



Windows PowerShell 3.0 für Einsteiger 1

Übersicht über Windows PowerShell 3.0.

Erste Übungen mit Windows PowerShell-Objekten

Autor: Frank Koch, Microsoft Deutschland

Information in this document, including URLs and other Internet Web site references, are subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Kapitel: Erste Übungen mit Windows PowerShell Objekten

Für Programmierer sind Objekte nichts Neues, für Systemadministratoren und beim Skripting ist dies unter Umständen jedoch ein unbekanntes Terrain. Bei mehr Interesse am Thema Objektorientierte Programmierung findet sich umfangreiche Sekundärliteratur z.B. auf den MSDN Seiten von Microsoft unter <http://msdn.microsoft.com>. Für die Windows PowerShell ist es nur wichtig zu wissen, dass Objekte KEINE einfachen Texte sind, die man selber interpretieren muss, sondern „Dinge“ (Objekte), mit denen PowerShell direkt weiterarbeiten kann. Schauen wir uns die Arbeit mit Objekten doch am Beispiel des „process“-Objekts an. Sollte Ihnen Prozesse auf Ihrem Rechner kein Begriff sein, stellen Sie sich dies einfach als die Programme und Anwendungen vor, die Sie im Taskmanager auf Ihrem System aufgelistet bekommen.

Die Arbeit mit Systemprozessen

Der Befehl *get-process* listet alle Prozesse auf Ihrem System auf. Diese Liste kann sehr lang sein. Um die Liste zu sortieren kann man ein weiteres Cmdlet nutzen: *sort-object*. *Sort-object* sortiert standardmässig alle Objekte aufsteigend; wenn man die Reihenfolge umdrehen möchte, kann man den Parameter *-descending* verwenden. Als Argument gibt man die Objekteigenschaften (auch „property“ genannt) an, nach denen sortiert werden soll, z.B. die CPU Zeit. Während Parameter immer mit einem Bindestrich beginnen, fällt dieser für Argumente weg. So kann man diese einfacher unterscheiden.



A1: Ihre Aufgabe ist es, eine Liste aller Prozesse absteigend zu erstellen und diese nach deren CPU Nutzung zu sortieren. Hierzu benötigen Sie lediglich die bereits bekannten Cmdlets: *get-process*, *sort-object*, und die Pipe *|*. Hinweis: „CPU“ ist kein Parameter sondern das Argument zum Sortieren. Nutzen Sie *get-help*, wenn Sie unschlüssig sind.

In der nächsten Übung wollen wir die Liste ein wenig einschränken, um sie handlicher zu machen. Hierzu verwenden wir den Befehl *select-object*. *Select-object* kennt mehrere Parameter (verwenden Sie *get-help* um diese herauszubekommen), wir benutzen aber nur *-first x* und *-last y*, um die ersten x oder die letzten y Objekte aus einer Liste zu erhalten, z.B. *select-object -first 5*. Übrigens, das Cmdlet *select-object* ohne vorgestellte Objekte funktioniert nicht; das Cmdlet erwartet zwingend eine Übergabe durch die Pipe etc. und kann somit eigentlich nie das erste Cmdlet in einer Befehlszeile sein.

