServices.Pro...
instanceType                        Property   System.DirectoryServices.Pro...
isCriticalSystemObject              Property   System.DirectoryServices.Pro...
lockoutDuration                     Property   System.DirectoryServices.Pro...
lockOutObservationWindow            Property   System.DirectoryServices.Pro...
...
uSNChanged                          Property   System.DirectoryServices.Pro...
uSNCreated                          Property   System.DirectoryServices.Pro...
wellKnownObjects                    Property   System.DirectoryServices.Pro...
whenChanged                         Property   System.DirectoryServices.Pro...
whenCreated                         Property   System.DirectoryServices.Pro...
```

Kicsit megvágtam, de ebből is látszik, hogy jó néhány tulajdonsága van egy ilyen objektumnak. Nézzük, hogy hogyan tudunk egy nevesített objektumhoz, mondjuk egy felhasználói fiókhoz csatlakozni:

```
PS C:\> $user = [ADSI] "LDAP://cn=János Vegetári,ou=Demó,dc=iqjb,dc=w08"
PS C:\> $user

distinguishedName
-----------------
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
{CN=János Vegetári,OU=Demó,DC=iqjb,DC=w08}
```

Fontos!

Az `[ADSI]` utáni sztringben csupa nagybetűs az LDAP, és normál perjelek vannak utána. Ha nem csupa nagybetűs az LDAP, vagy fordított perjelet használunk, akkor nem rögtön kapunk hibajelzést, hanem akkor, amikor először használjuk az objektumot.

A későbbiekben majd azt is megnézzük, hogy egy ilyen felhasználói fióknak milyen tulajdonságai vannak és hogyan lehet azokat módosítani.

3.5.6 AD objektumok létrehozása

AD objektumokat létrehozni a VBScriptben megszokott (már aki programozott VBScriptben) ADSI szintaxishoz nagyon hasonló módon lehet. Először egy AD konténerre kell csatlakozni, ahova létre szeretnénk hozni az új objektumot. Ez a csatlakozás az előző fejezetben látott módon megy, ezzel ugye „átemeljük" a PowerShellbe az adott konténert, mint objektumot.

Az így „átemelt" AD objektum `Create` metódusával lehet létrehozni az új AD elemet. A `Create` paramétereként meg kell adni a létrehozandó objektum típusát és az un. „*relative distinguished name*" nevét, azaz az adott konténeren belüli megkülönböztető nevét.

Az alábbi példában magán a tartományvezérlőn (`localhost`), közvetlenül a tartomány objektum alá hozok létre egy szervezeti egységet (`organizational unit`):

```
PS C:\> $konténer = [adsi] "LDAP://localhost:389/dc=iqjb,dc=w08"
PS C:\> $adObj = $konténer.Create("OrganizationalUnit", "OU=Emberek")
PS C:\> $adObj.Put("Description", "Normál felhasználók")
PS C:\> $adObj.SetInfo()
```

Ebben ugye az az újdonság, hogy az LDAP kifejezésbe beillesztettem a tartományvezérlő nevét és a portszámot, ahol a címtárszolgáltatás elérhető. A szemléltetés kedvéért még a „`Description`" attribútumát is kitöltöttem. Vigyázat, amit idáig tettem azt mind a memóriában tárolódó gyorsítótárban végeztem el, ahhoz, hogy mindez ténylegesen bekerüljön a címtár adatbázisba meg kell hívni a `SetInfo` metódust!

3.5.7 AD objektumok tulajdonságainak kiolvasása, módosítása

Ha PowerShell, akkor objektumok. Az előzőekhez hasonlóan csatlakozzunk a most létrehozott szervezeti egység objektumhoz, és nézzük meg a tagjellemzőit a `get-member` cmdlet segítségével:

```
PS C:\> $adou = [ADSI] "LDAP://OU=Emberek,DC=iqjb,DC=w08"
PS C:\> $adou

distinguishedName
-----------------
{OU=Emberek,DC=iqjb,DC=w08}
```

A könyv tanfolyami felhasználása nem engedélyezett!

```
PS C:\> $adou | get-member


   TypeName: System.DirectoryServices.DirectoryEntry

Name                    MemberType Definition
----                    ---------- ----------
description             Property   System.DirectoryServices.PropertyValueC...
distinguishedName       Property   System.DirectoryServices.PropertyValueC...
dSCorePropagationData   Property   System.DirectoryServices.PropertyValueC...
instanceType            Property   System.DirectoryServices.PropertyValueC...
name                    Property   System.DirectoryServices.PropertyValueC...
nTSecurityDescriptor    Property   System.DirectoryServices.PropertyValueC...
objectCategory          Property   System.DirectoryServices.PropertyValueC...
objectClass             Property   System.DirectoryServices.PropertyValueC...
objectGUID              Property   System.DirectoryServices.PropertyValueC...
ou                      Property   System.DirectoryServices.PropertyValueC...
uSNChanged              Property   System.DirectoryServices.PropertyValueC...
uSNCreated              Property   System.DirectoryServices.PropertyValueC...
whenChanged             Property   System.DirectoryServices.PropertyValueC...
whenCreated             Property   System.DirectoryServices.PropertyValueC...
```

Hát elég furcsa, amit kaptunk. Látjuk a szervezeti egységünk tulajdonságait, de hol vannak a metódusok? Hol a `Create`? Sajnos a PowerShell 1.0-ba még nincsen 100%-ban adaptálva a `System.DirectoryServices` osztály. Ennek több oka van. Az egyik, hogy valójában itt nem színtiszta .NET osztályról van szó, hanem COM objektum is meghúzódik a felszín alatt, és annak metódusait nem olyan egyszerű átemelni. Gondoljunk csak arra, hogy egy ilyen `DirectoryEntry` típusú objektum lehet felhasználói fiók, számítógép fiók, telephely, csoport, stb., ezeknek mind más és más metódusuk van, ezeknek az adaptálása a PowerShell környezetbe nem olyan egyszerű. Ebből származik a második ok, ami miatt ez nincs adaptálva, az pedig az, hogy a fejlesztők az 1.0 megjelenését nem akarták ezzel késleltetni, várhatóan a 2.0 verzió már precízebb AD támogatást fog nyújtani.

Szerencsére van egy kis menekvési ösvényünk, azaz kikapcsolhatjuk a PowerShell adaptációs rétegét, és megnézhetjük a „színtiszta” .NET objektumot is, ha a `psbase` nézeten keresztül nézzük az objektumunkat:

```
PS C:\> $adou.psbase | get-member


   TypeName: System.Management.Automation.PSMemberSet

Name                      MemberType Definition
----                      ---------- ----------
...
MoveTo                    Method     System.Void MoveTo(DirectoryEntry n...
RefreshCache              Method     System.Void RefreshCache(), System....
remove_Disposed           Method     System.Void remove_Disposed(EventHa...
Rename                    Method     System.Void Rename(String newName)
...
ToString                  Method     System.String ToString()
AuthenticationType        Property   System.DirectoryServices.Authentica...
Children                  Property   System.DirectoryServices.DirectoryE...
Container                 Property   System.ComponentModel.IContainer Co...
Guid                      Property   System.Guid Guid {get;}
Name                      Property   System.String Name {get;}
NativeGuid                Property   System.String NativeGuid {get;}
NativeObject              Property   System.Object NativeObject {get;}
ObjectSecurity            Property   System.DirectoryServices.ActiveDire...
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
Options                    Property   System.DirectoryServices.DirectoryE...
Parent                     Property   System.DirectoryServices.DirectoryE...
Password                   Property   System.String Password {set;}
Path                       Property   System.String Path {get;set;}
Properties                 Property   System.DirectoryServices.PropertyCo...
SchemaClassName            Property   System.String SchemaClassName {get;}
SchemaEntry                Property   System.DirectoryServices.DirectoryE...
Site                       Property   System.ComponentModel.ISite Site {g...
UsePropertyCache           Property   System.Boolean UsePropertyCache {ge...
Username                   Property   System.String Username {get;set;}
```

A fenti, kicsit megvágott, de még így is hosszú listából látszik, hogy az objektumot valójában lehet például mozgatni, átnevezni, és néhány újabb tulajdonság is feltárul a szemünk előtt. De például még mindig nem látjuk a `SetInfo` és a `Create` metódust, mert ezek az ADSI COM interfészből jönnek, és a .NET nem kérdezi ezeket le, így nem is mutatja meg, viszont meghívni, használni lehet őket.

Vagy nézzük a következőket:

```
PS C:\> $d = [ADSI] ""
PS C:\> $d

distinguishedName
-----------------
{DC=iqjb,DC=w08}
```

A fenti módon például nagyon egyszerűen lehet az aktuális tartományunkhoz csatlakozni. Próbáljuk meg ennek megnézni a „rejtett” `children` tulajdonságát:

```
PS C:\> $adou = [ADSI] "LDAP://OU=Demó,DC=iqjb,DC=w08"
PS C:\> $adou.psbase.Children

distinguishedName
-----------------
{CN=Csilla Fájdalom,OU=Demó,DC=iqjb,DC=w08}
{CN=Csoport,OU=Demó,DC=iqjb,DC=w08}
{CN=group1,OU=Demó,DC=iqjb,DC=w08}
{CN=János Vegetári,OU=Demó,DC=iqjb,DC=w08}
{CN=Márton Beléd,OU=Demó,DC=iqjb,DC=w08}
```

Hiszen ez megadta az adott konténer objektumban található al-objektumokat!

Megjegyzés

A lokális gép esetében a `PSBase` egészen extrém információkat rejt el:

```
[40] PS C:\> $computer = [ADSI]"WinNT://."
[41] PS C:\> $v = $computer.psbase.children | ForEach-Object {$_}
[42] PS C:\> $v[0].name; $v[0].psbase.schemaclassname
Administrator
User
[43] PS C:\> $v[30].name; $v[30].psbase.schemaclassname
BITS
Service
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

A [40]-es sorban hozzákapcsolódom a helyi géphez egy `$computer` változón keresztül. Majd betöltöm egy `$v` változóba ennek a `psbase` által feltárt `children` tulajdonságának elemeit egy ciklussal. Ennek 0. elemének `name` tulajdonságát nézve kiderül, hogy ez egy helyi felhasználó, az Administrator. Ennek típusát is ki lehet olvasni egy újabb `psbase` feltárás után a `schemaclassname` tulajdonsággal.
A meglepetés akkor jön, ha egy jóval későbbi elemet nézünk meg, mondjuk a 90.-et. Nálam ez meg egy szolgáltatás volt, a BITS (a kedves olvasónál lehet, hogy ez valami más). Azaz a lokális gép esetében a `children` a gépnek nagyon sok jellemzőjét megadja. Az eligazodást segíti a `Find` metódus:

```
[47] PS C:\> $service = $computer.psbase.Children.Find("Alerter")
[48] PS C:\> $service.serviceaccountname
NT AUTHORITY\LocalService
```

Miután itt vegyes elemekkel dolgozunk, itt különösen jól jön a `get-member` cmdlet.

