

Dvě knihy
v jedné

Patrik Malina

Jak vyzrát na

Microsoft Windows

PowerShell 2.0

**Rychlý zdroj informací pro
zanepřázdňené administrátory**

Nové cmdlety, proměnné a operátory, spouštění procesů na pozadí

Práce s příkazy a skripty, správa Active Directory

Shrnutí, opakování na koncích témat, otázky a odpovědi

DVD obsahuje:

Další knihu o základech PowerShellu ve formátu PDF

Všechny skripty uvedené v knize

PowerShell Pack, PowerShell 2 SDK, LogParser

PowerGUI, AD Cmdlets a další užitečné nástroje



 **C P R E S S**

Patrik Malina

Jak vyžrát na Windows PowerShell 2.0

**Computer Press, a.s.
Brno
2010**

Jak vyzrát na Windows PowerShell 2.0

Patrik Malina

Computer Press, a. s., 2010. Vydání první.

Jazyková korektura: Alena Láníčková

Vnitřní úprava: Petr Klíma

Sazba: Ctibor Foltýn

Rejstřík: Daniel Štreit

Obálka: Martin Sodomka

Komentář na zadní straně obálky: Libor Pácl

Technická spolupráce: Jiří Matoušek,

Zuzana Šindlerová, Dagmar Hajdajová

Odpovědný redaktor: Libor Pácl

Technický redaktor: Jiří Matoušek

Produkce: Petr Baláš

Computer Press, a. s.,

Holandská 8, 639 00 Brno

Objednávky knih:

<http://knihy.cpress.cz>

distribuce@cpress.cz

tel.: 800 555 513

ISBN 978-80-251-2732-2

Prodejní kód: K1742

Vydalo nakladatelství Computer Press, a. s., jako svou 3583. publikaci.

© Computer Press, a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Obsah

Úvod 11

KAPITOLA 1

PowerShell a jeho svět..... 13

KAPITOLA 2

PowerShell jako software..... 17

Existující implementace a verze PowerShellu	18
Verze CTP a související potíže	19
Správa služby Active Directory.....	21
Instalace PowerShellu	22
Úvod.....	22
Instalace.....	23
Shrnutí	25
Důležité k zapamatování	25
Rozšíření a doplňky	26
Úvod.....	26
Windows System Modules.....	26
Windows 7 PowerShell Pack.....	26
PowerGUI	26
PowerShell Commands for Active Directory.....	27
Windows 7 Troubleshooting Platform	28
Windows PowerShell 2.0 SDK	28
PowerShell Community Extensions.....	28
Shrnutí	28
Důležité k zapamatování	29
Otázky a odpovědi	29

KAPITOLA 3

Práce v prostředí PowerShellu 31

Interaktivní práce	32
Úvod.....	32
Skriptovací blok a tělo skriptu	32
Všechny roury a kanály PowerShellu.....	35
Uchopení příkazového bloku	39
Okolnosti spuštění bloku - \$MyInvocation	46

Shrnutí	47
Důležité k zapamatování	47
Připojení ke vzdáleným počítačům	47
Úvod	47
Průzkum nastavení služby WinRM (WSMan)	49
Rychlá konfigurace služby WinRM	50
Nastavení přístupu ke službě WinRM mimo Active Directory	51
Vymezení cílového počítače pro vzdálené připojení	52
Povolení vzdálené komunikace pro více počítačů	52
Připojení ke vzdálené konzole PowerShellu	54
Hostování služby WSMan jako aplikace IIS – scénář „fan-in“	54
Shrnutí	67
Důležité k zapamatování	67
Hromadné provádění úloh	68
Úvod	68
Paralelní spuštění krátkých úloh na témže počítači	73
Paralelní spuštění větších skriptů na témže počítači	74
Shrnutí	75
Důležité k zapamatování	75
Plánování úloh	76
Úvod	76
Ovládání služby Scheduled Tasks pomocí COM na Windows XP	78
Seznam naplánovaných úloh na Windows XP	81
Ovládání služby Scheduled Tasks pomocí tříd .NET	82
Ovládání programu Schtasks.exe	85
Spuštění úlohy napsané v PowerShellu	86
Shrnutí	88
Důležité k zapamatování	88
Moduly v PowerShellu	88
Úvod	88
Shrnutí	93
Důležité k zapamatování	93
Otázky a odpovědi	93

KAPITOLA 4

Skriptovací jazyk PowerShell 97

Důležité techniky a postupy	98
Úvod	98
Setřídění pole/kolekce	98
Porovnání polí pomocí operátoru -Contains	99
Porovnání polí pomocí příkazu Compare-Object	100
Výběr metody porovnávání polí	101
Výkonnostní aspekty filtrování dat	103

Shrnutí	105
Důležité k zapamatování	105
Efektivní a pokročilé možnosti využití roury	106
Úvod	106
Přímé převedení vlastnosti objektu na řetězec	110
Přímý vstup vlastnosti objektu z roury do příkazu	110
Udržení dat v rouře: přepínač -PassThru	111
Udržení dat v rouře: příkaz Tee-Object	113
Rozšiřování objektů o další členy	114
Tvorba nové třídy objektu	122
Třídění dle jednoho a více kritérií	124
Porovnání objektů	127
Shrnutí	128
Důležité k zapamatování	128
Zpracování textu v PowerShellu	129
Úvod	129
Využití struktury Here-String	132
Základní použití regulárních výrazů	133
Hledání textu a Select-String	135
Zpracování cesty DN objektu v Active Directory	139
Nalezení a extrakce e-mailové adresy z textu	142
Načtení obsahu celého textového souboru	144
Odstranění diakritiky v textu	145
Odstranění diakritiky v textu pomocí regulárních výrazů	147
Shrnutí	149
Důležité k zapamatování	149
Vstup a výstup strukturovaných dat	149
Úvod	149
Import souborů CSV	149
Export/import souboru CLI XML	152
Import obecného souboru XML	153
Shrnutí	156
Důležité k zapamatování	156
Využití funkcí a filtrů	157
Úvod	157
Návrat dat z funkce: hledání v LDAPu	160
Filtr a vstup parametrů	162
Funkce a její obor působnosti (scope)	167
Vytvoření nového cmdletu PowerShellu	169
Pokročilý Cmdlet PowerShellu	170
Shrnutí	174
Důležité k zapamatování	175
Možnosti konstrukce Switch	175
Úvod	175

Větvení SWITCH nad kolekcí objektů	175
Switch a regulární výrazy	176
Switch jako parametr	178
Shrnutí	180
Důležité k zapamatování	181
Chyby, výjimky a jejich zpracování	181
Úvod	181
Přesměrování chybového výstupu	184
Ošetření chyby pomocí bloku Try-Catch-Finally	185
Ošetření chyby pomocí bloku Trap	192
Zachycení jakékoliv případné chyby	194
Rozlišení chyby: problém implementace PowerShellu	194
Rozlišení chyby nepřímým ověřením	196
Ladění skriptu v grafickém rozhraní pomocí zářezek	198
Shrnutí	204
Důležité k zapamatování	204
Spouštění skriptů a bezpečnost	204
Úvod	204
Předání přihlašovacích údajů	204
Jednorázové zadání přihlášení	205
Bezpečný vstup hesla z konzoly	206
Opětovné použití zabezpečeného hesla	207
Shrnutí	208
Důležité k zapamatování	208
Zpracování událostí v PowerShellu	208
Úvod	208
Inventura zastavených procesů	213
Shrnutí	215
Důležité k zapamatování	215
Otázky a odpovědi	216

KAPITOLA 5

Základní správa Windows.....219

Správa služeb ve Windows	220
Úvod	220
Rychlý náhled služeb a jejich stavu	222
Shrnutí	223
Důležité k zapamatování	224
Záznamy událostí – logy	224
Úvod	224
Výběr událostí dle časového intervalu	224
Novinky v načítání obsahu logu	226
Shrnutí	229

Důležité k zapamatování	230
Procesy operačního systému.....	230
Interaktivní spuštění procesu.....	230
Spuštění procesu s povýšením oprávnění.....	232
Spuštění PowerShellu pod právy systému.....	233
Čítače stavu a výkonu	234
Shrnutí	238
Důležité k zapamatování	239
Souborový systém.....	239
Úvod.....	239
Jména v souborovém systému – zpracování Path.....	239
Výběrové kopírování určitých souborů	243
Výběrové kopírování dle převodní tabulky.....	246
Rozdělení seznamu v textovém souboru	249
Porovnání seznamů v textovém souboru.....	251
Shrnutí	253
Důležité k zapamatování	253
Konfigurační databáze Registry	253
Úvod.....	253
Hledání a nahrazení konkrétních hodnot v Registry	256
Shrnutí	260
Důležité k zapamatování	260
Práce s certifikáty	260
Úvod.....	260
Pořízení kořenového certifikátu bez certifikačních autorit	261
Vystavení certifikátu a podpis kódu.....	264
Nastavení důvěryhodného vydavatele	269
Export osobních certifikátů včetně privátních klíčů	270
Shrnutí	271
Důležité k zapamatování	271
Správa lokálních účtů a skupin.....	271
Úvod.....	271
Vytvoření seznamu vypnutých/zapnutých lokálních uživatelských účtů	272
Ovládání lokálních účtů.....	275
Shrnutí	277
Důležité k zapamatování	277
Sítové sdílení složek a tiskáren	278
Úvod.....	278
Přehled sdílených prostředků	278
Mapování sítové jednotky	279
Založení sdíleného zdroje.....	281
Práce s tiskárnami.....	283
Shrnutí	284
Důležité k zapamatování	284

Zálohování ve Windows	285
Úvod	285
Instalace modulu pro zálohování	285
Záloha System State	286
Záloha složky a souborů	288
Shrnutí	290
Důležité k zapamatování	291
Otázky a odpovědi	291

KAPITOLA 6

Správa adresářových služeb Active Directory293

Hledání a načítání objektů	296
Úvod	296
Nalezení uživatele dle hodnoty atributu	297
Zpřesnění hledání uživatelů	306
Určení výstupní sady	311
Export objektů do souborů	313
Totální export dat AD/LDAP	315
Shrnutí	317
Důležité k zapamatování	317
Účty uživatelů	318
Úvod	318
Active Directory Recycle Bin a smazané objekty	324
Shrnutí	325
Důležité k zapamatování	326
Skupiny a členství	326
Úvod	326
Nalezení skupin	326
Nalezení členů skupin	331
Kopírování a replikace členství	332
Shrnutí	336
Důležité k zapamatování	336
Přesuny a hromadné manipulace	336
Úvod	336
Přesuny objektů	337
Hromadný zápis stejné hodnoty atributů	338
Hromadný zápis různých hodnot atributů	345
Tvorba objektů klonováním	349
Hromadná tvorba účtů importem dat	352
Shrnutí	355
Důležité k zapamatování	355
Zabezpečení objektů	355
Úvod	355
Průzkum zabezpečení objektu	356

Přenos zabezpečení objektu.....	360
Shrnutí	363
Důležité k zapamatování	363
Skupinové politiky	363
Úvod	363
Získání přehledu objektů Group Policy	364
Získání přehledu GPO pomocí cmdletů – modul Group Policy	366
Náhled nastavení objektu GPO	368
Založení a přiřazení objektu GPO	371
Záloha objektů GPO	372
Ovládání pomocí rozhraní COM.....	373
Shrnutí	375
Důležité k zapamatování	375
Otázky a odpovědi	376

KAPITOLA 7

PowerShell a síť.....379

Správa síťové konfigurace Windows	380
Úvod	380
Přiřazení více adres IP síťovému adaptéru.....	382
Přiřazení základních parametrů síťového rozhraní.....	384
Záloha a obnova nastavení síťových rozhraní.....	387
Ovládání firewallu Windows	388
Shrnutí	393
Důležité k zapamatování	393
Klienty síťových služeb a protokolů	394
Úvod	394
Zasílání e-mailu pomocí rozhraní CDO	395
Jednorázové odeslání e-mailové zprávy	397
Odesílání e-mailové zprávy s přihlášením k serveru	397
Hromadné odesílání e-mailové zprávy.....	398
Odesílání e-mailové zprávy s přílohami	400
Odeslání zprávy z Outlooku – protokolem MAPI.....	400
Načtení obsahu poštovní schránky Outlooku	401
Vyšetření obsahu zprávy v Outlooku: „nedoručenka“.....	404
Stažení obsahu webové stránky.....	410
Hromadný přenos souborů pomocí služby BITS	411
Průběžné ovládání služby a úloh BITS	414
Jednorázové spuštění úlohy služby BITS	416
Shrnutí	417
Důležité k zapamatování	417
Průzkumy sítě a inventarizace.....	418
Úvod	418
Hromadné prověření dostupnosti síťových klientů	422

Sítové adresy a ověření příslušnosti k podsíti.....	423
Shrnutí.....	428
Důležité k zapamatování.....	428
Otázky a odpovědi.....	428

KAPITOLA 8

Správa dalších aplikací a služeb.....431

Úvod.....	432
Správa aplikačního serveru IIS 7.....	432
Databáze v PowerShellu.....	434
Správa služeb MS Exchange.....	439
MS SharePoint a PowerShell.....	440
Virtual Server, Hyper-V a PowerShell.....	441
Clustering a PowerShell.....	443
Shrnutí.....	445
Otázky a odpovědi.....	445

Příloha.....445

Obsah DVD.....	447
Kniha.....	447
Skripty.....	447
Instalace.....	447
Moduly PowerShellu.....	448
Přehled použitých skupin příkazů.....	449
Informační zdroje.....	452
Oficiální zdroje.....	452
Aplikovaný PowerShell.....	452
Další blogy.....	452
Novinky v PowerShellu 2.....	453
Vzdálené připojení čili remoting.....	453
Vylepšené funkce a vlastní cmdlety.....	453
Moduly a jejich využití.....	453
Nezávislé úlohy – joby.....	454
Skripty řízené událostmi.....	454
Grafické rozhraní ISE.....	454
Ladící rozhraní a cmdlety.....	454
Podpora transakčního zpracování.....	455
Zajímavé konstrukce jazyka PowerShell.....	455
Nové cmdlety.....	455

Rejstřík.....457

Úvod

Je tomu už téměř pět let, kdy jsem si poprvé opatřil veřejnou testovací verzi aplikace, které se říkalo Monad, a začal zkoumat, co že to vlastně Microsoft pouští mezi dělný počítačový lid. Dostupnost dokumentace na Internetu a tehdejší informační zázemí dávalo tušit, že se jedná spíše o undergroundovou zábavu než o stabilní produkt v produkci softwarového gigantu. A byl tu ještě jeden spolehlivý příznak toho, že Monad je něco krajně „podezřelého“ – před započítím prvních pokusů mi už bylo z doslechu známo, že Monad má být pouze a jenom příkazové rozhraní, tedy shell, ale zdráhal jsem se tomu uvěřit, dokud si to sám neprověřím. A po chvíli, když jsem Monad nainstaloval, bylo jasné jen to, že Monad je opravdu funkční shell.