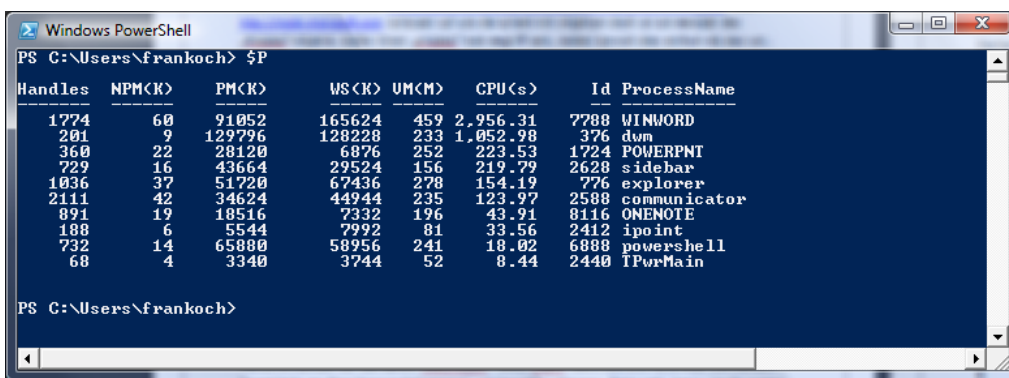


A2: Erzeugen Sie die Liste der Top 10 Prozesse basierend auf deren CPU Zeit. Nehmen Sie dazu das Ergebnis von A1 und erweitern Sie es um den Befehl *select-object*. Für die optimale Lösung gibt es zwei Wege, je nachdem, wie die Liste sortiert werden soll. Versuchen Sie beide Wege zu finden. Hinweis: ein Weg verwendet den Parameter *-first*, der andere *-last*

Wir benutzen diese kleine Übung, um noch schnell Variablen einzuführen. Vereinfacht speichern Variablen alle möglichen Werte, Funktionen, Ausdrücke und sogar auch Objekte! Auch hier ist wieder auf Sekundärliteratur verwiesen, wenn man mehr zum Thema Variablen lernen möchte. Für uns ist es nur wichtig zu wissen, dass Variablen in PowerShell immer mit \$ anfangen müssen. Das Ergebnis der Aufgabe A2 kann man nun in einer Variablen speichern und so auf die Liste der Top 10 Prozesse zu diesem Zeitpunkt später immer wieder zugreifen. Das erlaubt den Vergleich mit anderen Zeitpunkten, um vielleicht auszuwerten, was sich an einem System so alles ändert. Die Zuordnung ist dabei sehr einfach: `$a = get-process | sort-object CPU -descending`



A3: Weisen Sie der Variable \$P die verkürzte Prozessliste aus Aufgabe A2 zu. Hinweise: verwenden Sie die Cursortaste „Hoch“, um den letzten Befehl wieder hervorzurufen und bewegen Sie den Cursor mit *Home* zum Anfang der Zeile, um etwas hinzuzufügen.



```
PS C:\Users\frankoch> $P
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
1774	60	91052	165624	459	2.956.31	7788	WINWORD
201	9	129796	128228	233	1.052.98	376	dwm
360	22	28120	6876	252	223.53	1724	POWERPNT
729	16	43664	29524	156	219.79	2628	sidebar
1036	37	51720	67436	278	154.19	776	explorer
2111	42	34624	44944	235	123.97	2588	communicator
891	19	18516	7332	186	43.91	8116	ONENOTE
188	6	5544	7992	81	33.56	2412	ipoint
732	14	65800	58956	241	18.02	6888	powershell
68	4	3340	3744	52	8.44	2440	TPurMain