Mindebből az következik, hogy nem érdemes még kidobni korábbi ADSI ismereteinket, illetve ismernünk kell az AD objektumok tulajdonságainak neveit, hogy ezeket a tulajdonságokat módosíthassuk. Nézzünk ez utóbbira példát egy felhasználói fiókkal kapcsolatban. Van egy már létező felhasználóm, annak szeretném kiolvasni és beállítani a telefonszám tulajdonságát. Ehhez kell nekünk az, hogy tudjuk, hogy az AD-ben a telefonszám tulajdonságnak mi is a belső elnevezése. Ennek felderítésére több módszer van, nézzünk egy PowerShell-est:

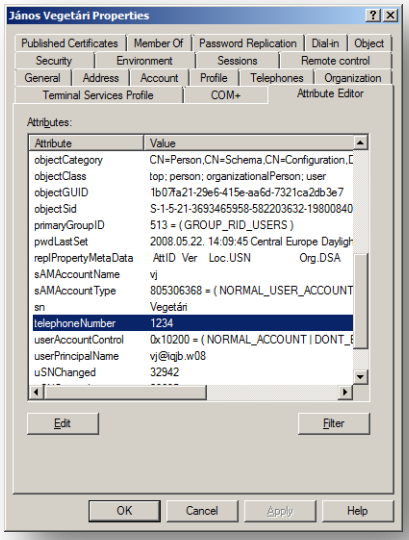
```
PS C:\> $user = [ADSI] "LDAP://cn=János Vegetári,OU=Demó,DC=iqjb,DC=w08"
PS C:\> $user.psbase.properties

PropertyName       Value                          Capacity              Count
------------       -----                          --------              -----
objectClass        {top, person, o...                    4                  4
cn                 János Vegetári                        4                  1
sn                 Vegetári                              4                  1
telephoneNumber    2008                                  4                  1
givenName          János                                 4                  1
distinguishedName  CN=János Vegetá...                    4                  1
...
```

A fenti listában látjuk, hogy a telefonszám attribútum neve - meglepő módon – `telephoneNumber`.

Vagy a Windows Server 2008 esetében az *Active Directory Users and Computers* eszköz szerencsére már tartalmaz egy *Attribute Editor* fület, a korábbi Windows változatoknál az *ADSIEdit* eszközzel érhetjük el ugyanezt:

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu



84. ábra A Windows Server 2008 ADUC új Attribute Editor-a

Nézzük meg, hogyan lehet ezt a telefonszámot kiolvasni, majd átírni. Az első megoldás a „PowerShell-es”:

```
PS C:\> $user.telephoneNumber
1234
PS C:\> $user.telephoneNumber=2008
PS C:\> $user.setinfo()
PS C:\> $user.telephoneNumber
2008
```

Ebben az a furcsa, hogy a `Get-Member`-rel nem lehetett kiolvasni, hogy a `$user`-nek van `telephoneNumber` tulajdonsága, mégis lehet használni.

A második megoldás a hagyományos, ADSI-stílus:

```
PS C:\> $u.Get("telephoneNumber")
1234
PS C:\> $u.Put("telephoneNumber",9876)
PS C:\> $u.SetInfo()
PS C:\> $u.Get("telephoneNumber")
9876
```

A `Get` metódussal tudjuk az adott tulajdonságot kiolvasni, a `Put`-tal átírni. Egyik esetben sem szabad megfeledkezni a `SetInfo`-ról, ami az objektum memóriabeli reprezentációját írja be ténylegesen az AD-ba.

A `Get` és a `Put` is „rejtett” metódus, a paraméterezésük a példában látható. Az SD attribútumok LDAP nevére kell hivatkozni mindkettőnél.

Megjegyzés

Sajnos nem minden attribútum kezelhető a PowerShell-es módszerrel. Ilyen például a *Company* attribútum:

```
PS C:\> $u.company
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
PS C:\> $u.get("company")
Cég
```

Az első esetben nem kaptam semmilyen választ az attribútum kiolvasására, de `get`-tel mégis működött.

Mi van akkor, ha kiolvastuk egy felhasználó adatait egy változóba, majd ezután valaki egy másik gépről vagy egy másik alkalmazással módosítja a felhasználónk valamely attribútumát. Ilyenkor a `getinfo` metódussal lehet frissíttetni az adatokat a memóriában:

```
PS C:\> $u.get("company")
Egyik
PS C:\> $u.getinfo()
PS C:\> $u.get("company")
Másik
```

A fenti példában az első kiolvasás után az ADUC eszközzel az első `get` után átírtam a felhasználó *Company* attribútumát, és a `getinfo`-val ezt frissítettem a memóriában, így az új érték már a PowerShell-ből is látszik.

3.5.7.1 Munka többértékű (multivalued) attribútumokkal

Az Active Directory egyik jellegzetes attribútuma az un. „multivalued property”. Ez olyan tulajdonság, ahova az értékek listáját, tömbjét tehetjük. Legtipikusabb ilyen attribútum az Exchange Server bevezetése után a felhasználók e-mail címeit tartalmazó `ProxyAddresses`, vagy a csoportok `Member` attribútuma, de erről majd külön értekezek. Marad mondjuk az `other`... kezdetű különböző telefonszámok tárolására szolgáló attribútumok, mint például az `otherMobile` vagy az `otherTelephone`.

Ezeket ki lehet olvasni az eddig megismert módszerekkel is, de nézzük, hogy milyen problémákkal szembesülhetünk. Ha a `get` metódust használom, és csak egy értéket tárol a „multivalued propery”, akkor nem egy elemű tömböt kapok, hanem sima skaláris értéket:

```
PS C:\> $user.get("otherMobile")
othermobil1234
PS C:\> $user.get("otherMobile").gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object
```

Ezzel szemben, ha több értéket tárolunk, akkor már tömböt kapunk:

```
PS C:\> $user.getinfo()
PS C:\> $user.get("otherMobile")
othermobil2345
othermobil1234
PS C:\> $user.get("otherMobile").gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

könyv tanfolyami felhasználása nem engedélyezett!

Ez nem biztos, hogy jó nekünk, mert így a szkriptünket kétfajta esetre kell felkészítenünk: külön arra az esetre, ha csak egy értéket tárolunk és külön arra az esetre is, ha többet. Ez bonyolítja a programjainkat.

Ha konzisztensen, mindig tömbként akarjuk kezelni az ilyen „multivalued property”-ket, akkor vagy használjuk a PowerShell-es stílust:

```
PS C:\> $user.otherMobile
othermobil1234
PS C:\> $user.otherMobile[0]
othermobil1234
PS C:\> $user.otherMobile.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    PropertyValueCollection                  System.Collec...
```

Vagy használjuk a `GetEx` metódust:

```
PS C:\> $user.getex("otherMobile")
othermobil1234
PS C:\> $user.getex("otherMobile").gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Az ilyen multivalued property-k írása sem egyértelmű, hiszen több lehetőségem is van:

- a meglevő értékekhez akarok egy újabbat hozzáfűzni,
- a meglevő értékek helyett akarok egy újat betölteni.

Ezeket a lehetőségeket én magam is le tudom programozni a szkriptemben. Ha az első változatra van szükségem, akkor előbb kiolvasom az attribútum aktuális tartalmát egy változóba, hozzárakom az új értéket és így rakom vissza a `put`-tal, vagy egyszerű értékadással. Ha pedig a második változatra van szükségem, akkor egyszerűen felülírom az attribútumot az új értékkel.