Vše ostatní ovšem bylo naprosto záhadné. Šířily se odvážné zkazky o tom, že Monad nahradí starý dobrý příkazový řádek – inu, už bylo načase, však on ten starý dobrý Cmd.exe nikdy nebyl žádná velká hvězda, pokud jsme si připustili srovnání s jeho ekvivalenty v Unixu. Šířily se však ještě bláznivější drby, a těm mohl tehdy uvěřit opravdu jen málokdo: Monad by jednou měl nabýt vrchu nad grafickým uživatelským rozhraním a stát se hlavním nástrojem pro správu! Kdo by na jaře roku 2005 takové báchorce uvěřil! Na druhou stranu, Monad přišel s alespoň nějakou dokumentací, nápověda fungovala a jeho základní kvality bylo možné rozpoznat a prověřit. Upřímně řečeno, byl to pramen živé vody pro někoho, kdo nikdy neztratil pocit, že ovládání operačního systému primárně pomocí klikání je často nejen nepraktické, ale v zásadě též kacířské a nactiutrháčké. Nakonec přišla na podzim roku 2005 i příležitost podělit se o své nadšení s ostatními aktivisty „microsoftího undergroundu“ – počítačová škola Gopas mi nabídla v dramaturgii své konference TechEd jednu přednášku, v níž jsem mohl dobré zprávy o Monadu zvěstovat i ostatním. Po jejím skončení jsem během započaté diskuze měl pocit, že někteří posluchači čekali na podobnou událost jako na návrat spasitele. Bylo opravdu znát, že ani léta grafického drilu ve Windows nevykořenila unixovou tradici a sílu myšlenky příkazové konzoly.

Podobné pocity zřejmě měli i samotní tvůrci Monadu, neboť věci poté nabraly rychlý spád. O rok později již byla blízko první finální verze produktu, jehož jméno se změnilo přes Microsoft Shell na výsledné PowerShell, a bylo jasné, že tvůrčí tým není parta zapomenutých zoufalců v nitru redmondského bludiště. Chystal se nový operační systém Windows Vista a PowerShell se málem stal jeho výchozí součástí, nakonec byl uvolněn jako instalační balíček pro několik verzí Windows a doprovázela jej dobrá dokumentace a podpora na Internetu. V té době jsem již natolik věřil v budoucnost této technologie, že jsem nabídl své počítačové škole sestavení kurzu na téma PowerShell s ambiciózním rozsahem pěti dnů. A spolu s tímto úmyslem jsem přistoupil na závazek, že ke kurzu připravím i další nezbytnou součást, o kterou byla tehdy velká nouze: učebnici. Tak se zrodila první původní česká kniha o PowerShellu, která nakonec v rozšířené podobě vyšla v nakladatelství, jež stojí i za tímto svazkem. A je to ona kniha, kterou najdete jako přílohu na doprovodném disku k této učebnici, která je jejím nástupcem.

Od uvedení první knihy uběhly více než dva roky a PowerShell neuvěřitelně dozrál. Jeho vývoj je stejně zajímavý, jako je bouřlivý, a především uvedení finální verze PowerShellu 2 znamená zásadní krok vpřed. Ze zajímavé rarity se stal de facto standard pro správu operačních systémů a aplikací společnosti Microsoft a v současné době je znalost PowerShellu v podstatě nezbytná, pokud chceme i nadále provádět jejich administraci, především pak u nových a přicházejících verzí. Na počátku předchozí knihy jsem spekuloval, že správci, kteří včas nevezmou PowerShell na vědomí, se mohou časem ocitnout mimo „první ligu“. Čas dozrál pro tuto knihu, jež se zabývá PowerShellem po dalších třech letech vývoje, a já již nemusím spekulovat: ten, kdo bude PowerShell ignorovat, se nejspíše s příští verzí „svého“ systému či aplikace ocitne jako správce mimo hru. I to byl důvod, proč jsem se pustil do tvorby této publikace, která v mnohých tématech navazuje na dílko první, která však především přináší řadu zcela nových témat, jež se v PowerShellu objevila spolu s vývojem verze 2.

Publikace, kterou čtenář dostává do ruky, tedy nespadá z čistého nebe. Jak jsem již naznačil, navazuje na první knihu, kterou jsem PowerShellu věnoval a která vysvětlovala základy jeho použití i úvodní problematiku skriptovacího jazyka. Následující kapitoly pak navazují na látku, která již byla jednou vysvětlena – PowerShell 2 sice přidává řadu vylepšení, avšak základy práce jsou stejné a stejně tak výchozí skriptovací postupy se nezměnily. Čtenář nemusí litovat, pokud předchozí svazek nevlastní, neboť jeho kompletní verze je umístěna na doprovodném disku této publikace a má tak sloužit jako referenční materiál i úvodní kurz pro toho, komu se látka v této knize zdá dosti pokročilá.

Tento svazek navazuje na knihu předchozí také svou strukturou. První a druhá kapitola jsou jednak úvodem, shrnujícím momentální postavení PowerShellu, jednak základním roztríděním existujících variant a popisem jejich zprovoznění. Třetí kapitolu jsem věnoval popisu spouštění příkazů a skriptů v PowerShellu za jakýchkoliv podmínek – interaktivně či automaticky, lokálně či na vzdáleném počítači. Čtvrtá část je zaměřena na jazyk PowerShell jako takový a jeho zajímavé možnosti, zatímco pátá část je přesným opakem, neboť se zaměřuje na konkrétní oblasti správy systému Windows. Kapitola šestá je úplně zasvěcena adresářové službě Active Directory, kde PowerShell zaznamenal velký rozmach, a část sedmá je zaměřena na správu sítí v užším slova smyslu. V osmé kapitole jsem podal stručný přehled současných aplikovaných variant PowerShellu a rozšíření, část devátá (příloha) je pak referenčním přehledem. Jsem si jako autor plně vědom, že ani druhý svazek na dané téma nemůže zdaleka vyčerpávat nabízené možnosti, a látka celé knihy tak nutně musí být vzorkem nabízených možností. Pevně doufám, že tento vzorek je alespoň dostatečně reprezentativní a zajímavý.

Na tomto místě knihy se sluší poděkovat. Můj vděk patří radě účastníků mých kurzů, kteří neváhali a chtěli se s novou technologií co nejdříve seznámit, čímž podněcovali i mé úsilí o co nejlepší poznání možností PowerShellu a mou snahu je co nejlepším způsobem vysvětlit ostatním. Můj vděk určitě patří též sdružení WUG, které mi umožnilo opakovaně o PowerShellu přednášet a dopřát tak předchozí publikaci větší popularitu. A mám-li na závěr někoho výslovně zmínit, pak děkuji Martinu Trnkovi, který trpělivě přečetl rukopis a poskytl cenné připomínky, a především rodině, která to trpělivě vydržela.

KAPITOLA 1

PowerShell
a jeho svět

PowerShell se pomalu stal neodmyslitelnou součástí operačních systémů Windows a dalšího aplikačního softwaru společnosti Microsoft. Nebylo by na tom nic divného, kdyby se nejednalo o technologii, která byla ještě před pár lety pro Windows poměrně netypická: textové příkazové rozhraní, správčovská textová konzola, zkratka shell. Tradiční správci Windows sledovali nástup PowerShellu před několika lety tu se shovívavým úsměvem, tu s mírnou zvědavostí, tu s naprostým nepochopením. Úloha a postavení grafického rozhraní při správě se zdály tak neotřesitelné, že jen málokdo spatřoval v PowerShellu opravdovou budoucnost – proč by Microsoft opouštěl zavedenou a osvědčenou koncepci a navíc riskoval, že jeho rozmazlení správci se budou muset učit příkazový shell?

V průběhu několika posledních let však došlo k řadě překvapení. PowerShell jako technologie ukazoval působivé kvality a získával si řady příznivců. Autorský tým připravil před několika lety jeho první oficiální verzi a přes některé chybějící možnosti jsme tak získali hodnotný nástroj pro správu a automatizaci. Řada příznivců i softwarových firem rozpoznali potenciál nové technologie a využili její otevřenost k tomu, aby připravili různá vylepšení a rozšíření, která naznačovala cestu vpřed a dále za obzor původních představ. A společnost Microsoft se rozhodla k razantnímu kroku, který řada skalních uživatelů produktů této firmy považovala za „nůž do zad“: produkt MS Exchange se stal svého druhu průkopníkem a jeho správci pokusnými králíky, když ve verzi 2007 se stal rozšířený PowerShell primárním administrátorským rozhraním a grafická konzola zdaleka nepokrývala všechnu potřebnou funkcionalitu. Toto byl důležitý signál, že PowerShell není hříčka jakéhosi bohémského tvůrčího týmu, ale že jde o zásadní koncepci, jejíž nástup máme jednou okusit všichni.

PowerShell se však zdaleka nestal jen novým konzolovým rozhraním v oné podobě, jakou připravil vývojový tým společnosti Microsoft. PowerShell je již dnes ve skutečnosti opravdovou základnou – platformou pro vývoj dalších součástí, které nadále rozšiřují možnosti správy operačního systému a aplikací. Vývoj přitom postupuje hned v několika proudech a využívá PowerShell různými způsoby. Třeba tvůrci produktu PowerGUI naplnili původní myšlenku autorů PowerShellu a „dotáhli do konce“ originální návrh: PowerShell bude prostředníkem pro ovládání operačních systémů a aplikací a grafická rozhraní budou jeho nadstavbou. Právě toto provádí PowerGUI, stejně jako to dnes dělá třeba konzola pro správu produktu MS Exchange nebo nové grafické rozhraní pro správu Active Directory v serverech Windows 2008. Řada vývojářů sleduje jiný směr a využívá PowerShell jako ovládací nástroj při přístupu k dalším a dalším datovým zdrojům a rozhraním. Dobré podpory se dočkal MS SQL Server a databáze vůbec, zcela samostatnou kategorií je pak třeba služba Active Directory. V těchto případech tvůrci těží především z klíčových vymožeností PowerShellu, kterými jsou objektové zpracování dat a především objektová roura. Do třetice můžeme zmínit nastoupený trend, který prosazují sami tvůrci operačního systému Windows. Systém je již dodáván s PowerShellem a sadou rozšiřujících modulů, které poskytují rozhraní pro správu dalších a dalších součástí Windows.

Poslední tři léta byla pro příznivce PowerShellu obzvláště plodná a zajímavá, neboť tvůrčí tým pracoval na verzi 2, která byla nakonec uvolněna ve druhé polovině roku 2009 postupně pro všechny operační systémy Windows počínaje verzí XP. Druhá oficiální verze představuje velmi výrazný posun vpřed a přináší vylepšení v mnoha směrech –

nacházíme zde zásadní koncepční rozšíření (vzdálený přístup, úlohy na pozadí, zpracování událostí, moduly atd.), rozšiřuje se rodina cmdletů a také některé struktury jazyka nabízejí nové, významné možnosti (kupříkladu funkce jako vlastní cmdlety). Verze 2 je již automaticky distribuována jako součást nejnovějších verzí Windows (7 a 2008 R2) a konečně též pracuje – jak bychom koneckonců očekávali – i ve variantě Core serverového operačního systému. Stal se tak opravdu univerzálním rozhraním pro správu, na němž se bude do budoucna stavět. Svou roli PowerShell ve verzi 2 potvrdil i tam, kde už způsobil jeden šok: produkt MS Exchange 2010 je opět průkopníkem, neboť jeho administrátorské konzoly jsou postaveny právě na nové verzi a grafická nadstavba je pak závislá právě na implementaci PowerShellu.