```
PS C:\Users\frankoch>
```

Abbildung 1: Ausgabe der Variablen \$P

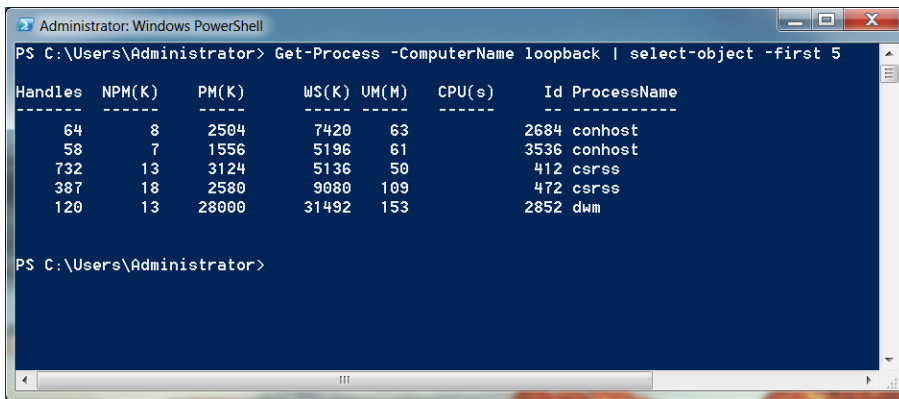
Haben Sie sich die Hilfe zu *Get-process* angeschaut, so sind Sie vielleicht auch über den Parameter *-Computername* gestolpert. Seit der PowerShell 2.0 können einige Cmdlets Informationen nicht nur über den eigenen Computer, sondern auch über das Netz auch von anderen Systemen anzeigen (sofern die Berechtigungen und evtl. Firewalls dazwischen dies erlauben). Der Parameter *-computername* erlaubt als Wert den (DNS-) Namen oder auch die IP Adresse des gewünschten Zielsystems. Sie können dies sehr einfach mit der Loopback-Adresse 127.0.0.1 oder der Eingabe „localhost“ auf Ihrem System einmal testen:



Rufen Sie *Get-process -computername localhost* oder *get-process -computername 127.0.0.1* auf. Vergleichen Sie die Ergebnisse mit dem Aufruf von *get-process* ohne den Parameter. Was fällt Ihnen hierbei auf?

Wie Sie in der Ergebnisliste sehen, werden beim „Remotezugriff“ nicht alle Werte der

Objekte übertragen, sondern lediglich ein Teilbereich. So fehlt z.B. die CPU Zeit. Sollten Sie jedoch auch diese Werte benötigen, gibt es eine Alternative. So kennt PowerShell die generelle Möglichkeit, Cmdlets auf einem anderen System auszuführen und dabei deren Standardfunktionalität beizubehalten. Hierbei werden die Möglichkeiten des WinRM (Windows Remote Management, eine definierte Schnittstelle basierend auf Webservices) genutzt. Mehr hierzu später im Buch.



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-Process -ComputerName loopback | select-object -first 5

Handles  NPM(K)  PM(K)  WS(K)  UM(M)  CPU(s)  Id ProcessName
-----  -
64        8    2504    7420    63      2684  conhost
58        7    1556    5196    61      3536  conhost
732       13    3124    5136    50      412   csrss
387       18    2580    9080    109     472   csrss
120       13    28000   31492    153    2852  dwm
```

Abbildung 2: Ergebniss von Get-Process über das Netz

Die Ausgabe in eine TXT, CSV und XML Datei

Standardmässig gibt Windows PowerShell die Ergebnisse einer Befehlskette auf dem Bildschirm aus. Hierbei werden die Objekte in Text umgewandelt, damit Menschen sie auch lesen können. Dazu dient der Befehl *Out-Host*. Da Windows PowerShell jedoch effizient ist, wird dies automatisch unsichtbar hinzugefügt, wenn man es selber einmal vergessen sollte. Statt *Out-host* gibt es noch weitere Alternativen, schauen Sie selber einmal nach mit *get-help out**.

Die Ausgabe in eine Textdatei kann man damit recht schnell erreichen: *out-file Dateiname* ist die Lösung. In vielen anderen Shells findet sich hierzu auch der Befehl *>* der daher auch in Windows PowerShell unterstützt wird. Dies haben wir bereits beim Erstellen der Hilfetextdatei verwendet. Neben der Ausgabe als Textdatei gibt es auch den Export als CSV oder als XML Datei. Wie *out-Host* gibt es dafür eigene Cmdlets, die dies für Sie erledigen. Sie lauten *Export-CSV* und *Export-CliXML*. Beide verlangen als Argument den Dateinamen und als Eingabe die Objekte, die exportiert werden sollen. Und ja, Sie vermuten richtig: wo ein Export da ein Import. Mit *Import-CSV* oder *Import-CliXML* können Sie auch wieder die Dateien einlesen und z.B. einer Variablen zuweisen.



A4: Nehmen Sie die Variable *\$P* (die gekürzte Liste der Systemprozesse) aus A3 und speichern Sie deren Inhalt in eine Textdatei mit dem Namen „A4.txt“. Im nächsten Schritt exportieren Sie den Inhalt von *\$P* in eine CSV Datei mit dem Namen „A4.CSV“ und in eine XML Datei „A4.XML“. Hinweis: der Befehl *>* ersetzt direkt die Pipe *|*, welche nur für echte Cmdlets wie *Out-File*, *export-csv* etc. nötig ist. Schauen Sie sich die Ergebnisse an, zur Not reicht dafür Notepad. Beachten Sie dabei auch die Größe der jeweiligen Ausgabedatei.