Sokkal elegánsabb, ha ezt már maga az objektum tudná egy „okosabb” metódussal. Ilyen létezik, ez pedig a `PutEx`:

```
PS C:\> $user.getex("otherMobile")
othermobil1234
PS C:\> $user.putex(3,"otherMobile",@("othermobilPutEx2"));$user.setinfo()
PS C:\> $user.getex("otherMobile")
othermobilPutEx2
othermobil1234
PS C:\> $user.putex(2,"otherMobile",@("othermobilPutEx3"));$user.setinfo()
PS C:\> $user.getex("otherMobile")
othermobilPutEx3
```

A fenti példában a kiinduló állapotban egy mobilszámunk van. Ezután hozzáfűzök egy újabbat a `putex` használatával, a hozzáfűzést az első paraméterként szereplő 3-as jelzi. Fontos, hogy a hozzáfűzendő értéket tömbként kell kezelni, ezért van ott a kukac-zárójelpár! Ezután egy újabb `putex`-et hívok meg, immár 2-es paraméterrel, ez a felülírás művelete, ennek hatására már csak ez a legújabb mobilszám lesz az attribútumban.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Használhatom még az 1-es paramétert is, ez ekvivalens az attribútum értékeinek törlésével, vagy használhatom a 4-es paramétert, ez a paraméterként megadott elemet töröli az értékek közül:

```
PS C:\> $user.putex(3,"otherMobile",@("Append"));$user.setinfo()
PS C:\> $user.getex("otherMobile")
Append
othermobilPutEx3
PS C:\> $user.putex(4,"otherMobile",@("Append"));$user.setinfo()
PS C:\> $user.getex("otherMobile")
othermobilPutEx3
```

A fenti példában elsőként hozzáfűzök egy értéket, majd ugyanezt eltávolítom.

3.5.7.2 Speciális tulajdonságok kezelése

Van néhány attribútum, amelyek az eddig megismert módszerek egyikével sem kezelhetők:

```
PS C:\> $user.AccountDisabled
PS C:\> $user.get("AccountDisabled")
Exception calling "get" with "1" argument(s): "The directory property canno
t be found in the cache.
"
At line:1 char:10
+ $user.get( <<<< "AccountDisabled")
```

A PowerShelles szintaxis meg se nyikkan, a `get` meg még hibát is jelez. Ilyen esetekben használhatjuk a `psbase` nézeten keresztül az `InvokeGet` és `InvokeSet` metódusokat:

```
PS C:\> $user.psbase.invokeget("AccountDisabled")
False
PS C:\> $user.psbase.invokeset("AccountDisabled","TRUE")
PS C:\> $user.SetInfo()
PS C:\> $user.psbase.invokeget("AccountDisabled")
True
```

3.5.8 Jelszó megváltoztatása

Speciális attribútum a jelszó, hiszen tudjuk, hogy valójában nem (feltétlenül) tárolja a címtár a jelszavakat, hanem csak a belőlük képzett hasht. Így a jelszó kezelésekor nem egyszerűen egy attribútumot kell beállítani, hanem ezt a hasht kell képezni. Szerencsére erre a célra rendelkezésünkre áll két metódus, a `SetPassword`, illetve a `ChangePassword`:

```
PS C:\> $user.SetPassword("UjPass2")
PS C:\> $user.ChangePassword("UjPass2","MégújabbPass3")
```

A `SetPassword` felel meg a *Reset Password* műveletnek. Ezt ugye csak megfelelő rendszergazda jogosultságok birtokában tudjuk megtenni. A `ChangePassword` a meglevő jelszó birtokában módosítja a jelszót, ehhez már nem kell külön rendszergazda jogosultság.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

3.5.9 Csoportok kezelése

Az Active Directory-ban csoportokat leginkább a rendszer üzemeltetésének megkönnyítésére veszünk fel. Segítségükkel osztunk ki hozzáférési jogokat, felhasználói jogokat, de még a csoportos házirendek érvényre jutását is szabályozhatjuk csoportokkal. Miután ilyen széleskörű a felhasználásuk, így fontos lehet a csoportok kezelésének automatizálása. Erre is kiválóan alkalmas a PowerShell, nézzük meg a leggyakoribb műveleteket.

Csoportot létrehozni a már látott módszerrel lehet:

```
PS C:\> $target = [ADSI] "LDAP://ou=Demó,DC=iqjb,DC=w08"
PS C:\> $group = $target.create("group","CN=Csoport")
PS C:\> $group.setinfo()
```

Ez alaphelyzetben globális biztonsági csoportot hoz létre. A későbbi, összetett példában majd bemutatom, hogy hogyan lehet másfajta csoportokat létrehozni.

Ezután kétféleképpen lehet tagokat adni a csoportokhoz. Az első módszer a hagyományos „ADSI”-s módszer, ahol a csoport `Add` metódusát hívom meg, paramétereként a berakni kívánt felhasználó LDAP-os szintaxisú elérési útját kell megadni. Vagy, ha már megragadtam a felhasználói fiókot, akkor vissza kell alakítani az LDAP-os elérési úttá, mint ahogy ebben a példában tettem:

```
PS C:\> $user = [ADSI] "LDAP://CN=János Vegetári,OU=Demó,DC=iqjb,DC=w08"
PS C:\> $group.add("LDAP://$($user.distinguishedname)")
PS C:\> $group.setinfo()
```

Hasonlóan lehet tagot eltávolítani, csak az `Add` helyett a `Remove` metódust kell meghívni.

A második módszer kicsit PowerShell-szerűbb, itt nem kell ide-oda alakítgatni, elég a felhasználó `distinguishedname` tulajdonságát használni:

```
PS C:\> $user = [ADSI] "LDAP://CN=Csilla Fájdalom,OU=Demó,DC=iqjb,DC=w08"
PS C:\> $group.member += $user.distinguishedname
PS C:\> $group.setinfo()
```

Természetesen a két megoldás egyenértékű, csak stílusbeli különbség van közöttük. A második módszer hátránya talán, hogy egyszerűen nem lehet csoporttagot eltávolítani, külön képezni kellene a nemkívánatos tag nélküli tömböt, és azt betölteni a csoport `member` tulajdonságába.

3.5.10 Keresés az AD-ben

A következő gyakori feladat a keresés az AD-ben. Az első módszer a `Find` metódus használata, ami szintén a `PSBase` nézetben érhető el:

```
PS C:\> $ou = [ADSI] "LDAP://ou=Demó,dc=iqjb,dc=w08"
PS C:\> $ou.psbase.children.find("cn=Csilla Fájdalom")

distinguishedName
-----------------
{CN=Csilla Fájdalom,OU=Demó,DC=iqjb,DC=w08}
```

Ez a keresés azonban csak az adott konténerobjektum gyerekei között keres, így nem biztos, hogy igazán jó segítség ez nekünk

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Az igazán profi keresőhöz a .NET keretrendszer egyik osztályát, a `System.DirectoryServices.DirectorySearcher`-t hívjuk segítségül, ennek egy objektuma lesz a keresőnk, és ennek különböző tulajdonságait beállítva adjuk meg a keresésünk mindenféle feltételét. Elsőként nézzünk egy nagyon egyszerű feladatot, egy konkrét felhasználói fiókra keressünk rá:

```
[6] PS I:\>$objRoot = [ADSI] "LDAP://OU=IQJB,DC=dom"
[7] PS I:\>$objSearcher = New-Object System.DirectoryServices.DirectorySearc
her
[8] PS I:\>$objSearcher.SearchRoot = $objRoot
[9] PS I:\>$objSearcher.Filter = "(&(objectCategory=user)(displayName=Soós
Tibor))"
[10] PS I:\>$objSearcher.SearchScope = "Subtree"
[11] PS I:\>$colResults = $objSearcher.FindAll()
[12] PS I:\>$colresults

Path                                    Properties
----                                    ----------
LDAP://CN=Soós Tibor,OU=Normal,OU=...   {homemdb, distinguishedname, count...
```

Elsőként definiálom, hogy az AD adatbázis-elemek fastruktúrájában, hol is keresek majd (`$objRoot`). Majd elkészítem a keresőt (`$objSearcher`), aminek `SearchRoot` tulajdonságaként az előbb létrehozott keresési helyet adom meg. Majd definiálom az LDAP formátumú szűrőt, amely ebben az esetben a „Soós Tibor” nevű felhasználókat jelenti, és ezt betöltöm a kereső `Filter` tulajdonságaként. LDAP szűrőben a következő összehasonlító operátorokat használhatok:

Operátor	Jelentés
=	Egyenlő
~=	Közel egyenlő
<=	Kisebb egyenlő
>=	Nagyobb egyenlő
&	És
|	Vagy
!	Nem

Végül meghatározom a keresés mélységét, ami itt `Subtree`, azaz mélységi, mert nem pont közvetlenül a kiindulópontként megadott helyen van a keresett objektum. Nincs más hátra, ezek alapján ki kell listázni a feltételeknek megfelelő objektumokat a `FindAll` metódussal.

A `$colResult` változóban tárolt eredmény nem közvetlenül `DirectoryEntry` típusú elemek tömbje! Hanem tulajdonképpen egy hashtábla-szerűség, ahol a `Path` oszlop tartalmazza a megtalált objektum LDAP formátumú elérési útját, a `Properties` meg a kiolvasható tulajdonságait. Azaz ahhoz, hogy kiolvassuk például az én nevemet és beosztásomat egy kicsit trükközni kell:

```
[25] PS I:\>"$($colResults[0].properties.displayname) az én nevem, beoszt
ásom $($colResults[0].properties.title)"
Soós Tibor az én nevem, beosztásom műszaki igazgató
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Megjegyzés

A PowerShell ténykedésem során ez a második eset, amikor kis-nagybetű érzékenységet tapasztaltam! (Az első az LDAP:: kifejezésnél volt, de ez félig-meddig betudható az ADSI örökségnek.) A második ez: ha `$colResults[0].properties.displayName`-et írok (nagy „N” az utolsó tagban), akkor nem kapok semmit. Ez azért is furcsa, mert eredetileg a címtárban nagy az „N”.

A következő példában kikeresem a sémából az összes olyan tulajdonság sémaelemet, amely a globális katalógusba replikálódik:

```
$strFilter =
"(&(objectCategory=attributeSchema)(isMemberOfPartialAttributeSet=TRUE))"