Na tomto místě je potřeba zmínit, že správce prozatím nemůže pomocí PowerShellu ovládat úplně vše, co by v operačním systému či aplikacích chtěl. Některé části Windows a řada nastavení prozatím nemají vhodné rozhraní, které by umožnilo přístup pomocí prostředků PowerShellu, byť zprostředkovaně. Týká se to kupříkladu síťových funkcí a řady služeb, jež se sítěmi souvisejí, ale i řady jiných systémových služeb, pro jejichž ovládání potřebujeme speciální rozhraní, často reprezentované specifickým programem pro textovou konzolu. Řada úloh je také pomocí PowerShellu hůře zvladatelná než prostřednictvím jiného, dosud tradičně používaného nástroje (třeba některá nastavení oprávnění na souborový systém či objekty Active Directory). Tato situace však bude pouze dočasná, neboť PowerShell se dravě vyvíjí a funkce, které nezávládně připravit samotný autorský tým, se třeba objevují v příspěvcích autorů z komunity volného softwaru. Tato zmínka nás přivádí k další důležité skutečnosti: PowerShell dokázal vzbudit velký zájem tvůrců a v současné době najdeme řadu projektů, které doplňují a rozšiřují jeho možnosti.

Ani v případě, že jsme k prosazování PowerShellu stále mírně skeptičtí, bychom neměli přehlížet právě skutečnost, že Microsoft jej využívá stále více jako nosnou technologii pro správu svých produktů. Vedle Exchange, o kterém již padla zmínka, se PowerShell objevuje v dalším klíčovém produktu SharePoint, kde plní roli nosného správcovského rozhraní, a třeba služba Active Directory již nabízí ve verzi Server 2008 R2 některé funkce, které jinak než v PowerShellu nemůžeme ovládat. Tento trend se zdá zřejmý a nezvratný.

PowerShell ve verzi 2 se tedy zdá být onou „pravou“ chvílí, kdy bychom se nejpozději měli začít s novou koncepcí správy seznamovat. Jistě, i do budoucna zde budou grafické konzoly, a jejich pravověrní příznivci se na ně budou moci spolehnout. Ale třeba již zmíněný Exchange jasně ukázal, jaké pravidlo bude do budoucna platit: dobře ovládnout znamená dobře ovládnout pomocí PowerShellu. Neznalost této technologie bude znamenat hendikep, jehož velikost se bude zvyšovat úměrně s pronikáním PowerShellu do všech klíčových produktů Microsoftu. A je docela možné, že ona zlomová chvíle již nastala: možná že již dnes neznalost PowerShellu znamená dobrovolný vstup do „druhé ligy“ správců platformy Windows a dalších aplikací jejich výrobce.

V následujících kapitolách čtenář nalezne řadu úloh z různých zákoutí používání PowerShellu a také z všemožných oblastí správy Windows. V mnoha případech si ukážeme, jak dravým způsobem se PowerShell prosazuje – tu a tam se zatím dotahuje na pozice jiných nástrojů pro správu, místy ale již výrazně vede a představuje nejlepší cestu k pro-

vedení té či oné práce. V každém případě ale představíme čtenáři konzistentní, funkční a velmi užitečné prostředí pro moderní administraci, na jehož základních kamenech lze spolehlivě vybudovat rozsáhlý aparát pro pokročilou správu sítí se systémy Windows prakticky jakékoliv velikosti.

KAPITOLA 2

PowerShell jako software

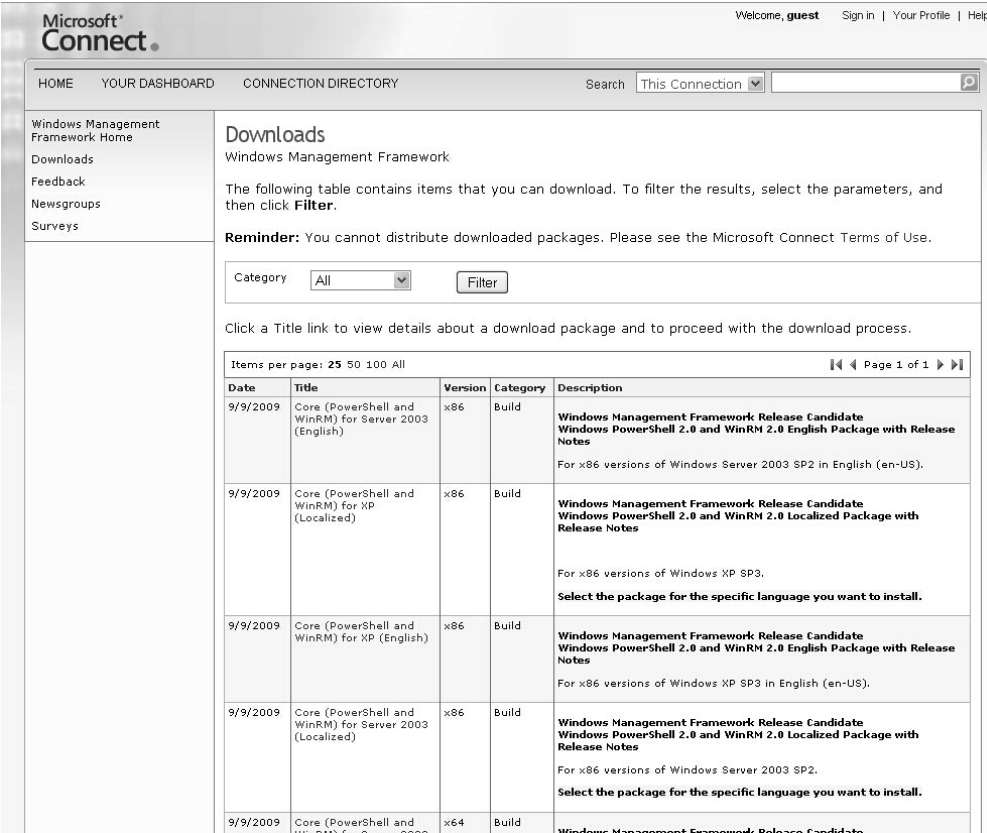
V této kapitole:

- ◆ Verze a varianty PowerShellu
- ◆ Instalace PowerShellu
- ◆ Rozšíření a doplňky PowerShellu

Existující implementace a verze PowerShellu

PowerShell se dosud dočkal dvou oficiálních samostatných verzí, které byly uvedeny do světa a označeny jako „finální“. Jeho vývoj však byl a je bouřlivý a usilovný, což se projevuje vnášením zmatku do množství dostupných verzí a variant, často dodávaných s „mateřským“ produktem jako jeho správcovské rozhraní. V této části se tedy budeme blíže věnovat verzím PowerShellu a různým odrůdám, jež se postupně rodí s uváděním nových verzí různých produktů Microsoftu.

Na počátku zkusme vytyčit základní orientační body – distribuce PowerShellu si hrubě rozdělíme na dvě velké skupiny. V jedné z nich budou zařazeny základní distribuce (PowerShell v úzkém slova smyslu), tedy samotné „jádro“ PowerShellu (core). Druhá skupina naší klasifikace pak zahrnuje odvozené varianty (odrůdy), které přidávají k jádru další možnosti či rozšíření. Tato bývají typicky poplatná samotnému produktu, se kterým je takovýto PowerShell dodáván (zmiňme první z nich, Exchange 2007 a jeho EMS – Exchange Management Shell).



Microsoft Connect

Welcome, **guest** | Sign in | Your Profile | Help

HOME YOUR DASHBOARD CONNECTION DIRECTORY Search This Connection

Windows Management Framework Home
Downloads
Feedback
Newsgroups
Surveys

Downloads

Windows Management Framework

The following table contains items that you can download. To filter the results, select the parameters, and then click **Filter**.

Reminder: You cannot distribute downloaded packages. Please see the Microsoft Connect Terms of Use.

Category:

Click a Title link to view details about a download package and to proceed with the download process.

Items per page: 25 50 100 All Page 1 of 1

Date	Title	Version	Category	Description
9/9/2009	Core (PowerShell and WinRM) for Server 2003 (English)	x86	Build	Windows Management Framework Release Candidate Windows PowerShell 2.0 and WinRM 2.0 English Package with Release Notes For x86 versions of Windows Server 2003 SP2 in English (en-US).
9/9/2009	Core (PowerShell and WinRM) for XP (Localized)	x86	Build	Windows Management Framework Release Candidate Windows PowerShell 2.0 and WinRM 2.0 Localized Package with Release Notes For x86 versions of Windows XP SP3. Select the package for the specific language you want to install.
9/9/2009	Core (PowerShell and WinRM) for XP (English)	x86	Build	Windows Management Framework Release Candidate Windows PowerShell 2.0 and WinRM 2.0 English Package with Release Notes For x86 versions of Windows XP SP3 in English (en-US).
9/9/2009	Core (PowerShell and WinRM) for Server 2003 (Localized)	x86	Build	Windows Management Framework Release Candidate Windows PowerShell 2.0 and WinRM 2.0 Localized Package with Release Notes For x86 versions of Windows Server 2003 SP2. Select the package for the specific language you want to install.
9/9/2009	Core (PowerShell and WinRM) for Server 2003 (English)	x64	Build	Windows Management Framework Release Candidate Windows PowerShell 2.0 and WinRM 2.0 English Package with Release Notes

Obrázek 2.1: Stránky Microsoft Connect jsou výchozím místem jak pro pořízení nejnovější verze PowerShellu, tak pro zaslání připomínek a námětů na další vylepšení

První skupina – jádra PowerShellu – dnes zahrnuje dvě finální verze, označované jako 1 a 2. Obě jsou plně funkční a použitelné, verze 2 je pak vybavena velkým množstvím nových možností oproti o tři roky starší verzi 1. V současné době platí, že obě tyto verze jsou dostupné jako samostatný instalační balík pro Windows od verze XP až po současnost (dělicí čarou do minulosti jsou Windows 2000, pro něž není PowerShell distribuován). Samostatný instalační balík není potřeba shánět u verzí Windows, které obsahují PowerShell jako svoji výchozí součást. Tabulka nám ukazuje tyto systémy a verzi jádra PowerShellu.

Tabulka 2.1: Verze jádra PowerShellu a operační systémy, v nichž je předinstalován

Verze Windows	Verze jádra PowerShellu
Windows 2008 Server (ne varianty „Core“)	PowerShell 1.0
Windows 7	PowerShell 2.0
Windows 2008 Server R2	PowerShell 2.0

Ostatní systémy, jež nejsou v tabulce zmíněny, neobsahují PowerShell ve své základní instalaci a tento musí být přidán dodatečně (Windows XP, 2003, 2003 R2, Vista). Dodejme, že v současné době je jádro PowerShellu (tedy ona základní distribuce PowerShellu 2 a těsně souvisejících knihoven) označováno jako Windows Management Framework.



Tip: Čtenář může najít oficiální informace o uvedení tohoto balíku v následujícím článku, který je součástí oficiální podpory společnosti Microsoft:

<http://support.microsoft.com/kb/968929>

Na této stránce jsou i odkazy na stahování instalačních balíků.

Druhá skupina, kterou jsme si výše vymezili, pak zahrnuje rozšířené varianty PowerShellu, jež jsou doplněny o různé doplňky, moduly apod. Základem je vždy jádro (core), k němuž jsou přidána rozšíření buďto ve formě snap-inu (především ve verzi 1), nebo modulu (ve verzi 2). Takovéto varianty jsou dodávány s aplikacemi Microsoftu, a jsou tedy poskytovány jako rozhraní pro aplikační správu. Těmto rozšířeným variantám se budeme blíže věnovat v jedné z dalších kapitol.

Tabulka 2.2: Významné aplikované varianty PowerShellu v produktech firmy Microsoft

Produkt	Rozšíření PowerShellu	Funkcionalita
MS Exchange 2007	Exchange Management Shell	Správa služby Exchange
MS Exchange 2010	Exchange Management Shell	Správa služby Exchange
Rodina MS System Center	Management Shell	Správa aplikací
MS SQL Server 2008	SQL Server PowerShell	Správa databázového serveru a databází

Verze CTP a související potíže

Jak jsme již zmínili, vývoj jádra PowerShellu byl a je bouřlivý, což se mimo jiné projevilo postupným uvolňováním testovacích verzí. Mezi verzemi PowerShell 1 a PowerShell 2 byly postupně uvolněny tři testovací verze, jež byly veřejně dostupné a pro jejichž

označení se vžily zkratky CTP (community technology preview – tedy něco ve smyslu ukázkové verze pro nedočkavé příznivce). Tyto distribuce měly a mají pro vývoj PowerShellu obrovský význam, neboť vývojový tým bedlivě naslouchá připomínkám uživatelů a průběžně je zpracovává do dalších verzí.



Tip: Zájmcům o vylepšení PowerShellu doporučujeme navštívit následující stránky, kde lze přispět do diskuze a oznámit případné chyby:

<https://connect.microsoft.com/windowsmanagement>

Testovací verze CTP však s sebou nesou jisté riziko. Vyznačují se totiž odlišnostmi (někdy dosti výraznými) od závěrečné, oficiální uvolněné verze PowerShellu 2. To, co funguje v některé z verzí CTP (PowerShell 2 měl postupně takové verze tři), nemusí fungovat ve finální podobě, případně to může fungovat jinak, pod jiným jménem apod. Jinak řečeno, během vývoje testovacích verzí se mění jména cmdletů, jména jejich přepínačů, někdy cmdlety i jejich přepínače zcela mizí a objevují se jiné. PowerShell se tedy může vyvíjet dosti dramaticky a my to musíme mít na zřeteli.