Ausgabe in Farbe

Manchmal möchte man Ausgaben betonen, um das Lesen einfacher zu machen. Dies kann man z.B. durch den Einsatz von Farbe erreichen. Der Befehl *Write-host* kennt dazu mehrere Parameter wie *–foregroundcolor* und *–backgroundcolor*. Was würde wohl folgendes bewirken:

Write-host „Rot auf Blau“ –foregroundcolor red –backgroundcolor blue

Sie ahnen es sicherlich. *Get-help write-host* listet Ihnen die möglichen Farben auf. Mit diesem Befehl könnte man z.B. nun alle Prozesse in Farbe ausgeben. Zusätzlich gibt es auch bereits vordefinierte Farbkombinationen: mit *write-warning „error“* erreichen Sie ebenfalls die nötige Aufmerksamkeit, versuchen Sie es einmal.

Schöner als eine einfarbige Liste wäre es jedoch, wenn man eine Liste anhand von zusätzlichen Bedingungen unterschiedlich einfärben könnte. Das wollen wir uns nun genauer anschauen. Dazu verwenden wir einfachheitshalber statt der Prozesse die Systemdienste (Services) Ihres Rechners. Sollten Sie nicht wissen, was Services sind, schlagen Sie dies bitte wieder im Internet nach, z.B. auf den MSDN Seiten. Vereinfacht sind Services die Objekte, die Sie in der *Systemsteuerung / Verwaltungswerkzeuge / Services* aufgelistet bekommen. Das Schöne an ihnen ist, dass es sie im Zustand „running“ und „stopped“ gibt, was sich hervorragend für Farbausgaben nutzen lässt. Aber zunächst einmal schauen wir uns die Services mit dem Cmdlet *get-service* an.



A5: Erzeugen Sie eine Liste aller Services und sortieren Sie diese nach deren Eigenschaft Status. Hinweis: Nehmen Sie die Lösung der Prozess-Sortierung nach CPU Zeit, verwenden Sie jedoch *get-service* und als Argument für *sort-object* das Wort *status*.

Nun wollen wir die Liste der Services nicht sortieren sondern in roter Schrift ausgeben. *Write-host* soll das ja angeblich können. Leider funktioniert der Befehl *get-service | write-host –foregroundcolor red* nicht wie erhofft. *Write-host* ist leider nicht so nett wie die bisherigen Cmdlets und nimmt eine Liste von Objekten direkt an und gibt diese eingefärbt wieder aus. Hierzu müsste *write-host* für jedes Objekt ja auch wissen, welche Eigenschaften des Objekts denn ausgegeben werden sollen. Wir müssen *write-host* daher helfen. Hierzu wird die Liste der Objekte Schritt für Schritt einzeln abgearbeitet, was man mit einer sogenannten Schleife macht. Es gibt viele Arten von Schleifen, jede hat ihre Berechtigungen und Einsatzgebiete.

Für unseren Zweck verwenden wir die *ForEach-Object* Schleife, welche durch eine Liste von Objekten geht und jedes Objekt einzeln abarbeitet. Innerhalb der Schleife spricht man das jeweilige Objekt unter dem Kürzel *\$_* an, eine „Willkür“ der PowerShell Entwickler seit Version 1.0. Mit der Version PowerShell 3.0 kann man in einigen Fällen darauf zwar verzichten, wie wir in späteren Beispielen sehen werden, da die *\$_* Variante jedoch weiterhin unterstützt wird und auch komplexere Szenarien unterstützt, werden wir uns auf diese Variante konzentrieren.

Eine Eigenschaft des Objekts wird entsprechend mit *\$_.Name-der-Eigenschaft (oder auch property)* angesprochen. Schauen wir uns das in einem Beispiel an:

Get-process | ForEach-Object { write-host \$_.ProcessName \$_.CPU }

Während *get-process* alleine eine Liste aller Prozesse mit vielen Informationen erstellt, zeigt die Ausgabe des Beispiels nur noch den Namen und die CPU Zeit an. Die komplette Objektliste von *get-process* wird über die Pipe (|) an die Schleife *Foreach-Object* weitergereicht. Anschließend wird Objekt für Objekt durchgearbeitet und nur noch die zwei spezifizierten Eigenschaften ausgegeben. (Sollten Sie das Beispiel aufrufen, wundern Sie sich nicht: nicht jeder Prozess hat eine CPU Zeit, so dass unter Umständen eine Zeile auch nur aus dem Namen bestehen kann). Kommen wir nun zum Ausgangsproblem, der farblichen Ausgabe von Texten, zurück.