$objRoot = [ADSI] "LDAP://CN=Schema,CN=Configuration,DC=iqjb,DC=w08"
$objSearcher = New-Object System.DirectoryServices.DirectorySearcher
$objSearcher.SearchRoot = $objRoot
$objSearcher.PageSize = 1000
$objSearcher.Filter = $strFilter
$objSearcher.SearchScope = "Subtree"

$colProplist = "name"
foreach ($i in $colPropList){$objSearcher.PropertiesToLoad.Add($i)}

$colResults = $objSearcher.FindAll()

foreach ($objResult in $colResults)
    {
     $objItem = $objResult.Properties
     "$($objItem.item('name'))"
    }

$objSearcher.Dispose()
$colResults.Dispose()
```

Itt annyival gazdagítottam a keresőm tulajdonságait, hogy meghatároztam a maximális találatok számát (`PageSize`) és a találati listába betöltött elemek tulajdonságainak listáját (`PropertiesToLoad`), ami itt most csak az elem neve. A szkriptemben az első `foreach` ciklust tulajdonképpen feleslegesen használtam, hiszen csak egy tulajdonságot (`name`) töltöttem a kereső „`PropertiesToLoad`” képességébe, de én ezt a szkriptet sokszor olyankor is fel akarom használni kevés változtatással, amikor több tulajdonságot is meg akarok kapni a keresés eredményében.

A kiíratásnál most a hashtáblás stílussal hivatkoztam a „`name`” tulajdonságra, és a folyamat végén, memóriatakarékossági okokból, eldobom a kereső és a találati lista objektumokat.

A következő példa függvényével felhasználói fiókok tetszőleges attribútumát lehet tömegesen lecserélni valami másra. A kód megfejtését az előzőek ismeretében az olvasóra bízom. Csak egy kis segítséget adok: a középrészen található `If` vizsgálat `Else` ágában azt érem el, hogy ha a kicserélendő attribútum érték üres, azaz azt szeretnénk, hogy a ki nem töltött attribútumokat töltsük ki, akkor ez az LDAP filterben a `!$Attr=*` kifejezést kell szerepeltetni, ennek az a jelentése, hogy *„az $Attr változó által jelzett attribútum nem egyenlő akármi, azaz van értéke*”.

```
function ModifyUserAttrib
{
```

```
param (
$domain =
     [System.DirectoryServices.ActiveDirectory.Domain]::getcurrentdomain().Name
,
$Attr = $(throw "Melyik attribútumot?"),
$sValue = $null,
$cValue = $(throw "Mire változtassam?")
)

$root= [ADSI] "LDAP://$domain"
$Searcher = New-Object DirectoryServices.DirectorySearcher
$Searcher.SearchRoot = $root
if($sValue)
{
     $buildFilter = "(&(objectClass=user)($Attr=$sValue))"
}
else
{
     $buildFilter = "(&(objectClass=user)(!$Attr=*))"
}
$Searcher.Filter = $buildFilter
$users = $searcher.findall()
Foreach ($i in $users)
{
     $dn=$i.path
     $user = [ADSI] $dn
     write-host $dn
     $user.Put($Attr,$cValue)
     $user.SetInfo()
}
}
```

3.5.10.1 Keresés idő típusú adatokra

A címtárban nem csak szöveges adatok vannak, hanem például dátum típusúak is. Ezekre nem triviális a keresés. Például keresem az utóbbi 2 napban módosított AD felhasználói objektumokat:

```
PS C:\> $tól=get-date ((get-date).AddDays(-2)) -Format yyyyMMddHHMMss.0Z
PS C:\> $tól
20080530110514.0Z
PS C:\> $searcher = New-Object directoryservices.directorysearcher
PS C:\> $searcher.searchroot = [ADSI] ""
PS C:\> $searcher.filter = "(&(objectCategory=person)(objectClass=User)(when
Changed>=$tól))"
PS C:\> $result = $searcher.findall()
PS C:\> $result

Path                                    Properties
----                                    ----------
LDAP://CN=János Vegetári,OU=Demó,D...   {samaccounttype, lastlogon, dscore...
```

Az egészben a lényeg a `$tól` változó generálása. Látjuk, hogy egy speciális formátumra kell hozni a dátumot: ÉÉÉÉHHNNÓÓPPMM.0Z. Ilyen formázást szerencsére a `get-date format` paraméterével könnyen elvégezhetünk.

Sajnos nem mindig ilyen egyszerű a helyzetünk, hiszen néhány dátumot jelző attribútum nem ilyen formátumban tárolja az időt, hanem un. „tick”-ekben, ketyegésekben. Ez 1601. január 1. 0:00 időponttól eltelt

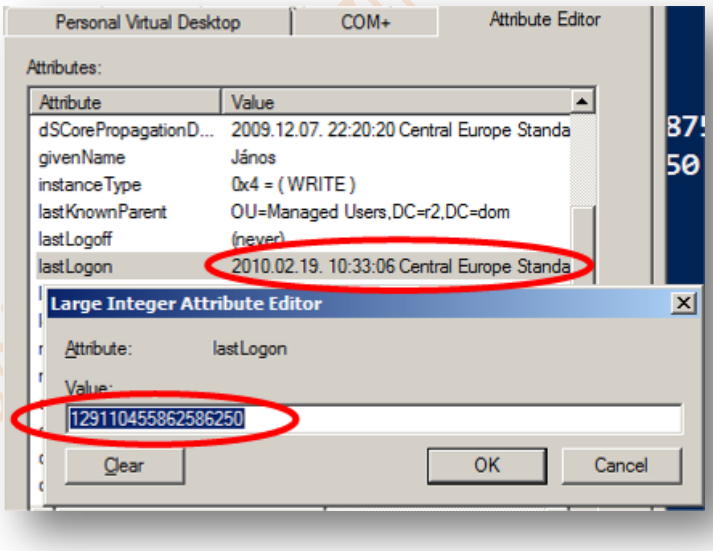
100 nanoszekundumokat jelenti, mindez *long-integer* formátumban. Ilyen attribútum például a *lastLogon*, *lastLogonTimestamp*, *lastLogoff*, *pwdLastSet*. Ha ilyen attribútumok valamely értékére akarunk rákeresni, akkor elő kell tudnunk állítani ezt az értéket. Szerencsére a .NET keretrendszer ebben is segít. Elsőként nézzük, hogy dátumból hogyan tudunk ilyen *long-integer*-t előállítani:

```
[5] PS C:\> $most = get-date
[6] PS C:\> $most

2008. június 1. 20:49:41
[7] PS C:\> $most.ticks
633479501812656250
```

Látjuk, hogy elég a `ticks` tulajdonságát meghívni a dátumnak. Nézzük visszafele, azaz *long-integer*-ből hogyan kapunk dátumot? Ez sem nagyon bonyolult:

```
[8] PS C:\> $MyLongDate = 633479501812656250
[9] PS C:\> $mydate = New-Object datetime $MyLongDate
[10] PS C:\> $mydate

2008. június 1. 20:49:41
```

Egyszerűen kell kreálni egy dátum típusú objektumot, az objektum konstruktorának kell átadni ezt a számot és ebből a .NET létrehozza a dátumot. Azonban vigyázzunk! Az Active Directory-ban más az időszámítás kezdete a dátum típusú adatmezőknél:



85. ábra Az AD ketyegései

A fenti ábrán látható, hogy itt egy 2010-es dátum hány ketyegésnek felel meg, ez eltér a PowerShell által kezelt dátumformátumétól. Az átváltást a következő függvénnyel oldhatjuk meg, ráadásul még az időzónát is bele kell kalkulálni:

```
function convert-ADTicks
{
    param ($object)
```

```
    $offset = ([datetime] "1601.01.01").ticks

    if($object -is [datetime]){ $object.ticks-$offset -
        [system.timezone]::CurrentTimeZone.getutcoffset($object).ticks
    }
    elseif($object -is [system.int64]) {[datetime] ($object + $offset +
        [system.timezone]::CurrentTimeZone.getutcoffset([datetime]
$object).ticks
    )}
}
```

Ez már oda-vissza jól számolja az AD ketyegéseket:

```
[55] PS C:\> convert-ADTicks 129110455862586250

2010. február 19. 10:33:06


[56] PS C:\> convert-ADTicks ([datetime]::parse("2010. február 19. 10:33:06"))
129110455860000000
```

Persze az dátumból történő konvertálásnál elveszítünk néhány ketyegést, de nem olyan vészes a helyzet, hiszen ez a különbség mindenképpen kevesebb lesz, mint 1 másodperc:

```
[59] PS C:\> ([timespan] (129110455862586250 - 129110455860000000)).totalsecond
s
0,258625
```

3.5.10.2 Keresés bitekre

A másik nem triviális eset, hogy bizonyos attribútumokban tárolt szám egyes bitjei jelentenek valamit. Például a felhasználói fiók *userAccountControl* attribútumának különböző bitjei a következőket jelentik:

Jellemző	userAccountControl bit	2^x
ACCOUNT DISABLED	2	1
PASSWORD NOT REQUIRED	32	5
PASSWORD NEVER EXPIRES	65536	16
SMARTCARD REQUIRED	262144	18
ACCOUNT TRUSTED FOR DELEGATION	524288	19
ACCOUNT CANNOT BE DELEGATED	1048576	20

Itt a feladat az, hogy olyan vizsgálatot végezzünk a szűrőben, amellyel csak egy adott bit értékére keresünk rá. Szerencsére van erre a célra két un. vezérlő (control) az AD-ben, ami ezt a célt szolgálja:

- 1.2.840.113556.1.4.803 – bit szintű AND szabály
- 1.2.840.113556.1.4.804 – bit szintű OR szabály

Hogyan kell ezeket használni? Nézzük például a hatástalanított felhasználói fiókokat:

```
(&(objectCategory=person)(objectClass=user)(userAccountControl:1.2.840.113556.1
.4.803:=2))
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Ebben a példában az utolsó feltétel azt jelenti, hogy a `userAccountControl` 2-es bitje helyén 1 van. Azaz „összeéselem” 2-vel az attribútum értékét, a többi bit nem érdekel engem, így az adott feltétel akkor értékelődik ki igazzá, ha ott 1-es szerepel.