Windows PowerShell V2 Community Technology Preview 3 (CTP3)

Brief Description
Windows PowerShell 2.0 (RTM) has been released as part of Windows 7 and Windows 2008 R2. To download Windows PowerShell 2.0 for XP/W2K3/Vista/W2K8, please refer to KB 968929.

On This Page

Quick Details	Overview
System Requirements	Instructions
Related Resources	What Others Are Downloading

[Download files below](#)

Quick Details

Version:	2.0
Date Published:	12/22/2008
Language:	English
Download Size:	599 KB - 27.4 MB*

*Download size depends on selected download components.

Overview

NOTE: Windows PowerShell 2.0 (RTM) has been released as part of Windows 7 and Windows 2008 R2. To download Windows PowerShell 2.0 for XP/W2K3/Vista/W2K8, please refer to KB 968929.

Use the Download link on this page to download the installation package for the Windows PowerShell V2 CTP3. Windows PowerShell V2 CTP3 introduces several significant features to Windows PowerShell 1.0 and Windows PowerShell V2 CTPs that extends its use, improves its usability, and allows you to control and manage the Windows environment more easily and comprehensively.

- **This software is a pre-release version and should not be deployed in a production environment. It will not work the way a final version of the software does. Features will change before final release.**

Obrázek 2.2: Vývojové verze PowerShellu jsou stále dostupné ke stahování. K jejich použití již ale není žádný důvod – finální verze jsou k dispozici.

S vývojem verzí CTP souvisí i druhotný problém. Vývoj PowerShellu 2, který trval několik let, poskytl zájemcům verze CTP, na jejichž základě vznikla spousta dokumentace, ale také článků na Internetu s řadou příkladů a řešení. Tato „powershellová“ literatura však trpí tím, že se řada věcí v PowerShellu změnila a třeba příklady nemusí fungovat v uvedené podobě. Pak musíme sáhnout k jejich úpravě dle aktuálního stavu. Tyto skutečnosti musí uživatel PowerShellu stále respektovat a ke zdrojům na Internetu musí přistupovat s odpovídající rezervou.



Tip: Tobias Weltner, přispěvatel blogu PowerShell.com a autor pěkné, zdarma dostupné knihy o PowerShellu, napsal krásný článek o problematice kontroly verzí PowerShellu. Na následující stránce je krom výkladu také ukázka skriptů, které nám mohou pomoci.

<http://powershell.com/cs/blogs/tobias/archive/2010/01/24/are-you-using-the-correct-powershell-version.aspx>



Tip: Jiný autor zajímavých stránek, Oisín Grehan, se také věnuje důslednému sledování vývoje verzí PowerShellu a jejich proměn. Řadu informací naleznete na jeho blogu, především v těchto článcích:

<http://www.nivot.org/2009/05/22/PowerShellV20DifferencesBetweenCTP3Win7BetaAndWin7RC.aspx>

<http://www.nivot.org/2008/12/23/PowerShell20CTP3HasArrived.aspx>

<http://blogs.msdn.com/powershell/archive/2008/12/24/windows-powershell-ctp2-to-ctp3-conversion-guide.aspx>

Správa služby Active Directory

Adresářová služba Active Directory patří mezi nejdůležitější služby v sítích s Windows, a proto asi vypadá zvláště, že její správě se tvůrci PowerShellu začali věnovat až v době vývoje verze 2. V první verzi vlastně neexistuje žádná speciální podpora pro práci s Active Directory (myšleny cmdlety, provider pro PSDrive apod.) a veškerá správa se musí provádět nepřímou, pomocí existujících rozhraní (starší ADSI, nověji pomocí tříd .NET Frameworku). Teprve uvedení verze 2 a nových verzí operačního systému Windows s sebou konečně přineslo také odvozenou implementaci PowerShellu s modulem pro Active Directory.

The screenshot shows a web page with the following content:

ACTIVE DIRECTORY POWERSHELL BLOG
PIPELINING AD - ONE OBJECT AT A TIME

HOME ABOUT EMAIL

ADDING/REMOVING MEMBERS FROM ANOTHER FOREST OR DOMAIN TO GROUPS IN ACTIVE DIRECTORY

Adding/removing members belonging to the same domain from a group is very simple using AD Powershell cmdlets. All you have to do is pass an identifier (either samAccountName, distinguishedName, securityIdentifier or GUID) of the member and group to one of the membership cmdlets:

- Add-ADGroupMember
- Remove-ADGroupMember
- Add-ADPrincipalGroupMembership
- Remove-ADPrincipalGroupMembership

Example:

```
C:\PS> Add-ADGroupMember -SvcAccPSOGroup -Member SQL01, SQL02 ## Adds the user accounts with SamAccountNames SQL01,SQL02 to the group SvcAccPSOGroup
C:\PS> Remove-ADPrincipalGroupMembership -Identity "Wilson Pais" -MemberOf "Administrators" ## Remove the user 'Wilson Pais' from the adm
```

However, when it comes to adding and removing cross-forest or cross-domain members from a group, things become a little difficult. Here is an example of the error message that you would see while trying to do cross-forest/domain operations the regular way:

```
PS C:\> Add-ADGroupMember Administrators -Members "CN=Users, CN=Users, DC=forest1888, DC=com"
Add-ADGroupMember : Cannot find an object with identity 'CN=Users, CN=Users, DC=forest1888, DC=com' under 'DC=forest1888, DC=com'
Add-ADGroupMember : Cannot find an object with identity 'CN=Users, CN=Users, DC=forest1888, DC=com' under 'DC=forest1888, DC=com'
```

Obrázek 2.3: Rozšíření Active Directory PowerShell od Microsoftu je podporováno i prostřednictvím blogu jeho autorů

Active Directory PowerShell můžeme používat právě jen v omezeném počtu nových operačních systémů, takže jeho masové rozšíření a využití teprve nastane (tabulka shrnuje jeho dostupnost). Na serverových variantách je tento odvozený PowerShell instalován automaticky spolu s dalšími nástroji na správu Active Directory, do Windows 7 jej můžeme přidat jako součást balíku RSAT (Remote Server Administration Tools). Bohužel na starších verzích Windows tato odvozená implementace správně nepracuje, a proto řada uživatelů jistě zůstane u další logické volby.

Tabulka 2.3: Zbrusu nový MS Active Directory PowerShell a jeho možnosti použití

Verze Windows	Active Directory PowerShell
Windows 2008 R2	Ano
Windows 2008	Ne
Windows 2003	Ne
Windows 7	Ano
Windows Vista	Ne
Windows XP	Ne



Tip: Podrobné informace o práci s rozšířením Active Directory PowerShell nalezne čtenář na následujících stránkách:

<http://technet.microsoft.com/en-us/library/dd378937%28WS.10%29.aspx>

<http://blogs.msdn.com/adpowershell/default.aspx>

<http://blogs.msdn.com/photos/adpowershell/images/9453590/original.aspx>

Pro doplnění dodejme, že omezené možnosti instalace Active Directory PowerShellu nevyklučují v žádném případě správu starších radičů domény a domén vůbec. Domény provozované na verzi serveru 2003 je možno spravovat, pokud máme v síti zapojen alespoň jeden radič ve verzi 2008 R2.



Tip: Čtenář nalezne výchozí informace pro výše popsany scénář na této stránce:

<http://blogs.msdn.com/adpowershell/archive/2009/09/18/active-directory-management-gateway-service-released-to-web-manage-your-windows-2003-2008-dcs-using-ad-powershell.aspx>

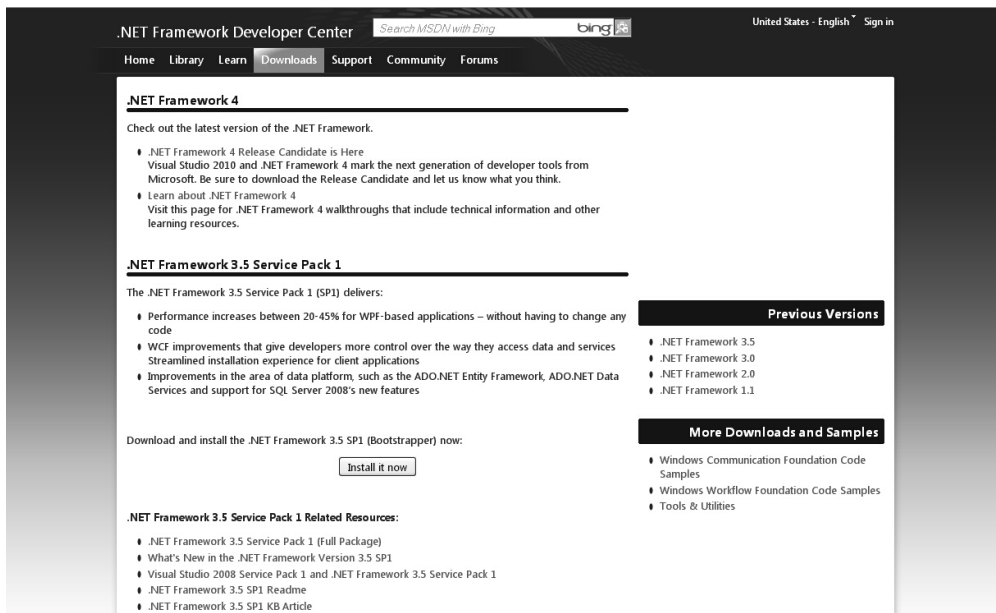
Instalace PowerShellu

Úvod

V některých verzích Windows není PowerShell nainstalován jako výchozí součást, takže musíme instalaci provést sami. Stejně tak budeme instalaci provádět u všech systémů, kde nemáme novější verzi 2.0. Celý proces je poměrně jednoduchý, a pokud splníme výchozí požadavky, tak i bezproblémový.

Instalace

PowerShell, či přesněji Windows Management Framework, můžeme nainstalovat na jakýkoliv systém Windows od verze Windows XP k novějším (s výjimkou nejnovějších verzí 7 a 2008, kde už nainstalován je).



KAPITOLA 2
PowerShell jako software

Obrázek 2.4: Před instalací se ujistíme, zda máme správnou verzi .NET Frameworku, případně ji stáhneme a nainstalujeme

1. Prověříme na cílovém počítači verzi platformy .NET Framework. V případě potřeby stáhneme a nainstalujeme nejnovější verzi. Instalované verze můžeme prověřit několika způsoby: jednak prohledáním adresáře s instalacemi, jednak prozkoumáním klíčů v databázi Registry.

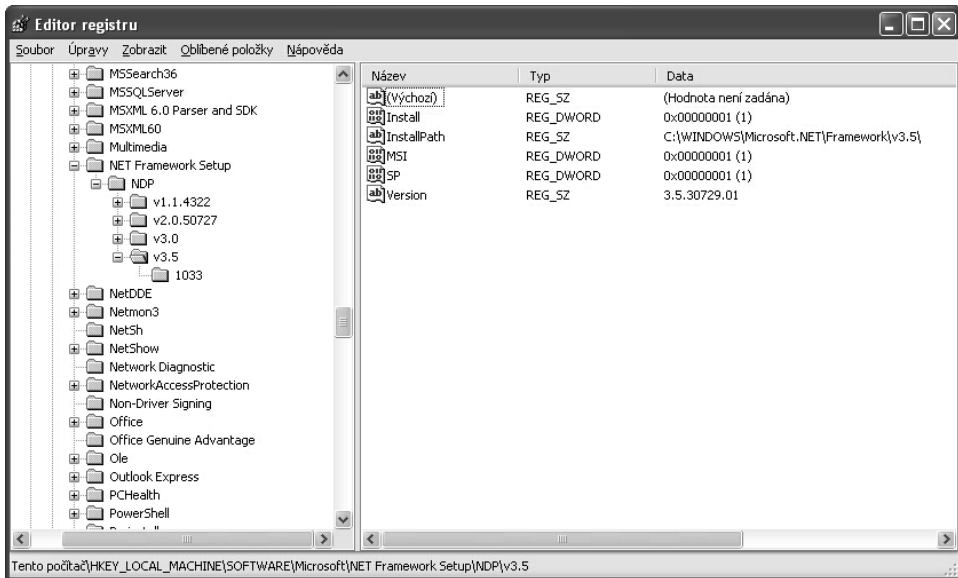
```
C:\>dir %windir%\Microsoft.NET\Framework\
Svazek v jednotce C je 120GB.
Sériové číslo svazku je E86A-5C34.
```

Výpis adresáře C:\WINDOWS\Microsoft.NET\Framework

```
09. 08. 2009  22:14 <DIR>      .
09. 08. 2009  22:14 <DIR>      ..
...
09. 08. 2009  22:11 <DIR>      v1.0.3705
17. 10. 2009  09:58 <DIR>      v1.1.4322
16. 09. 2006  19:37 <DIR>      v2.0.40607
01. 02. 2010  22:41 <DIR>      v2.0.50727
01. 02. 2009  15:02 <DIR>      v3.0
09. 08. 2009  22:14 <DIR>      v3.5
...
```


V Registry prozkoumáme následující klíč:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP



Obrázek 2.5: Příslušný klíč v databázi Registry nese informace o instalovaných verzích .NET Frameworku.