Nézzünk egy olyan példát, hogy keresem a Global és Domain Local biztonsági csoportokat:

```
(&(groupType:1.2.840.113556.1.4.804:=6)(groupType:1.2.840.113556.1.4.803:=2147483648))
```

Itt az első feltételelem akkor értékelődik igazzá, ha vagy a 2-es, vagy a 4-es bit helyén szerepel 1-es, ez a két bit jelzi a Globális és Domain Local csoporttípust.

A második feltétel az már egy AND szabály, az azt vizsgálja, hogy a csoport biztonsági típusú.

3.5.11 Objektumok törlése

Az objektumok törlésének menete nagyon hasonlatos a létrehozásukhoz. Egy felhasználó törlése például így néz ki, ha már megragadtuk a felhasználói fiókot a `$user` változóba:

```
PS C:\> $user.psbase.parent.delete("user","cn=$($user.cn)")
```

A `$user.psbase.parent` megadja a szülő konténert, ezen belül most a `Create` helyett a `Delete` metódust kell meghívni. Miután már van `$user`, ezért legegyszerűbb, ha rögtön ebből olvassuk ki a *relative distinguished* nevét.

Ha nincs megragadva a felhasználó, akkor rá is kereshetünk, és utána törölhetjük:

```
PS C:\> $t = "tt"
PS C:\> $searcher = New-Object directoryservices.directorysearcher
PS C:\> $searcher.searchroot = [ADSI] ""
PS C:\> $searcher.filter = "(&(objectclass=user)(sAMAccountName=$t))"
PS C:\> $tuser = $searcher.findone()
PS C:\> $user=$tuser.getdirectoryentry()
PS C:\> $user.psbase.parent.delete("user","cn=$($user.cn)")
```

Itt most a *pre-Windows 2000* név, azaz a *samaccountname* („tt”) alapján kerestem rá a felhasználóra. Megint oda kell figyelni, hogy a keresés eredménye még nem közvetlenül `DirectoryEntry` típusú objektum, ezért kell az utolsó előtti sorban a találatból igazi felhasználói fiókot konvertálni. A törlés maga már a megismert módon megy.

3.5.12 AD objektumok hozzáférési listájának kezelése

Miután jelenleg az AD adatbázisa PSDrive-ként nem elérhető a PowerShell 1.0-ban, így kicsit körülményesebb a hozzáférési lista kezelése. Ráadásul az ADSI objektumoknál láttuk, hogy a PowerShell adaptációs rétege sok tulajdonságot és metódust elrejt, például épp a hozzáférési listával kapcsolatos adatokat, így itt is szükségünk van a `PSBase` nézet használatára.

Elsőként nézzük, hogy egy konkrét AD objektum, jelen esetben egy felhasználó hozzáférési listáját hogyan lehet kiolvasni:

```
[PS] C:\>$u= [ADSI] "LDAP://cn=János Vegetári,OU=Demó,DC=adatum,DC=com"
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
[PS] C:\>$acl = $u.psbase.objectsecurity
[PS] C:\>$acl.GetAccessRules($true,$true,[System.Security.Principal.Security
Identifier])


ActiveDirectoryRights : GenericRead
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : S-1-5-10
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
...
```

A `$acl` változó tartalmazná a hozzáférési listát, de valójában ezt nem tudjuk közvetlenül kiolvasni egy propertyből, hanem egy metódust kell meghívni (`GetAccessRules`), hogy emberi fogyasztásra alkalmas információkhoz jussunk. Ezen metódusnak paramétereket kell átadni a következők szerint:

```
PS C:\> $acl.GetAccessRules.Value


MemberType          : Method
OverloadDefinitions : {System.Security.AccessControl.AuthorizationRuleColle
                      ction GetAccessRules(Boolean includeExplicit, Boolean
                       includeInherited, Type targetType)}
TypeNameOfValue     : System.Management.Automation.PSMethod
Value               : System.Security.AccessControl.AuthorizationRuleCollec
                      tion GetAccessRules(Boolean includeExplicit, Boolean
                      includeInherited, Type targetType)
Name                : GetAccessRules
IsInstance          : True
```

Az első bool paraméterrel lehet szabályozni, hogy kíváncsi vagyok-e az explicit hozzáférési bejegyzésekre, a második bool paraméter szabályozza, hogy kíváncsi vagyok-e az örökölt hozzáférési bejegyzésekre. A harmadik paraméter a zűrösebb, azzal lehet szabályozni, hogy a metódus kimenete milyen formátumú legyen. Én a fenti példában SID-ekre kértem, hogy fordítsa le a bejegyzéseket. Ez nem biztos, hogy jól értelmezhető, így használhatunk egy másik paraméterezési formát is, amellyel a felhasználói fiókok neveit kapjuk vissza a hozzáférési bejegyzésekben:

```
[PS] C:\>$acl.GetAccessRules($true,$true,[System.Security.Principal.NTAccoun
t])


ActiveDirectoryRights : GenericRead
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SELF
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

```
...
```

A tulajdonosra is szükségünk lehet, ahhoz a `GetOwner` metódussal juthatunk hozzá, szintén használhatnánk akár a SID-es formátumot is, de talán itt is gyakoribb az olvasmányosabb nevek használata:

```
[PS] C:\>$acl.GetOwner([System.Security.Principal.NTAccount])

Value
-----
ADATUM\Judy
```

Azért álljon itt emlékeztetőül a SID-es formátum listázása is:

```
[PS] C:\>$acl.GetOwner([System.Security.Principal.SecurityIdentifier])

             BinaryLength AccountDomainSid         Value
             ------------ ----------------         -----
                       28 S-1-5-21-150787130-28... S-1-5-21-150787130-28...
```

A hozzáférési listák módosításához szintén számos metódust tudunk segítségül hívni:

```
ModifyAccessRule                Method         System.Boolean ModifyAcce...
ModifyAuditRule                 Method         System.Boolean ModifyAudi...
PurgeAccessRules                Method         System.Void PurgeAccessRu...
PurgeAuditRules                 Method         System.Void PurgeAuditRul...
RemoveAccess                    Method         System.Void RemoveAccess(...
RemoveAccessRule                Method         System.Boolean RemoveAcce...
RemoveAccessRuleSpecific        Method         System.Void RemoveAccessR...
RemoveAudit                     Method         System.Void RemoveAudit(I...
RemoveAuditRule                 Method         System.Boolean RemoveAudi...
RemoveAuditRuleSpecific         Method         System.Void RemoveAuditRu...
ResetAccessRule                 Method         System.Void ResetAccessRu...
SetAccessRule                   Method         System.Void SetAccessRule...
SetAccessRuleProtection         Method         System.Void SetAccessRule...
SetAuditRule                    Method         System.Void SetAuditRule(...
SetAuditRuleProtection          Method         System.Void SetAuditRuleP...
SetGroup                        Method         System.Void SetGroup(Iden...
SetOwner                        Method         System.Void SetOwner(Iden...
```

A könyv keretein túlmutat, hogy mindegyiket külön bemutassam, de ezen információk alapján azt hiszem, el lehet indulni.

3.5.13 Összetett feladat ADSI műveletekkel

Végezetül nézzünk egy összetettebb feladatot a Windows Server 2008 tartományi környezetben felmerülő problémára. Egy új lehetőség itt az un. „*Fine grained password policy*”, azaz a tartományi szintnél alacsonyabb szinten beállítható jelszó házirendek létrehozásának lehetősége. Ezzel csak egy probléma van: nem szervezeti egységekre, hanem biztonsági csoportokra tudjuk ezeket kiosztani. Ha konzisztensek szeretnénk maradni a házirendek tekintetében, akkor érdemes „árnyékcsoportokat” létrehozni, azaz olyan csoportokat, amelyek egy adott szervezeti egység összes felhasználóját tartalmazzák. Erre készítettem ezt a függvényt, amely automatikusan létrehozza „*shadow*” névelőtaggal és az OU nevével mint utótaggal a csoportot az adott OU-ban, ha még nincs ilyen. Belerakja az összes olyan felhasználót ebbe a csoportba, akik még nem tagjai.

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Paraméterként egy OU *distinguished name* adatát kell megadni. Ha egy felhasználót kitörlünk az adott OU-ból, akkor azt az AD automatikusan kiszedi a shadowgroup-ból, viszont ha csak átmozgatjuk a felhasználót egy másik tárolóba, akkor annak a csoport tagjaiból való eltávolításáról nekünk kell gondoskodni. Ezt hajtja végre a szkript vége.

Rendszeresen futtatva szinkronban tarthatjuk az árnyékcsoportok tagságát az OU felhasználói objektumaival.