Tip: Aktuální verze .NET Frameworku je k dispozici na těchto stránkách:

<http://msdn.microsoft.com/en-us/netframework/default.aspx>

Podrobné informace o označení verzí pak najdeme zde:

<http://msdn.microsoft.com/en-us/kb/kb00318785.aspx>

Balík po stažení nainstalujeme pomocí spuštění jednoduchého průvodce.

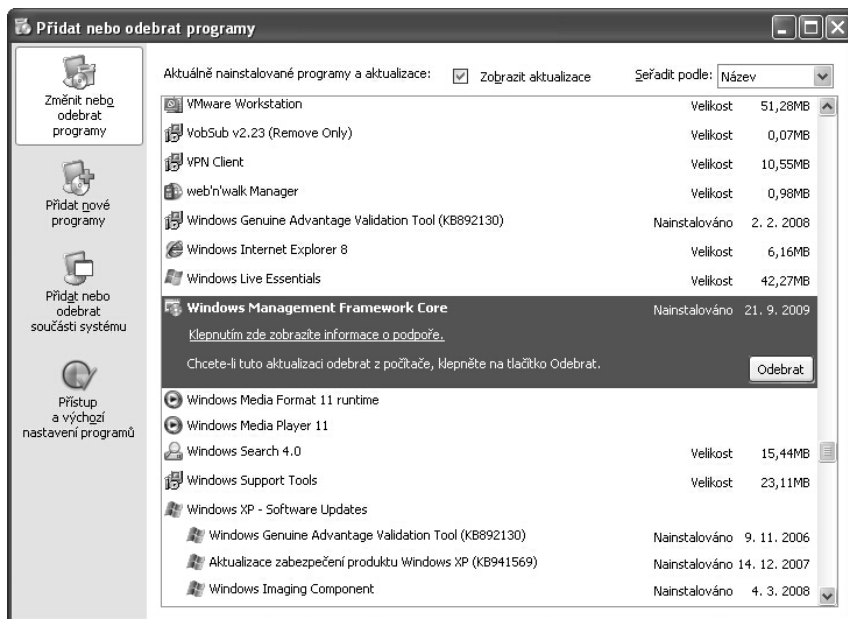
2. Pokud pracujeme se systémem Windows XP, prověříme, že je nainstalován balík Service Pack 3, případně jej stáhneme a nainstalujeme.



Tip: Distribuční varianta SP3 pro Windows XP je ke stažení zde:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=5B33B5A8-5E76-401F-BE08-1E1555D4F3D4&displaylang=en>

3. Přejdeme na stránku podpory PowerShellu a vyhledáme stažení potřebné verze shellu: <http://support.microsoft.com/kb/968929>
4. Spustíme instalační balíček a prostě dokončíme grafický průvodce.



Obrázek 2.6: Nezapomínejme, že PowerShell v nejnovější verzi je instalován jako Windows Management Framework a pod tímto jménem také figuruje u služby Windows Installer

Shrnutí

PowerShell je distribuován v dobře popsáných instalačních balících, jež obsahují nejdůležitější součásti, včetně potřebné verze služby Windows Remote Management pro vzdálenou správu. Základem stoprocentní funkcionality PowerShellu je ovšem .NET Framework, a proto bychom neměli váhat s instalací jeho poslední verze všude tam, kde je to možné.

Důležité k zapamatování

- ◆ Operační systémy Windows Server 2008 R2 a Windows 7 mají PowerShell 2 jako svou součást a nevyžadují žádnou další instalaci.
- ◆ Operační systém Windows Server 2008 obsahuje verzi PowerShell 1. Novější verzi je potřeba doinstalovat.
- ◆ Operační systém Windows Server 2008 Core nemá oficiálně podporován PowerShell. Jeho instalaci můžeme provést způsobem, která je popsán na následující internetové stránce. Tento postup není podporován společností Microsoft. <http://dmitrysoznikov.wordpress.com/2008/05/15/powershell-on-server-core/>
- ◆ Systém Windows XP vyžaduje instalaci SP3 pro PowerShell 2.

Rozšíření a doplňky

Úvod

PowerShell je prozatím určitě „mladý produkt“, myšleno ve srovnání s dobou vývoje jiného osvědčeného softwaru. Jeho nízký věk však výrazně kontrastuje s počtem uživatelů, vývojářů a softwarových firem, kteří byli vyprovokováni a chytili hozenou rukavici. Díky tomu dnes můžeme směle instalovat pro PowerShell řadu zajímavých doplňků a rozšíření.

Windows System Modules

Nástup PowerShellu ve verzi 2 znamenal také masivní přechod k distribuci různých rozšíření prostřednictvím tzv. modulů (zmníme se o nich podrobněji v následující kapitole). Nejnovější verze Windows (Server 2008 R2) tak již přichází s řadou zajímavých modulů, které jsou připraveny ve výchozím adresáři PowerShellu a můžeme je rovnou „natáhnout“ a začít používat. Zmíňme třeba Active Directory PowerShell, podporu pro přenos souborů BITS, ale také třeba modul pro ovládání služby IIS. Grafické rozhraní Serveru 2008 R2 též nabízí přímý odkaz na spuštění konzoly PowerShellu, která automaticky tyto moduly natahuje a dává k dispozici. Operaci můžeme provést také sami kdykoliv později následujícím způsobem:

```
=>powershell -importsystemmodules
```

Windows 7 PowerShell Pack

<http://code.msdn.microsoft.com/PowerShellPack>

<http://blogs.msdn.com/powershell/archive/2009/10/15/introducing-the-windows-7-resource-kit-powershell-pack.aspx>

Tento balík je vlastně sbírkou zajímavých skriptů, modulů a funkcí pro PowerShell, které se hodí pro správu a také pro některé další experimenty s PowerShellem. Najdeme zde skripty pro tvorbu grafického rozhraní, ale také rozhraní pro práci se službou Plánování úloh či inventuru operačního systému. Balík vznikl postupným sbíráním řešení, jež poskytl vývojový tým a především autoři stránek podpory PowerShellu (populární rubriky The Scripting Guy).

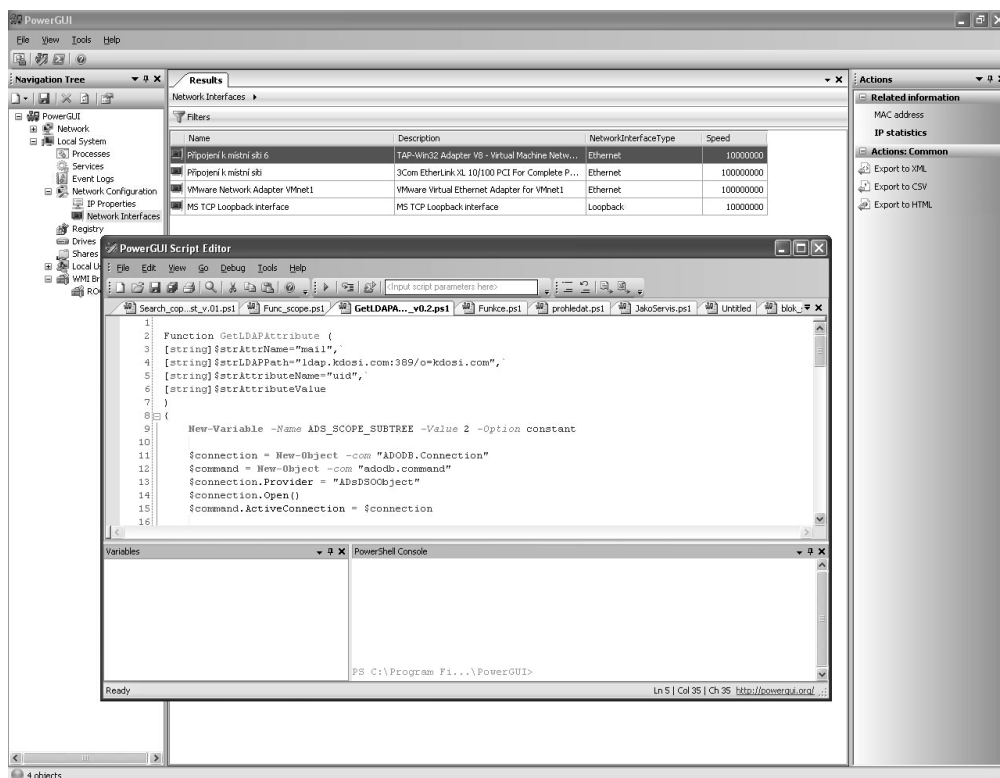
PowerGUI

www.powergui.org

PowerGUI bezesporu patří mezi nejlepší rozšíření, jež byla na bázi PowerShellu vyvinuta. Zahrnuje v sobě dva klíčové nástroje, jež poslouží stejně dobře správcům jako všem tvůrcům skriptů v jazyce PowerShell. Software je vyvíjen a distribuován jako volný – jednak na něm pracují dobrovolní přispěvatelé, jednak je podporován softwarovou firmou Quest Software.

Balík zahrnuje dvě klíčové části. Samotné PowerGUI je grafické rozhraní pro správu Windows a případných dalších aplikací – ano, stále mluvíme o PowerShellu a zároveň

o grafickém správcovském rozhraní. PowerGUI je zcela postaveno na PowerShellu a všechny operace, které „naklikáme“, jsou převedeny na kód v PowerShellu a v něm také provedeny. Druhá část balíku se jmenuje PowerGUI Script Editor a jde o integrované vývojové rozhraní (IDE), v němž můžeme psát skripty a ladit jejich činnost pomocí ladicího rozhraní.



Obrázek 2.7: PowerGUI patří rozhodně mezi to nejlepší, co kdy pro PowerShell vzniklo.

PowerShell Commands for Active Directory

<http://www.quest.com/powershell/activeroles-server.aspx>

Tento doplněk je jedním z nejlepších a vůbec nejpobulárnějších rozšíření nového správcovského rozhraní. Objevil se v dobách PowerShellu 1, který nenabízel žádnou vlastní přímočarou variantu pro správu Active Directory, a znamenal bez nadsázky revoluci v používání PowerShellu. Ta se projevila mimo jiné tím, že řada správců začala používat PowerShell právě kvůli Active Directory a tomuto rozšíření, a ne jeden uživatel dokonce ani nepostřehl, že tato sada cmdletů nepohází z původní dílny Microsoftu! Balík se stal standardem de facto pro práci s Active Directory a my mu budeme v příslušné kapitole věnovat odpovídající pozornost. Je k dispozici zdarma a spolu s ním lze stáhnout též špičkovou doprovodnou dokumentaci.

Windows 7 Troubleshooting Platform

<http://msdn.microsoft.com/en-us/library/dd776530.aspx>

<http://technet.microsoft.com/en-us/library/ee424304%28WS.10%29.aspx>

<http://www.withinwindows.com/2009/01/12/crash-course-on-authoring-windows-7-troubleshooting-packs/>

<http://blogs.msdn.com/powershell/archive/2009/06/14/kudos-to-the-win7-diagnostics-team.aspx>

Operační systém Windows 7 obsahuje zajímavou novinku pod výše uvedeným názvem. Jedná se o sbírku skriptů v PowerShellu a doplňující soubory XML s popisy, které slouží k řešení různých typů problémů s operačním systémem. Určitě nejde o nahodilý počín, neboť můžeme vyhledat dokumentaci a začít sami tvořit další skripty a balíčky, které budou toto „ladičí centrum“ doplňovat a rozšiřovat. Jde o velmi zajímavou a bezprostřední ukázkou využití nové verze PowerShellu.

Windows PowerShell 2.0 SDK

<http://go.microsoft.com/fwlink/?LinkID=180421>

Tento balík je určen pokročilým zájemcům o PowerShell a programátorům aplikací na platformě .NET, kteří chtějí s PowerShellem pracovat. Jde o podrobnou dokumentaci jádra PowerShellu a popis jeho architektury. Najdete zde i řadu příkladů, ukázkových knihoven a skriptů, které mimo jiné ukazují, jak můžeme PowerShell rozšiřovat pomocí vlastních cmdletů.

Ačkoliv tento balík vypadá jako dosti specializovaný doplněk, neměli by jej přehlížet zájemci o tvorbu cmdletů pomocí samotného PowerShellu naleznou v něm popis potřebných objektů a ukázky obecných možností cmdletů, které lze využívat i ve funkcích v jazyce PowerShell.

PowerShell Community Extensions

<http://www.codeplex.com/Pscx>

Velmi populární balík rozšíření je jedním z nejlepších příspěvků komunity dobrovolných tvůrců a přispěvatelů, která se vytvořila kolem PowerShellu. Je velmi populární již od svého prvního uvedení a přinesl řadu vylepšení, kdy především po uvolnění PowerShellu verze 1 měli uživatelé pocit, že autoři mnoho věcí nestihli. Balík mimo jiné obsahuje implementaci Active Directory do podoby PSDrive, ale také řadu zajímavých funkcí a cmdletů, třeba pro práci s texty.

Shrnutí

PowerShell je již sám o sobě velmi silným nástrojem, avšak některá dostupná rozšíření jej proměňují v ještě všestrannější platformu pro ovládání aplikací a služeb. Doplnky jsou k dispozici jak ve starším provedení (snap-iny), tak především v novějších variantách jako moduly. Některá rozšíření se již stala v podstatě nezbytnou součástí shellu pro správu pokročilého prostředí na platformě Windows.

Důležité k zapamatování

- ◆ PowerGUI je jedním z nejlepších doplňků, který si můžeme nainstalovat – nabízí jak zajímavou grafickou nadstavbu, tak kvalitní editor pro tvorbu a ladění skriptů v PowerShellu.
- ◆ Quest PowerShell Commands for Active Directory je výborný balík pro všechny verze PowerShellu na správu Active Directory. Jedná se o prakticky nezbytné vybavení každého správce této služby.

Otázky a odpovědi

Otázka:

Můžeme nainstalovat PowerShell na všechny dostupné verze operačního systému Windows?

Odpověď:

Ne, nemůžeme. PowerShell závisí na dalších technologiích (především běhovém prostředí .NET Framework) a pracuje na Windows XP a novějších. PowerShell nelze použít na verze 2000 a starší.

Otázka:

Co jsou verze PowerShell 2 CTP (1, 2 nebo 3)? Mohu je bezpečně používat?

Odpověď:

Takto označené verze byly určeny k testování a není doporučeno jich využívat v produkčním prostředí. Ačkoliv jsou normálně funkční, liší se od konečné verze a některé příkazy nemusí pracovat stoprocentně.

Otázka:

Jakou verzi .NET Frameworku PowerShell vyžaduje?

Odpověď:

PowerShell běží i na starších verzích frameworku od verze 2. Avšak aby byl PowerShell plně funkční (všechny cmdlety), je potřeba nainstalovat framework ve verzi 3.5 SP1.

KAPITOLA 3

Práce v prostředí PowerShellu

V této kapitole:

- ◆ Skriptovací bloky a ovládání PowerShellu
- ◆ Vzdálené připojení v PowerShellu
- ◆ PowerShell jako webová služba
- ◆ Úlohy na pozadí a hromadné spouštění
- ◆ Automatizované úlohy
- ◆ Moduly v PowerShellu

Interaktivní práce

Úvod

Uživatelé PowerShellu, kteří nejsou úplnými začátečníky, jsou zřejmě zvyklí na dvě hlavní cesty používání PowerShellu: interaktivní provádění akcí v konzole a spouštění hotových skriptů v přímém či automatizovaném režimu. Při práci s konzolou jsme uvykli tomu, že konzolové prostředí je vlastně hřištěm a my jsme upoutání uvnitř – pokud chceme uchovat data, ukládáme je do souborů, chceme-li uchovat nastavení, rovněž provádíme export. .NET Framework nám však umožňuje mnohem více, neboť spouštění kódu v PowerShellu lze řídit takřka totálně: jakýkoliv kus kódu lze proměnit v objektovou proměnnou stejně tak jako vše, co nastane při jeho spuštění. PowerShell ve verzi 2 přináší nové možnosti, jak spouštět kód a přitom důsledně kontrolovat vše jakoby „zvenčí“. Nezapomínejme tedy na to, že vše v PowerShellu je objekt.

Skriptovací blok a tělo skriptu

Jakýkoliv kus kódu či příkazová sekvence v PowerShellu jsou při svém běhu důsledně řízeny a kontrolovány, stejně jako jakýkoliv jiný kód na platformě .NET, na níž PowerShell běží. Skriptovací blok (blok skriptu, prováděcí blok) je základní jednotkou, do níž patří určitá sekvence příkazů a dalších struktur jazyka PowerShell. Takovýto blok a jeho ohraničení má zcela zásadní význam, neboť při jeho spuštění (*Invocation*) vzniká jakési „hřiště“, které mu .NET Framework vymezí. Hranice tohoto hřiště jsou velmi důležité – představují bariéru pro řadu důležitých struktur, které v našich skriptech používáme. Skriptovací blok tedy vymezuje výchozí ohraničení pro platnost proměnných, které vzniknou uvnitř, a také třeba určuje rozsah platnosti chyb, které nastanou a mohou být zachyceny.

Hranice skriptovacího bloku vznikají buďto automaticky (spustíme konzolu, spustíme skript), nebo jsou určeny výslovně (nadefinujeme funkci či skriptovací blok jako takový). Při výslovném určování hranic používáme složené závorky, které se v praxi objevují v mnoha souvislostech.

```
=>{Write-Output "Zevnitř"}
Write-Output "Zevnitř"
```

```
=>Function Blok {"Zevnitř funkce"}
```

```
ForEach-Object {"Zevnitř roury"}
```

Všechny tyto konstrukce jsou pouze obměnou skriptovacího bloku, zasazeného do různých souvislostí. K jeho provedení (spuštění, invokaci) může dojít různými cestami.

```
=>& {"Z nepojmenovaného bloku"}
Z nepojmenovaného bloku
=>cat .\blok_skript.ps1
Write-Output "Zevnitř skriptu"
=>.\blok_skript.ps1
Zevnitř skriptu
=>
=>Function Blok {"Zevnitř funkce"}
```

```

Administrator: C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

=>$VS = new-object -com VirtualServer.Application
New-Object : Cannot load COM type VirtualServer.Application.
At line:1 char:17
+ $VS = new-object <<<< -com VirtualServer.Application
+ CategoryInfo          : InvalidType: (:) [New-Object], PSArgumentException
+ FullyQualifiedErrorId : CannotLoadComObjectType,Microsoft.PowerShell.Commands.NewObjectComma
nd

=>$VS = new-object -com "VirtualServer.Application"
New-Object : Cannot load COM type VirtualServer.Application.
At line:1 char:17
+ $VS = new-object <<<< -com "VirtualServer.Application"
+ CategoryInfo          : InvalidType: (:) [New-Object], PSArgumentException
+ FullyQualifiedErrorId : CannotLoadComObjectType,Microsoft.PowerShell.Commands.NewObjectComma
nd

=>Get-Service v*

Status      Name                DisplayName
-----
Running     VMAuthdService     VMware Authorization Service
Stopped     VMnetDHCP           VMware DHCP Service
Stopped     VMware NAT Service  VMware NAT Service
Stopped     VSS                 Stínová kopie svazku

=>Get-EventLog -LogName system -EntryType error | measure

Count       : 565
Average     :
Sum         :
Maximum     :
Minimum     :
Property    :

=>(Get-EventLog -LogName system -EntryType error | measure).count
565
=>=

```

Obrázek 3.1: Spuštěná konzola PowerShellu představuje samostatný „prováděcí blok“ – automaticky zakládá hranice pro platnost proměnných, funkcí a dalších objektů

```

=>Blok
Zevnitř funkce
=>
=>1..3 | ForEach-Object {"Zevnitř roury"}
Zevnitř roury
Zevnitř roury
Zevnitř roury
=>if (1 -eq 1) {"Z podmínky"}
Z podmínky

```

Řada případů ukazuje skriptovací blok, který je vázaný v nějaké jiné konstrukci (třeba funkci) a tím je i pojmenován a uložen. My však můžeme blok skriptu uložit i přímo do proměnné.

```

=>[ScriptBlock]$blok = ("Z ulozeného bloku")
=>$blok
"Z ulozeného bloku"

=>$blok.Invoke()
Z ulozeného bloku
=>

```

```
=>$blok | Get-Member
```

```
    TypeName: System.Management.Automation.ScriptBlock
```

```
...
```

Ve všech těchto případech dochází při spuštění k vytvoření onoho „hřiště“, o němž jsme se zmínili výše – jde o pracovní prostor, v němž jsou platné proměnné, sbírány chyby atd. Tento prostor označujeme jako *Runspace*. Pokud chceme, aby nějaká data překročila hranice vymezeného hřiště, musíme je určitým způsobem „prostrčit“ přes hranice – typickou možností jsou proměnné, kterým nastavíme větší rozsah platnosti, než je ten výchozí.



blok_skript.ps1

```
=>cat .\blok_skript.ps1
$Global:vzkaz = "Zevnitř skriptu"
Write-Host $vzkaz
```

```
=>.\blok_skript.ps1
Zevnitř skriptu
```

```
=>$vzkaz
Zevnitř skriptu
=>
```

```
Administrator: C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell_ise.exe
File Edit View Debug Help
adonet.ps1 Search_copy_pst_v.01.ps1 X
1 #search for PST and copy it out
2
3 $stop = "c:\\"
4 $target_top = "d:\"
5
6 if (-not (test-path $stop))
7 {
8
9     write-host "The source folder/drive does not exist; quitting."
10
11 }
12
13 $pst_files = dir c:\ -Include *.pst -Recurse
14
15 ForEach ($pst in $pst_files)
16 {
17
18     $parent = [string]$pst.fullname
19     $original_path = $parent
20     $filename = [string]$pst.name
21
22
23
PS C:\Documents and Settings\Patrik>
Ln 1 Col 1 14
```

Obrázek 3.2: Grafický nástroj PowerShell ISE velmi názorně demonstruje, jak mají skripty přiřazeny své vlastní „hřiště“ – každá záložka představuje samostatný *Runspace*

V našem příkladu tedy proměnná překročila původní hranice – jako globální entita je dostupná i v nadřazeném „hřišti“, odkud byl skript (podřízené „hřiště“) zavolán. Stejným způsobem pracují třeba funkce a filtry.

Všechny roury a kanály PowerShellu

Čtenář třeba není úplným začátečníkem v oblasti skriptování, takže si možná přinesl určité znalosti o použití příkazové konzoly a programu Cmd.exe. Příkazy (programy) v konzole i cmdlety PowerShellu pracují na první pohled stejně při předávání dat. Načítají data z tzv. standardního vstupu a vysílají je na standardní výstup. Krom toho většinou posílají chybová hlášení do tzv. chybového kanálu, který proudí nezávisle na datech, pokud neřekneme jinak. Postaru tedy pracujeme v zásadě se třemi kanály: vstupním, výstupním a chybovým. PowerShell vnáší do této ustálené situace určitý zajímavý pokrok. Jeho cmdlety totiž dokáží zasílat data do více směrů, které jsou přesně pojmenovány a určeny, a využívají tak více výstupních kanálů, s nimiž lze dále pracovat. Vstupní a výstupní datový kanál zůstávají nejdůležitějšími, ale jsou zde i další, které se mohou hodit.

Prozkoumejme několik situací. Zkusíme restartovat nějakou službu a uvidíme, jak se příkaz zachová.

```
=>Restart-Service spooler
WARNING: Waiting for service 'Zařazování tisku (spooler)' to finish starting...
=>
```

```
=>Restart-Service spooler -PassThru
WARNING: Waiting for service 'Zařazování tisku (spooler)' to finish starting...
```

Status	Name	DisplayName
Running	spooler	Zařazování tisku

```
=>Restart-Service spooler | Out-Null
WARNING: Waiting for service 'Zařazování tisku (spooler)' to finish starting...
=>
```

```
=>Restart-Service spooler -PassThru | out-null
WARNING: Waiting for service 'Zařazování tisku (spooler)' to finish starting...
=>
```

PowerShell se na první pohled chová podivně. Ačkoliv jsme opakovaně zakončili tok dat rourou pomocí příkazu *Out-Null*, varování je nepřekonatelné a stále se objevuje. Znamená to, že varování sice končí v naší textové konzole, ale neprošlo rourou jako ostatní data – konzola je tedy „konečnou stanicí“ pro varovnou informaci, avšak ta zjevně do konzoly přišla nezávislým kanálem.

```

Administrator: C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
'C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\BitsTransfer\BitsTransfer.psd1'.
VERBOSE: Loading 'Assembly' from path
'C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\BitsTransfer\Microsoft.BackgroundIntelligentTransfer.Management.Interop.dll'.
VERBOSE: Loading 'FormatsToProcess' from path
'C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\BitsTransfer\BitsTransfer.Format.ps1xml'.
VERBOSE: Importing cmdlet 'Add-BitsFile'.
VERBOSE: Importing cmdlet 'Remove-BitsTransfer'.
VERBOSE: Importing cmdlet 'Complete-BitsTransfer'.
VERBOSE: Importing cmdlet 'Get-BitsTransfer'.
VERBOSE: Importing cmdlet 'Start-BitsTransfer'.
VERBOSE: Importing cmdlet 'Resume-BitsTransfer'.
VERBOSE: Importing cmdlet 'Set-BitsTransfer'.
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Add-BitsFile'.
VERBOSE: Exporting cmdlet 'Remove-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Complete-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Get-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Start-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Resume-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Set-BitsTransfer'.
VERBOSE: Exporting cmdlet 'Suspend-BitsTransfer'.
VERBOSE: Importing cmdlet 'Add-BitsFile'.
VERBOSE: Importing cmdlet 'Complete-BitsTransfer'.
VERBOSE: Importing cmdlet 'Get-BitsTransfer'.
VERBOSE: Importing cmdlet 'Remove-BitsTransfer'.
VERBOSE: Importing cmdlet 'Resume-BitsTransfer'.
VERBOSE: Importing cmdlet 'Set-BitsTransfer'.
VERBOSE: Importing cmdlet 'Start-BitsTransfer'.
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.
=>Import-Module BitsTransfer -Verbose
VERBOSE: Importing cmdlet 'Add-BitsFile'.
VERBOSE: Importing cmdlet 'Complete-BitsTransfer'.
VERBOSE: Importing cmdlet 'Get-BitsTransfer'.
VERBOSE: Importing cmdlet 'Remove-BitsTransfer'.
VERBOSE: Importing cmdlet 'Resume-BitsTransfer'.
VERBOSE: Importing cmdlet 'Set-BitsTransfer'.
VERBOSE: Importing cmdlet 'Start-BitsTransfer'.
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.
=>Import-Module BitsTransfer -ueyhd
Import-Module : A parameter cannot be found that matches parameter name 'ueyhd'.
At line:1 char:34
+ Import-Module BitsTransfer -ueyhd <<<<
    + CategoryInfo          : InvalidArgument: (:) [Import-Module], ParameterBindingException
    + FullyQualifiedErrorId : NamedParameterNotFound,Microsoft.PowerShell.Commands.ImportModuleCommand
=>

```

Obrázek 3.3: Textová konzola je výchozím výstupním „zobrazovačem“ pro všechny kanály, jež PowerShell ovládá. Pokud neřekneme jinak, všechna data směřují sem.