```
function update-shadowgroup ([string] $ou = "OU=Demó 2,DC=iqjb,DC=w08")
{
     $adou = [ADSI] "LDAP://$ou"

     $query = new-object system.directoryservices.directorysearcher
     $query.SearchScope = "OneLevel"
     $query.SearchRoot = $adou
     $query.filter = "(&(objectCategory=group)(name=shadow-$($adou.name)))"
     $sg = $query.FindOne()
     if(-not $sg)
     {
           $ADS_GROUP_TYPE_GLOBAL_GROUP = 0x00000002
           $ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP = 0x00000004
           $ADS_GROUP_TYPE_LOCAL_GROUP = 0x00000004
           $ADS_GROUP_TYPE_UNIVERSAL_GROUP = 0x00000008
           $ADS_GROUP_TYPE_SECURITY_ENABLED = 0x80000000

           $groupType = $ADS_GROUP_TYPE_SECURITY_ENABLED -bor
$ADS_GROUP_TYPE_GLOBAL_GROUP
           $sg = $adou.Create("Group", "CN=shadow-$($adou.name)")
           Write-Host "Creating group: shadow-$($adou.name)..."
           $sg.Put("groupType", $groupType)
           $sg.SetInfo()
     }
     else
     {
           $sg = [ADSI] $sg.Path
     }
     $query.SearchScope = "OneLevel"
     $query.SearchRoot = $adou
     $query.filter = "(objectCategory=user)"
     $users = $query.FindAll()

     foreach($user in $users)
     {
           if($user.properties.memberof -notcontains $sg.distinguishedname)
                 {
                       $sg.member += $user.properties.distinguishedname
                       write-host "Inserting $($user.properties.name)..."
                 }
     }
     $sg.setinfo()

     $members = $sg.member
     foreach($member in $members)
     {
           if(-not $member.Contains(",OU=$($adou.name),"))
           {
                 write-host "Removing $member..."
                 $sg.Remove("LDAP://$member")
                 $sg.setinfo()
           }
```