Vyzkoušejme zase jiný pokus. Zkusíme nahrát modul PowerShellu pomocí standardního příkazu a přimějeme jej, aby vypsal podrobnější informaci o celé akci.

```

=>Import-Module BitsTransfer -Verbose
VERBOSE: Importing cmdlet 'Add-BitsFile'.
...
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.

=>Import-Module BitsTransfer -Verbose | Out-Null
VERBOSE: Importing cmdlet 'Add-BitsFile'.
...
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.

```

Ani v tomto případě jsme přesměrováním roury do „černé díry“ ničeho nedosáhli, neboť podrobný výpis stále končí v naší konzole. I tento kanál tedy směřuje svůj výpis do textové konzoly, avšak i v tomto případě je to pouze „závěrečný zobrazovač“ a zpráva přišla jinudy, než kudy běžně proudí data.

Předchozí příklady ukazují, že cmdlety PowerShellu mohou komunikovat paralelně pomocí několika různých datových proudů (kanálů). To však neznamená, že těmito

kanály vždy posílají data – jde o teoretickou možnost a mnohé cmdlety zdaleka nevyužívají celou šíři těchto možností.



Poznámka: Na tomto místě se možná naskytá otázka, proč se vlastně PowerShell chová při zasílání svých dat tak, jak nyní popisujeme. Proč máme hned několik výstupních kanálů namísto jednoho, kterým by klidně mohlo proudit všechno, obzvláště když to nakonec skončí v textové konzole? Jak jsme viděli, přináší to vlastně pouze problémy!

Inu, několik dobrých důvodů se jistě najde. V první řadě, několik kanálů máme k dispozici právě proto, aby se nám různé výstupy nepletly do jednoho guláše! PowerShell se snaží držet čistá „výstupní“ data – požadovaná informace je zaslána hlavním výstupním datovým kanálem (tedy tím, co označujeme jako roura). Pokud cmdlety produkují dodatečné informace, jsou tyto schválně izolovány, aby nám požadovaná data „neznečišťovaly“.

S tím souvisí další možnost, a to nasměrovat tyto dodatečné kanály odděleně do vlastních proměnných. Můžeme tak získat specifická data v čisté, oddělené podobě. Další výhodou je, že můžeme řídit bezprostřední výstup pro každý kanál odděleně, a tak ovládat, jaká data nakonec skončí ve kterém výstupním proudu (a třeba v textové konzole). Na závěr uvedme zdaleka ne poslední vymoženost: do jednotlivých kanálů můžeme odděleně zapisovat vlastní data pomocí sady odpovídajících cmdletů.

S výstupními kanály lze různě pracovat a možnosti jsou shrnuty v tabulce 3.1. Do výstupních kanálů lze zapisovat data dle našeho přání, a obráceně: namísto do konzoly můžeme různé výstupní kanály přeměrovat do proměnných. Povšimněme si také sloupců, jež uvádějí Řídící proměnnou a Jednotlivé řízení – hlavní proměnná určuje chování výstupního kanálu pro aktuální rozsah platnosti (relaci v konzole, skript...), kdežto přepínače u jednotlivých příkazů vynucují chování právě pro daný cmdlet bez ohledu na globální nastavení (a mají tedy přednost).

Tabulka 3.1: Přehled datových kanálů PowerShellu a odpovídajících proměnných, cmdletů a přepínačů

Kanál	Zápis	Řídící proměnná	Do proměnné	Jednotlivé řízení
StdOut	Write-Host		-OutVariable	
	Write-Output			
StdIn	Read-Host			
Error	Write-Error	ErrorActionPreference	-ErrorVariable	-ErrorAction
Debug	Write-Debug	DebugPreference		-Debug
Verbose	Write-Verbose	VerbosePreference		-Verbose
Warning	Write-Warning	WarningPreference	-WarningVariable	-WarningAction
Progress	Write-Progress	ProgressPreference		

Ukažme si prakticky, jaký je vztah mezi těmito nastaveními.

```
=>$VerbosePreference  
SilentlyContinue
```

```
=>Import-Module BitsTransfer
```

```

=>$VerbosePreference = "Continue"

=>Import-Module BitsTransfer
VERBOSE: Importing cmdlet 'Add-BitsFile'.
VERBOSE: Importing cmdlet 'Complete-BitsTransfer'.
VERBOSE: Importing cmdlet 'Get-BitsTransfer'.
VERBOSE: Importing cmdlet 'Remove-BitsTransfer'.
VERBOSE: Importing cmdlet 'Resume-BitsTransfer'.
VERBOSE: Importing cmdlet 'Set-BitsTransfer'.
VERBOSE: Importing cmdlet 'Start-BitsTransfer'.
VERBOSE: Importing cmdlet 'Suspend-BitsTransfer'.

```

Na počátku jsme si prověřili nastavení řídicí proměnné, která určuje, zda se podrobný výpis (Verbose) má zasílat do konzoly, a její určující funkci jsme ověřili spuštěním příkazu (už víme, že přepínač *-Verbose* by vše změnil, neboť má přednost). Následně je vidět změnu chování spolu se změnou proměnné.



Tip: Přepínače uvedené v naší tabulce patří do kategorie „obecných“ přepínačů, které se vyskytují univerzálně u řady příkazů. Podrobnosti se o nich dočteme ve vynikajícím tématu nápovědy PowerShellu: `Get-Help about_CommonParameters`.



Obrázek 3.4: Kanál *Progress* je schopen vykreslit do konzoly stavový pruh s průběžnou informací o probíhající operaci

Řídicí proměnné mají vliv nejen na cmdlety, jež různé kanály využívají, ale také na samotné příkazy zápisu do těchto kanálů, jak si můžeme ověřit.

```

=>$DebugPreference = "silentlycontinue"
=>$DebugPreference
SilentlyContinue
=>Write-Debug "Debug"
=>
=>$DebugPreference = "continue"
=>Write-Debug "Debug"
DEBUG: Debug

```

V tuto chvíli dodejme, že PowerShellu vlastně chybí jednoduchý mechanismus, pomocí kterého bychom mohli snadno vybrat jednotlivé kanály, sloučit je v jeden proud dat a přeměrovat je třeba do výstupního souboru. Takovouto operaci můžeme udělat, ovšem pokročilejším způsobem, jak si ukážeme dále.



Tip: Zapáleným zájemcům k podrobnějšímu prozkoumání této problematiky autor doporučuje perfektní skript Oisina Grehana, který najdete na jeho blogu. Jde o modul, který právě umožňuje soustředit všechny kanály do jednoho výstupu. Jde o krásnou ukázkou využití pokročilých nových možností PowerShellu 2.

<http://www.nivot.org/Trackback.aspx?guid=21136a8a-636d-4665-8637-e93a24bbd61d>

Uchopení příkazového bloku

V předchozích příkladech jsme si ukazovali skriptovací blok, jeho ohraničení a také význam pro vytvoření rozsahu (scopu) proměnných či třeba chybových stavů. V běžných situacích považujeme skriptovací blok prostě za automatickou součást našich řešení – ve smyčce *ForEach* prostě udělá opakovaně to, co potřebujeme, za podmínkou *If* se provede podmíněná část kódu, a pokud je skriptovací blok založen jako celý spustitelný skript, tak prostě předpokládáme, že po jeho spuštění se uvnitř odehraje vše na „připraveném hřišti“.

Mnohokrát zdůrazníme v této knize, že PowerShell je postaven na objektovém přístupu, a to důsledně – proto i příkazový blok je vlastně založen jako objekt, a my se o o tom můžeme nejen přesvědčit, ale můžeme toho případně i využít. PowerShell 2 přinesl krom viditelných rozšíření i některá méně zjevná – a jedním z nich je i třída *PowerShell*, která je novou součástí knihoven tvořících PowerShell jako takový. Tato třída je určena ke tvorbě skriptovacích bloků, které jakoby uchopíme „zvenčí“ – celý skriptovací blok založíme jako objektovou proměnnou a posléze budeme se skriptem pracovat pomocí vlastností a metod této proměnné. Začněme jednoduchým příkladem.

```
=>$blok1 = [powershell]::Create()
=>$blok1 | Get-Member
```

```
TypeName: System.Management.Automation.PowerShell
```

Name	MemberType	Definition
InvocationStateChanged	Event	System.EventHandler`1[System.Management.Au
AddArgument	Method	powershell AddArgument(System.Object value
AddCommand	Method	powershell AddCommand(string cmdlet), powe
AddParameter	Method	powershell AddParameter(string parameterNa
AddParameters	Method	powershell AddParameters(System.Collection
AddScript	Method	powershell AddScript(string script), power
BeginInvoke	Method	System.IAsyncResult BeginInvoke(), System.
BeginStop	Method	System.IAsyncResult BeginStop(System.Asyn
CreateNestedPowerShell	Method	powershell CreateNestedPowerShell()
Dispose	Method	System.Void Dispose()
EndInvoke	Method	System.Management.Automation.PSDataCollect
EndStop	Method	System.Void EndStop(System.IAsyncResult as

Equals	Method	bool Equals(System.Object obj)
GetHashCode	Method	int GetHashCode()
GetType	Method	type GetType()
Invoke	Method	System.Collections.ObjectModel.Collection[
Stop	Method	System.Void Stop()
ToString	Method	string ToString()
Commands	Property	System.Management.Automation.PSCommand Com
InstanceId	Property	System.Guid InstanceId {get;}
InvocationStateInfo	Property	System.Management.Automation.PSInvocationS
IsNested	Property	System.Boolean IsNested {get;}
Runspace	Property	System.Management.Automation.Runspaces.Run
RunspacePool	Property	System.Management.Automation.Runspaces.Run
Streams	Property	System.Management.Automation.PSDataStreams

Ve výpisu členů objektu dokážeme rozpoznat přinejmenším některé „podezřelé“, takže je zkusíme prověřit. Třeba metoda *AddCommand* napovídá, že pomocí ní do bloku vložíme příkaz, a třeba metoda *AddScript* pak učiní totéž s větším celkem. Pojdme to vyzkoušet.

```
=>$blok1.AddCommand("Get-Process")
```

```
Commands          : System.Management.Automation.PSCommand
Streams           : System.Management.Automation.PSDataStreams
InstanceId        : 75ec5575-b205-45a9-85b9-01b26f6147b7
InvocationStateInfo : System.Management.Automation.PSInvocationStateInfo
IsNested         : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
RunspacePool     :
```

```
=>$blok1.Commands
```

```
Commands
-----
{Get-Process}
```

V tuto chvíli tedy objektová proměnná reprezentující spustitelný blok v PowerShellu obsahuje jeden příkaz, který čeká na spuštění. Blok spustíme následujícím způsobem:

```
=>$blok1.Invoke()
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
-----	-----	-----	-----	-----	-----	---	-----
111	5	1672	108	35	0,75	3344	alg
52	2	584	452	18	1,72	1168	ati2evxx
45	2	568	536	18	1,22	3784	ati2evxx
99	3	3112	924	35	4,48	2712	atiptaxx
342	126	4960	1352	56	11,84	288	BTStackServer
147	3	1952	1040	50	19,05	3524	BTTray

To, co jsme vytvořili, je tedy vlastně skriptovací blok „na dálkové ovládání“ – máme jej v objektové proměnné a další řízení je v našich rukou. Skriptovací blok může samozřej-

mě obsahovat nejen jednotlivý příkaz, ale celou sekvenci. Založíme si zdrojový kód jako textový soubor a postupně jej do objektu vložíme.



script.txt

```
=>$blok1.Commands.Clear()

=>cat .\script.txt
$sluzba = Get-WmiObject win32_service -Filter "name='spooler'"
$sluzba.StopService()
$sluzba.StartService()
Write-Host "Restartovano"

=>$script = ((cat .\script.txt) -join "`n" )
=>$scriptobjekt = $executioncontext.InvokeCommand.NewScriptBlock($script)

=>$blok1.AddScript($scriptobjekt) | Out-Null
=>$blok1.Commands

Commands
-----
{{ $sluzba = Get-WmiObject win32_service -Filter "name='spooler'" $sluzba.StopService() $sluzba....