```
        }
}
```

A könyv tanfolyami felhasználása nem engedélyezett!

3.6 Hasznos linkek

Az alábbi linklista kb. 2 év gyűjtőmunkájának eredménye. Ez a könyv nem jöhetett volna létre, ha ezek az információk nem állnak rendelkezésemre. Természetesen nem az itteni információkat emeltem át egy az egyben, hanem ezek a weboldalak inspiráltak arra engem, hogy bizonyos témákat járjak jobban körül, illetve ötleteket adtak arra vonatkozólag, hogy milyen témákat érintsek.

Sok ezen linkek közül nem statikus oldal, hanem dinamikus tartalom, blog, folyóirat jellegű oldal, így érdemes ezeket rendszeresen látogatni.

3.6.1 PowerShell 1.0 linkek

PowerShell QuickStart:

http://channel9.msdn.com/wiki/default.aspx/Channel9.WindowsPowerShellQuickStart

PowerShell Wiki:

http://channel9.msdn.com/wiki/default.aspx/Channel9.WindowsPowerShellWiki

Scripting Ezine:

http://www.computerperformance.co.uk/ezine/

Tip of the Week (Script Center):

http://www.microsoft.com/technet/scriptcenter/resources/pstips/archive.mspx

Gyorsbillentyűk:

http://www.microsoft.com/technet/scriptcenter/topics/winpsh/manual/hotkeys.mspx

.NET formázási kifejezések:

http://msdn2.microsoft.com/en-us/library/fbxft59x(vs.71).aspx

Egyedi objektum, add-member:

http://thepowershellguy.com/blogs/posh/rss.aspx?Tags=PsObject&AndTags=1
http://blog.sapien.com/index.php/2008/02/20/how-can-i-write-a-powershell-function-that-outputs-a-table/
http://technet.microsoft.com/en-us/library/bb978596.aspx

Egyedi típus:

http://msdn2.microsoft.com/en-us/library/cc136149(VS.85).aspx
http://technet.microsoft.com/en-us/library/bb978568.aspx
http://blogs.msdn.com/powershell/archive/2006/06/24/645981.aspx
http://www.hanselman.com/blog/MakingJunctionsReparsePointsVisibleInPowerShell.aspx

A könyv tanfolyami felhasználása nem engedélyezett!

Regex:

http://msdn2.microsoft.com/en-us/library/hs600312.aspx
http://www.regular-expressions.info/
http://regexlib.com/
http://www.weitz.de/regex-coach/
http://www.sellsbrothers.com/tools/#regexd
http://www.ultrapico.com/ExpressoDownload.htm

Keresés az Active Directory-ban ([ADSI]):

http://www.microsoft.com/technet/scriptcenter/resources/qanda/nov06/hey1109.mspx
http://blogs.technet.com/benp/archive/2007/03/26/searching-the-active-directory-with-powershell.aspx
http://www.computerperformance.co.uk/powershell/powershell_active_directory.htm
http://www.microsoft.com/technet/scriptcenter/topics/winpsh/searchad.mspx

WMI:

http://www.microsoft.com/technet/scriptcenter/topics/msh/mshandwmi.mspx
http://www.codeplex.com/PsObject/Wiki/View.aspx?title=WMI&referringTitle=PSH%20Community%20Guide

PowerShell blog:

http://poshoholic.com/
http://blogs.microsoft.co.il/blogs/ScriptFanatic/

ADSI:

http://blogs.technet.com/industry_insiders/pages/windows-server-2008-protection-from-accidental-deletion.aspx
http://msdn.microsoft.com/en-us/library/ms256752(VS.80).aspx
http://bsonposh.com/archives/tag/adsi
http://www.leadfollowmove.com/archives/powershell/managing-group-membership-in-active-directory-with-powershell-part-1
http://www.leadfollowmove.com/archives/powershell/managing-group-membership-in-active-directory-with-powershell-part-2
http://technet.microsoft.com/en-us/magazine/cc162323.aspx
http://powershelllive.com/blogs/lunch/archive/2007/04/04/day-6-adsi-connecting-to-domains-computers-and-binding-to-objects.aspx

PowerShell on Windows Server 2008 Server Core:

http://dmitrysotnikov.wordpress.com/2008/05/15/powershell-on-server-core/

Effective PowerShell:

http://keithhill.spaces.live.com/?_c11_BlogPart_BlogPart=blogview&_c=BlogPart&partqs=cat%3dEffective%2bPowerShell

A könyv tanfolyami felhasználása nem engedélyezett!

ACL-ek kezelése:

http://www.microsoft.com/technet/scriptcenter/resources/pstips/may08/pstip0516.mspx

Performance Monitor:

http://www.leeholmes.com/blog/AccessingPerformanceCountersInPowerShell.aspx

LDAP filter:

http://www.tek-tips.com/faqs.cfm?fid=5667

DateTime típus a .NET keretrendszerben:

http://www.meshplex.org/wiki/C_Sharp/Dates_and_Times

COM Automation:

http://blogs.msdn.com/jmanning/archive/2007/01/25/using-powershell-for-outlook-automation.aspx
http://www.computerperformance.co.uk/powershell/powershell_com.htm
http://www.microsoft.com/technet/scriptcenter/resources/pstips/nov07/pstip1130.mspx

3.6.2 PowerShell 2.0 linkek

Az alábbiakban a 2. kiadáshoz felhasznált, jórészt a PowerShell 2.0 újdonságaira fókuszáló linkek találhatók:

Általános PowerShell információk

http://powershell.com/cs/blogs/ebook/archive/2009/03/30/chapter-12-command-discovery-and-scriptblocks.aspx
http://www.leeholmes.com/blog/default,date,2009-04-20.aspx
http://keithhill.spaces.live.com/?_c11_BlogPart_pagedir=Next&_c11_BlogPart_handle=cns!5A8D2641E0963A97!6930&_c11_BlogPart_BlogPart=blogview&_c=BlogPart
http://www.reskit.net/powershell/index.htm
http://www.codedigest.com/Tutorials/Windows-PowerShell-Commands1.aspx
http://www.jonathanmedd.net/

PS Toolbox:

http://www.microsoft.com/technet/scriptcenter/topics/winpsh/toolbox.mspx
http://code.msdn.microsoft.com/PowerShellPack
http://www.powershelltoys.com/

Windows Management Framework

http://support.microsoft.com/default.aspx/kb/968929

A könyv tanfolyami felhasználása nem engedélyezett!

Remoting Permissions:

http://msgoodies.blogspot.com/2009/09/using-ps-session-without-having.html

IE Automation

http://msdn.microsoft.com/en-us/magazine/cc337896.aspx

2.0 újdonságok:

http://blogs.technet.com/heyscriptingguy/archive/2009/10/28/hey-scripting-guy-october-28-2009.aspx

Try, Catch, Finally

http://blogs.msdn.com/powershell/archive/2009/06/17/traps-vs-try-catch.aspx
http://blogs.mssqltips.com/blogs/chadboyd/archive/2008/10/04/try-catch-finally-with-powershell.aspx
http://tfl09.blogspot.com/2009/01/error-handling-with-powershell.html

Modules:

http://msdn.microsoft.com/en-us/library/dd878324(VS.85).aspx
http://tfl09.blogspot.com/2009/01/modules-in-powershell-v2.html
http://get-powershell.com/2008/12/23/module-manifests-in-ctp3/
http://msmvps.com/blogs/richardsiddaway/archive/2009/02/24/powershell-modules-ii.aspx
http://chrisfederico.wordpress.com/2009/05/27/introduction-to-powershell-modules-w-advanced-functions/
http://richardsiddaway.spaces.live.com/Blog/cns!43CFA46A74CF3E96!2106.entry
http://huddledmasses.org/powershell-modules/
http://huddledmasses.org/powershell-modules-metadata-and-mysteries/

.NET fogalmak:

http://prog.hu/cikkek/860/Bevezetes+a+NET-be/oldal/6.html

Advanced Functions:

http://bartdesmet.net/blogs/bart/archive/2008/03/22/windows-powershell-2-0-feature-focus-script-cmdlets.aspx
http://get-powershell.com/category/advanced-function/
http://blogs.msdn.com/powershell/archive/2008/12/23/advanced-functions-and-test-leapyear-ps1.aspx
http://poshoholic.com/2009/09/18/powershell-3-0-why-wait-importing-typed-objects-with-typed-properties-from-a-csv-file/
http://blogs.msdn.com/powershell/archive/2009/01/11/test-pscmdlet.aspx
http://blogs.msdn.com/mediaandmicrocode/archive/2009/04/10/microcode-powershell-scripting-trick-fun-with-parameter-binding-the-fake-parameter-set-trick.aspx
http://get-powershell.com/2009/01/06/advanced-functions-support-confirm/

DynamicParameters:

http://www.powergui.org/thread.jspa?threadID=10607

A könyv tanfolyami felhasználása nem engedélyezett!

http://poshoholic.com/2007/11/28/powershell-deep-dive-discovering-dynamic-parameters/

Cmdlet mélységek:

http://msdn.microsoft.com/en-us/library/ms714433(VS.85).aspx

Should Process V2 előtt (jó a megértéshez):

http://blogs.msdn.com/powershell/archive/2007/02/25/supporting-whatif-confirm-verbose-in-scripts.aspx

Proxy functions:

http://blogs.msdn.com/powershell/archive/2009/01/04/extending-and-or-modifing-commands-with-proxies.aspx
http://get-powershell.com/2009/01/04/using-proxy-commands-in-powershell/
http://poshcode.org/1606
http://poshcode.org/?show=785
http://blogs.msdn.com/powershell/archive/2009/03/13/dir-a-d.aspx

Sok CTP2-es cikk:

http://community.bartdesmet.net/blogs/bart/archive/category/43.aspx

Add-Type:

http://msmvps.com/blogs/richardsiddaway/archive/2009/12/05/creating-objects.aspx
http://thepowershellguy.com/blogs/posh/archive/2008/06/02/powershell-v2-ctp2-making-custom-enums-using-add-type.aspx
http://get-powershell.com/category/add-type/
http://episteme.arstechnica.com/eve/forums/a/tpc/f/6330927813/m/683003352041
http://dougfinke.com/blog/index.php/2009/08/15/inject-dynamic-wpf-guis-into-your-powershell-pipeline/

SQL, Access:

http://msmvps.com/blogs/richardsiddaway/

Outlook PowerShell:

http://blogs.technet.com/heyscriptingguy/archive/2009/12/16/hey-scripting-guy-december-16-2009.aspx

AD PowerShell:

http://blogs.msdn.com/adpowershell/
http://blogs.msdn.com/adpowershell/archive/2009/05/19/tab-completing-ldap-attribute-names-inside-advanced-filters.aspx
http://blogs.msdn.com/adpowershell/archive/2009/09/22/how-to-find-extended-rights-that-apply-to-a-schema-class-object.aspx

LDAP syntax

http://msdn.microsoft.com/en-us/library/aa746475(VS.85).aspx

A könyv tanfolyami felhasználása nem engedélyezett!

SearchFlags:

http://msdn.microsoft.com/en-us/library/ms679765(VS.85).aspx

SystemFlags:

http://msdn.microsoft.com/en-us/library/ms680022(VS.85).aspx

FormatData:

http://blogs.msdn.com/powershell/archive/2006/04/30/how-powershell-formatting-and-outputting-really-works.aspx
http://blogs.technet.com/heyscriptingguy/archive/2009/06/21/hey-scripting-guy-event-6-solutions-from-expert-commentators-beginner-and-advanced-the-110-meter-hurdles.aspx

Stop pipeline:

http://powershell.com/cs/blogs/tobias/archive/2010/01/13/cancelling-a-pipeline.aspx

Pipeline begin, process, end order:

http://jdhitsolutions.com/blog/2010/01/potential-pipeline-pitfall/#utm_source=feed&utm_medium=feed&utm_campaign=feed

AD UAC:

http://bsonposh.com/archives/category/powershell/page/3
http://www.computerperformance.co.uk/ezine/ezine23b.htm

Eseménynapló:

http://www.jonathanmedd.net/2009/12/powershell-2-0-one-cmdlet-at-a-time-12-write-eventlog.html

WMI:

http://blogs.msdn.com/powershell/archive/2009/08/30/exploring-wmi-with-powershell-v2.aspx
http://technet.microsoft.com/en-us/library/ee198937.aspx
http://tfl09.blogspot.com/2009/05/powershell-and-wmi-namespaces.html
http://itknowledgeexchange.techtarget.com/powershell/
http://msmvps.com/blogs/richardsiddaway/archive/2009/08/13/system-up-time.aspx

WQL Referencia:

http://msdn.microsoft.com/en-us/library/aa394606(VS.85).aspx

WMIEvents:

http://msmvps.com/blogs/richardsiddaway/archive/2009/11/07/powershell-wmi-events.aspx
http://powershellcommunity.org/Blogs/CommunityBlogs/tabid/55/EntryID/43/Default.aspx
http://trevorsullivan.wordpress.com/2009/11/16/powershell-getting-started-with-wmi-events/

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

Eventing:

http://blogs.msdn.com/powershell/archive/2008/06/11/powershell-eventing-quickstart.aspx
http://richardsiddaway.spaces.live.com/blog/cns!43CFA46A74CF3E96!2598.entry
http://poshcode.org/859
http://www.nivot.org/2009/10/09/PowerShell20AsynchronousCallbacksFromNET.aspx
http://www.networkworld.com/community/node/36616
http://jdhitsolutions.com/blog/2009/09/powershell-exit-stage-left/
http://www.networkworld.com/community/node/36616
http://dmitrysotnikov.wordpress.com/2008/06/23/powergui-teched-demo/
http://www.leeholmes.com/blog/PowerShellAudioSequencer.aspx

Exchange powershell:

http://www.exchange-powershell.com/

Search:

http://blogs.technet.com/heyscriptingguy/archive/2007/08/27/how-can-i-find-all-the-files-in-a-directory-tree-that-contain-a-specific-word-or-phrase.aspx
http://msdn.microsoft.com/en-us/library/bb266517(VS.85).aspx
http://social.msdn.microsoft.com/Forums/en/windowsdesktopsearchdevelopment/thread/e71042f6-79d7-44a4-afcd-049e242ab6a4
http://www.codeproject.com/KB/vista/wdscmdlet.aspx
http://ochoco.blogspot.com/2008/12/querying-windows-desktop-search.html
http://www.thejoyofcode.com/Using_Windows_Search_in_your_applications.aspx
http://www.codeguru.com/csharp/csharp/cs_data/searching/article.php/c13719/
http://blogs.msdn.com/cheller/archive/2006/06/27/windows-vista-search-syntax-where-are-my-predicates.aspx

CHM help generator:

http://powershellcommunity.org/Default.aspx?tabid=55&EntryID=7

Group Policy:

http://technet.microsoft.com/en-us/library/ee461027.aspx

BreakPoints:

http://blog.usepowershell.com/2010/01/script-injection-with-set-psbreakpoint/
http://www.jonathanmedd.net/2010/02/powershell-2-0-one-cmdlet-at-a-time-59-set-psbreakpoint.html
http://www.jonathanmedd.net/2010/02/powershell-2-0-one-cmdlet-at-a-time-60-get-psbreakpoint.html
http://msgoodies.blogspot.com/2010/02/invoking-powershell-debugger-from.html
http://blogs.msdn.com/powershell/archive/2009/07/13/advanced-debugging-in-powershell.aspx
http://blogs.msdn.com/powershell/archive/2009/01/19/debugging-powershell-script-using-the-ise-editor.aspx

A könyv tanfolyami felhasználása nem engedélyezett!

Excel:

http://blogs.technet.com/heyscriptingguy/archive/2006/09/08/how-can-i-use-windows-powershell-to-automate-microsoft-excel.aspx
http://kentfinkle.com/PowershellAndExcel.aspx
http://www.microsoft.com/technet/scriptcenter/resources/pstips/nov07/pstip1130.mspx
http://www.eggheadcafe.com/software/aspnet/30217514/excel-automation.aspx
http://stackoverflow.com/questions/687891/exception-automating-excel-2007-with-powershell-when-calling-workbooks-add
http://wouter.shush.com/2007/08/excel-automation-with-powershell
http://forums.msexchange.org/m_1800511877/tm.htm
http://msmvps.com/blogs/richardsiddaway/archive/2010/01/03/excel-2010-and-powershell.aspx
http://msdn.microsoft.com/en-us/library/bb241279.aspx
http://msdn.microsoft.com/en-us/library/h1e33e36(VS.80).aspx

Access:

http://technet.microsoft.com/en-us/magazine/2009.05.scriptingguys.aspx?pr=blog
http://allenbrowne.com/func-DDL.html
http://bytes.com/topic/access/answers/467786-how-create-autonumber-field-sql-statement
http://msdn.microsoft.com/en-us/library/aa140015(office.10).aspx
http://allenbrowne.com/ser-49.html
http://www.chat11.com/Microsoft_Access_SQL

ACL:

http://www.microsoft.com/technet/scriptcenter/resources/pstips/may08/pstip0516.mspx
http://powershellcommunity.org/Forums/tabid/54/aff/19/aft/609/afv/topic/Default.aspx
http://blogs.msdn.com/johan/archive/2008/10/01/powershell-editing-permissions-on-a-file-or-folder.aspx
http://blogs.technet.com/heyscriptingguy/archive/2009/09/14/hey-scripting-guy-september-14-2009.aspx

A könyv tanfolyami felhasználása nem engedélyezett!

4.Tárgymutató

S

— 153

!

! ld. -not

#

214

$

$() 155, 168
$_ 55
$args 182, 211
$ConfirmPreference 350
$DebugPreference 326
$false 151
$foreach 169
$host 274
$input 198
$matches 135
$MyInvocation 266, 267
$null 151
$ofs 93
$PSBoundParameters 271
$pscmdlet 343
$switch 177
$this 101, 421
$true 151
$VerbosePreference 325

%

% 23, 124

&

& 50

(

() 154

,

, 70, 158

.

. 159

.. 77, 158

:

:: 94, 159

;

; 21, 52

@

@() 71, 155
@{*Lásd* hashtábla
@" 49

[

[] 157
[ADSI] 525
[decimal] 69
[double] 69
[int] 69
[long] 69
[math] 94
[PSObject] ..... Lásd PSObject
[regex] 145
[scriptblock] 200
[switch] 184
[system.convert] 95
[void] 76
[XML] 385

`

` 21, 48

{

{} 156

|

| 53

’

’ 48

”

” 48

A könyv tanfolyami felhasználása nem engedélyezett!

+

++ 153
+= 73

<

<# #> 214

>

> 163
>> 163

0

0x 69

2

2> 323

A,Á

Add-Computer 406
Add-Member 100
Add-PSSnapin 279
Add-Type 110
alias 37

B

begin/process/end 198
break 175, 317

C

Catch 320
Checkpoint-Computer 409
clear 23
Clear-EventLog 395
Clear-Host 23
Clear-Variable 46
-clike 130
cls 23
-cmatch 142
cmdlet 29
cmdletbinding 341
-cnotlike 130
–cnotmatch 143
CommonParameters 306
Compare-Object 239
Complete-Transaction 404
–contains 128
continue 317
ConvertFrom-Csv 247, 248
ConvertFrom-StringData 84
ConvertTo-Csv 248
–creplace 150

Cs

csővezeték 52
–csplit 165

D

DATA 367
DateTime 86
default 175
DefaultDisplayPropertySet 104
digitális aláírás 215
dinamikus paraméter 203
dir 27
DirectoryServices.DirectoryEntry 525
Disabe-PSBreakpoint 337
Disable-ComputerRestore 410
dotsourcing 194, 210
DO-WHILE 167
DynamicParam 359

E,É

ELSE 167
ELSEIF 167
Enable-ComputerRestore 410
Enable-PSBreakpoint 337
Enable-PSRemoting 257
EngineEvent 495
Enter-PSSession 260
env: 276
ErrorAction 306
Escape 48
Exception objektum 319, 324
ExceptionType 313
Exit 15, 58
Exit-PSSession 260
Export-Alias 39
Export-CliXML 118, 243
Export-Console 280
Export-Csv 243, 248
Export-ModuleMember 286

F

-f 161
filter 199
Finally 320
fl *Lásd* Format-List
FOR 168
FOREACH 168
ForEach-Object 54, 224
format-custom 98
Format-List 60
Format-Table 60, 235
Format-Wide 236
ft *Lásd* Format-Table
function: 201

G

Get-Acl 377, 400, 455
Get-ADDomainController 458
Get-ADForest 457
Get-ADGroupMember 443
Get-ADObject 444

A könyv tanfolyami felhasználása nem engedélyezett!

Get-ADUser 437
Get-Alias 37
Get-Command 29, 355
Get-ComputerRestorePoint 409
Get-Content 163, 244, 372
Get-Counter 431
Get-Culture 364
Get-Date 86
Get-Eventlog 387
Get-EventSubscriber 496
Get-ExecutionPolicy 208
Get-FormatData 117
Get-Help 30, 55, 360
Get-History 22, 294
Get-HotFix 408
Get-ItemProperty 398
Get-Member 35, 53, 72
Get-Module 282
Get-PSCallStack 334
Get-PSDrive 39
Get-PSProvider 40
Get-PSSnapin 279
get-random 223
Get-Service 103, 420
Get-Transaction 403
Get-UICulture 365
Get-Unique 241
Get-Variable 46, 190
Get-WinEvent 391
Get-WMIObject 411
GetType() 44
-GroupBy 62
Group-Object 226

H

hashtable 80
help függvény 31
here string 49
hexadecimális 69
hibakezelés 305

I,Í

idézőjelek használata 47
IF 167
Import-Alias 39
Import-CliXML 118
Import-Csv 244, 248
Import-LocalizedData 367
import-module 282
Invoke() 157
Invoke-Command 256
Invoke-Expression 160
Invoke-Item 160
Invoke-WMIMethod 423
–is 152
–isnot 152

J

–join 164

K

kimenet 55
kommandlet ..... *Lásd* cmdlet
komment 214
konzolfájl 280

L

–like 130
Limit-EventLog 395
logikai operátorok
-and 150
-band 151
-bnot 151
-bor 151
-bxor 151
–not 151
-or 150
-xor 150

M

makecert.exe 215
Measure-Command 293
Measure-Object 242
megosztások elérése 385
metódus 27
module manifest 287
moduljegyzék 287
MoveNext() 170
MyCommand 266

N

New-Alias 38
New-Event 495
New-EventLog 395
New-Item 202, 369, 400
New-Itemproperty 400
New-Module 291
New-ModuleManifest 287
New-Object 94, 110
-COM 485
New-PSDrive 41
New-PSSession 258
New-TimeSpan 89
New-Variable 46
nonterminating error 305
–notcontains 128
-notlike 130
–notmatch 143

O,Ó

objektum 27
op_Addition 120
operátor
szétpaszírozás (@) 184
operátorok
-ceq 126
-cge 126
-cgt 126

A könyv tanfolyami felhasználása nem engedélyezett!

© Soós Tibor – soost@ IQJB.hu

-cle ... 126
-clt126
-cne ... 126
-eq126
-ge126
-gt 126
-le 126
-lt 126
-ne126
végrehajtási ... 160
Out-Default ... 114
Out-File ... 243
Out-Gridview ... 237
Out-Host ... 56, 114
Out-Null ... 251
Out-Printer ... 243
output ... 55
Out-String ... 249

Ö,Ő

összehasonlító operátorok ... 126

P

param ... 185, 212
parameter metaadat ... 341
–PassThru ... 101
pipeline ... 52
PowerGUI ... 297
prompt ... 204, 277
>> 21
Provider
képességek ... 40
psbase ... 527
PSDrive ... 39
PSObject ... 99
PSProvider ... 40

R

range ... *Lásd* ..
Read-Host ... 215
Receive-Job ... 252
Reflector ... 95, 302
RegexBuddy ... 302
Register-EngineEvent ... 495
Register-ObjectEvent ... 499
Regular Expression ... 130
Regex ... 130
Remove-Computer ... 406
Remove-EventLog ... 395
Remove-Job ... 254
Remove-Module ... 284
Remove-PSDrive ... 42
Remove-PSSession ... 260
Remove-PSSnapin ... 279
Remove-Variable ... 45
Remove-WMIObject ... 428
Reset() ... 170
Reset-ComputerMachinePassword ... 408
Resolve-Path ... 43
Restart-Computer ... 406
Restore-Computer ... 410
return ... 196, 212

S

–scope ... 190
ScriptBlock ... 157
Search-ADAccount ... 445
Select-Object ... 229, 399
Select-String ... 148, 249
Send-MailMessage ... 295
Set-Acl ... 379
Set-ADUser ... 447
Set-Content ... 243
Set-Date ... 86
Set-ExecutionPolicy ... 206
Set-Item ... 202
Set-ItemProperty ... 400
Set-PSBreakPoint ... 332
Set-PSDebug ... 326
Set-StrictMode ... 328
Set-Variable ... 45, 190
Set-WMIInstance ... 426
Set-WSMandQuickCOnfig ... 258
Show-EventLog ... 395
Sort-Object ... 62, 234
–split ... 164
Split() ... 126
Split-Path ... 267
Start-Job ... 252
Start-Process ... 417
Start-Sleep ... 293
Start-Transaction ... 403
Start-Transcript ... 292, 376
static member ... 94
Stop-Computer ... 406
Stop-Job ... 254
Stop-Process ... 417
Stop-Transcript ... 292
SupportsShouldProcess ... 349
switch
-wildcard ... 176
SWITCH ... 174
System.Collections.ArrayList ... 74
System.Collections.SortedList ... 85
System.Console ... 472
System.DirectoryServices.ActiveDirectory.ActiveDirectorySite ... 524
System.DirectoryServices.ActiveDirectory.Domain ... 523
System.DirectoryServices.ActiveDirectory.Forest ... 523
System.DirectoryServices.DirectorySearcher ... 535
System.Environment ... 470
System.IO.FileSystemWatcher ... 500
System.IO.Path ... 383
System.Net.Dns ... 523
System.Timers.Timer ... 499

Sz

szkriptblokk ... 200
szótár ... 80

A könyv tanfolyami felhasználása nem engedélyezett!

T

TabExpansion 20, 204
Tee-Object 226
terminating error 305
Test-ComputerSecureChannel 408
Test-Connection 256, 407
Test-Path 369
throw 181, 305
típus 66
ToString() 240
tömb 69
  asszociatív 80
  egyelemű 70
  elem 77
  típusos 79
  többdimenziós 78
tömboperátor 158
Trace-Command 329
trap 311
Try320
tulajdonság 27

Ty

types.ps1xml 103, 421

U,Ú

Undo-Transaction 405
Unregister-Event 496
Update-FormatData 114, 116
Update-List 250

V

változók 43

W

Wait-Job 254
Wait-Process 417
Where-Object 55
WHILE 167
Windows PowerShell ISE 24
–wrap 63
Write-Debug 325, 326
Write-Error 323
Write-EventLog 395
Write-Host 55
Write-Output 55
Write-Progress 295
Write-Verbose 325
Write-Warning 323

A könyv tanfolyami felhasználása nem engedélyezett!