=>$blok1.Invoke()

__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS    :
__DYNASTY       : __PARAMETERS
__RELPATH       :
__PROPERTY_COUNT : 1
__DERIVATION    : {}
__SERVER        :
__NAMESPACE     :
__PATH          :
ReturnValue      : 5

__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS    :
__DYNASTY       : __PARAMETERS
__RELPATH       :
__PROPERTY_COUNT : 1
__DERIVATION    : {}
__SERVER        :
__NAMESPACE     :
__PATH          :
ReturnValue      : 10
```

PowerShell Members (System.Management.Automation)
msdn

MSDN Home

MSDN
MSDN Library
Win32 and COM Development
Administration and Management
Windows PowerShell
Windows PowerShell SDK
Windows PowerShell Managed Ref ...
System.Management.Automation
PowerShell Class

PowerShell Class

PowerShell Members
PowerShell Methods
PowerShell Properties
PowerShell Events

Switch View : [Classic](#) [Lightweight Beta](#) [ScriptFree](#) | [Feedback](#)

Provides methods that are used to create a pipeline of commands and invoke those commands either synchronously or asynchronously within a runspace. This class also provides access to the output streams that contain data that is generated when the commands are invoked. This class is primarily intended for host applications that programmatically use Windows PowerShell to perform tasks. This class is introduced in Windows PowerShell 2.0.

The following tables list the members exposed by the PowerShell type.

Public Properties

Name	Description
Commands	Gets and sets the commands of the pipeline invoked by the PowerShell object. This property is introduced in Windows PowerShell 2.0.
InstanceId	Gets the unique identifier for this instance of the PowerShell object. This identifier is primarily used for logging purposes. This property is introduced in Windows PowerShell 2.0.
InvocationStateInfo	Gets information about the current state of the invocation of the pipeline, such as whether it is running, completed, or failed. This property is introduced in Windows PowerShell 2.0.
IsNested	Gets a Boolean value that indicates whether the current PowerShell object is nested within a parent PowerShell object. This property is introduced in Windows PowerShell 2.0.
Runspace	Gets and sets the runspace used when the pipeline is invoked. The runspace is the operating environment that defines which commands, variables, and other elements are available. This property is introduced in Windows PowerShell 2.0.
RunspacePool	Gets and sets the runspace pool used by the PowerShell object. A runspace from this pool is used whenever the PowerShell object pipeline is invoked. This property is introduced in Windows PowerShell 2.0.
Streams	Gets the data streams that contain any messages and error reports that were generated when the pipeline of the PowerShell object is invoked.

Top

Public Methods
(see also Protected Methods)

Name	Description
AddArgument	Adds an argument for a positional parameter of a command without specifying the parameter name. This method is introduced in Windows PowerShell 2.0.
AddCommand	Overloaded. Adds a cmdlet to the end of the pipeline of the PowerShell object. The variants of this method can add the command with or without specifying that the command should be run within a new local scope. This method is introduced in Windows PowerShell 2.0.
AddParameter	Overloaded. Adds a parameter to the last command of the pipeline. This method is introduced in Windows PowerShell 2.0.
AddParameters	Overloaded. Adds parameters to the last command of the pipeline. The variants of this method can specify the parameters as a dictionary or as a list. This method is introduced in Windows PowerShell 2.0.
AddScript	Overloaded. Adds a script to the end of the pipeline of the PowerShell object. This method is introduced in Windows PowerShell 2.0.
BeginInvoke	Overloaded. Asynchronously starts running the commands of the PowerShell object pipeline. This method is introduced in Windows PowerShell 2.0.
BeginStop	Asynchronously stops the currently running execution of the pipeline. If execution of the pipeline is not started, the state of the PowerShell object is changed to Stopped and the corresponding events are raised. This method is introduced in Windows PowerShell 2.0.

Obrázek 3.5: PowerShell a jeho architektura jsou výborně popsány v dokumentační knihovně MSDN. Není problém si najít detailní popis cmdletů a dalších potřebných tříd.

Za zmínku určitě stojí objektová proměnná *\$executioncontext*, kterou jsme využili pro převod textového zdrojového souboru na prováděcí blok. Tato proměnná (znalci odpustí mírné zjednodušení) vlastně reprezentuje některé schopnosti interpretru PowerShellu, mezi něž patří i vyhodnocení skriptovací sekvence příkazů. Tím si tedy uložíme do proměnné *\$scriptobjekt* funkční blok skriptu, který posléze můžeme spustit. Pro úplnost dodejme, že jeho provedení bychom mohli zařídit třeba i následujícím způsobem:

```
=>Invoke-Command -ScriptBlock $scriptobjekt | select returnValue | ft -AutoSize
```

```
Restartovano
returnValue
-----
0
0
```

Ukažme si nyní jednu z nejzajímavějších možností, kvůli které stojí za to uchopit skriptovací blok „do vlastních rukou“. Naše objektová proměnná, která vznikla dle třídy *PowerShell*, totiž nabízí přístup ke všem datovým kanálům. Použijeme k tomu jiný zdrojový kód, který bude vypadat takto:



streams.ps1

```
=>cat .\streams.ps1
$VerbosePreference = "continue"
$DebugPreference = "continue"
```

```

$WarningPreference = "continue"
$errorActionPreference = "continue"

Write-Verbose "Start"
Write-Host "Startuji..."
Write-Verbose "Ctu data"
$verze = (Get-Host).version
Write-Verbose "Chystam vypis"
Write-Debug "Verze nactena"
Write-Warning "Pozor, zobrazime verzi PowerShellu!"
Write-Verbose "Vypis"
Write-Host $verze
Write-Host $verze #zamerna chyba
Write-Verbose "Hotovo"
Write-Debug "Konec uspesne"

```

Náš testovací skript tedy bude posílat výstupní informace všemi možnými kanály, jak to jen bude možné.



streams.ps1

```

=>$blok = [powershell]::Create()
=>$script = ((cat .\streams.ps1) -join "`n" )
=>$scriptobjekt = $executioncontext.InvokeCommand.NewScriptBlock($script)
=>$blok.AddScript($scriptobjekt) | Out-Null
=>$blok.Commands

```

```

Commands
-----
{$VerbosePreference = "continue"...

```

Náš zdrojový kód jsme tedy připravili jako prováděcí kód do skriptovacího bloku v objektové proměnné. Pojdme se nyní podívat na stav datových kanálů a také na to, jak se jejich obsah bude měnit.

```
=>$blok.Streams
```

```

Error      : {}
Progress   : {}
Verbose    : {}
Debug      : {}
Warning    : {}

```

```

=>$blok.Invoke()
=>$blok.Streams

```

```

Error      : {Cannot invoke this function because the current host does not implement it., Cannot invoke this function because the current host does not implement it., The term 'Writte-Host' is not recognized as the name of a cmdlet, function, 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.}

```

```
Progress : {}
Verbose  : {Start, Ctu data, Chystam vypis, Vypis...}
Debug    : {Verze nactena, Konec uspesne}
Warning  : {Pozor, zobrazime verzi PowerShellu!}
```

Výpis krásně ukazuje, že jednotlivé proudy jsou striktně odděleny a jsou v nich uložena data tak, jak jsme je posílali do toho kterého kanálu. Podívejme se podrobně třeba na chyby:

```
=>$blok.Streams.Error
```

```
Write-Host : Cannot invoke this function because the current host does not implement it.
At line:7 char:11
+ Write-Host <<<< "Startuji..."
    + CategoryInfo          : NotImplemented: (:) [Write-Host], HostException
    + FullyQualifiedErrorId : HostFunctionNotImplemented,Microsoft.PowerShell.Commands.WriteHostCommand
```

Chybové hlášení nám sděluje dosti důležitou informaci: cmdlet *Write-Host* nezpracoval, jak jsme očekávali, a textový výstup se skutečně neobjevil. Je to proto, že instance třídy *PowerShell* zasílá data do roury, kde se očekává další zpracování. Zkusme tedy náš výchozí kód upravit, znovu spustit a následně prozkoumat datové proudy.



streams2.ps1

```
=>cat .\streams2.ps1
$VerbosePreference = "continue"
$DebugPreference = "continue"
$WarningPreference = "continue"
$errorActionPreference = "continue"

Write-Verbose "Start"
Write-Output "Startuji..."
Write-Verbose "Ctu data"
$verze = (Get-Host).version
Write-Verbose "Chystam vypis"
Write-Debug "Verze nactena"
Write-Warning "Pozor, zobrazime verzi PowerShellu!"
Write-Verbose "Vypis"
Write-Output $verze
Write-Host $verze #zamerna chyba
Write-Verbose "Hotovo"
Write-Debug "Konec uspesne"

=>$blok3 = [powershell]::Create()
=>$script = ((cat .\streams2.ps1) -join "`n" )
=>$scriptobjekt = $executioncontext.InvokeCommand.NewScriptBlock($script)
=>$blok3.AddScript($scriptobjekt) | Out-Null

=>$blok3.Streams

Error : {}
```

```
Progress : {}
Verbose  : {}
Debug    : {}
Warning  : {}
```

```
=>$blok3.Invoke()
```

```
Startuji...
```

```
Major  Minor  Build  Revision
-----  -----  -----  -----
2      0      -1     -1
```

```
=>$blok3.Streams
```

```
Error      : {The term 'Writte-Host' is not recognized as the name of a cmdlet, func-
            tion, 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.}
```

```
Progress : {}
```

```
Verbose  : {Start, Ctu data, Chystam vypis, Vypis...}
```

```
Debug    : {Verze nactena, Konec uspesne}
```

```
Warning  : {Pozor, zobrazime verzi PowerShellu!}
```

Nyní již naše řešení pracuje perfektně, zcela dle předpokladů. Chybový kanál zachytil jednu chybu (záměrně chybně zapsaný příkaz) a ostatní kanály obsahují přesně informaci, kterou jsme jimi zaslali. Také je vidět, že provedení kódu vrátilo kýžená data – zaměnili jsme příkaz *Write-Host*, který pracuje s výslovným výpisem do konzoly, za příkaz *Write-Output*, který výslovně posílá data do roury. Zdá se to jako malý rozdíl, ale je to zcela zásadní odlišnost: z roury lze data dál číst a zpracovávat, v konzole však už máme jen výsledný text k zobrazení. Mohli bychom totiž pokračovat třeba takhle:

```
=>($blok3.Invoke() | measure).count
```

```
2
```

```
=>$blok3.Invoke() | Get-Member
```

```
    TypeName: System.String
```

```
...
```

```
    TypeName: System.Version
```

```
Name          MemberType Definition
```

```
----
```

```
-----
```

```
.....
```

```
Build          Property  System.Int32 Build {get;}
Major          Property  System.Int32 Major {get;}
MajorRevision  Property  System.Int16 MajorRevision {get;}
Minor          Property  System.Int32 Minor {get;}
MinorRevision  Property  System.Int16 MinorRevision {get;}
Revision       Property  System.Int32 Revision {get;}
```

Jinými slovy, náš skript poslal na výstup dva objekty. První z nich byl prostý text (hlášení „Startuji...“), druhý pak získaná data (verze PowerShellu). Další zpracování za rourou je jen na nás.

Okolnosti spuštění bloku – \$MyInvocation

Předcházející příklady ukázaly, že blok skriptu v PowerShellu můžeme uchopit jako objekt, stejně jako ostatní data. Možná tuto vymoženost nevyužijeme příliš často, přesto se však nabízí jeden zajímavý důsledek architektury PowerShellu, který se nám může hodit. Po spuštění konzoly, skriptu, obecně každého bloku, který má svůj rozsah působnosti, se vytvářejí některé výchozí proměnné PowerShellu a ukládají se do nich určité hodnoty. Jedna z nich se jmenuje *\$MyInvocation* a nese vlastně svědectví o tom, za jakých okolností byl skriptovací blok spuštěn. Toho můžeme v některých situacích využít.

Při tvorbě skriptů někdy potřebujeme získat odkaz na adresář, ze kterého byl původní skript spuštěn. Běžně se stává, že chceme z daného adresáře načítat další soubory, případně do něj ukládat nějaké exporty či záznamy událostí (logy). Při sestavování adresářových cest k takovým souborům se nám hodí, pokud víme, odkud jsme začali.

Prozkoumejme nejdříve první možnost – chceme v našem skriptu používat cestu místa, odkud jsme jej zavolali. Pak to provedeme tak, že sáhneme na jednu z automatických proměnných PowerShellu:



script.ps1

```
=>pwd
Path
----
C:\Patrik\Work\PowerShellBook2\temp\3\start

=>cat .\script.ps1

$MyInvocation.MyCommand | fl *

=>.\script.ps1

HelpUri      :
Path         : C:\Patrik\Work\PowerShellBook2\temp\3\start\script.ps1
Definition   : C:\Patrik\Work\PowerShellBook2\temp\3\start\script.ps1
Visibility   : Public
ScriptBlock  : $MyInvocation.MyCommand | fl *
OutputType   : {}
ScriptContents : $MyInvocation.MyCommand | fl *
OriginalEncoding : System.Text.SBCSCodePageEncoding
Name         : script.ps1
CommandType  : ExternalScript
ModuleName   :
Module       :
Parameters   : {}
ParameterSets : {}
```

Pouze kýženou cestu tedy získáme takto:

```
=>cat .\script.ps1
($MyInvocation.MyCommand).Path

=>.\script.ps1
C:\Patrik\Work\PowerShellBook2\temp\3\start\script.ps1
```