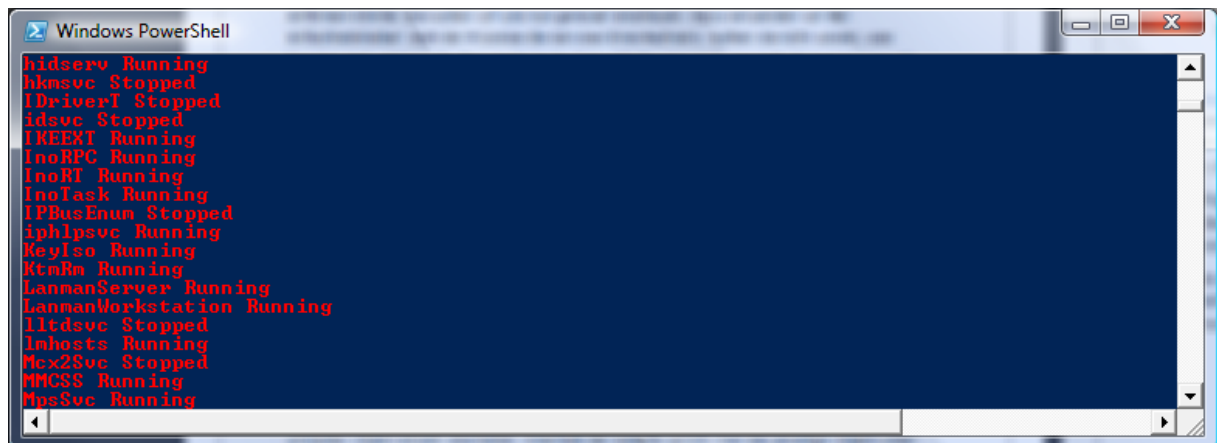


A6: Erzeugen Sie eine Liste aller Services und geben Sie nur die Eigenschaften Name und Status aus. Verwenden Sie hierzu die beschriebene *ForEach* Schleife, auch wenn es andere Lösungen gibt.

Nun können wir die Möglichkeiten von *write-host* nutzen um die Ausgabe in Farbe durchzuführen.



A7: Erzeugen Sie eine Liste der Services und geben Sie nur die Eigenschaften Name und Status in einer Farbkombination Ihrer Wahl aus. Hinweis: Nehmen Sie Ihre Lösung von A6 und ergänzen Sie die Parameter *-foregroundcolor* *-backgroundcolor*.



```
Windows PowerShell
hidsvr Running
hkmsvc Stopped
IDriverT Stopped
idsvc Stopped
IKEEXT Running
InoRPC Running
InoRT Running
InoTask Running
IPBusEnum Stopped
iphlpvc Running
KeyIso Running
KtmRm Running
LanmanServer Running
LanmanWorkstation Running
lltdsvc Stopped
lmhosts Running
Mcx2Svc Stopped
MMCSS Running
MpsSvc Running
```

Abbildung 3: Ausgabe von Servicenamen und Status in Rot

Möchten Sie nur eine Eigenschaft einer Objektliste verwenden, so können Sie ab Windows PowerShell 3.0 das Ganze übrigens vereinfachen. Als Beispiel wollen wir den Namen aller Prozesse ausgeben. Dafür lautet die korrekte Syntax für alle PowerShell Versionen (mit Verzicht aufs *write-host* Cmdlet):

```
Get-process | Foreach-Object { $_.ProcessName }
```

In Windows PowerShell 3.0 können Sie zusätzlich auch folgende Variante nutzen:

```
Get-process | Foreach ProcessName
```

Sie ersparen sich dabei die geschweiften Klammern und das *\$_* Objekt. Zusätzlich wurde die Kurzform *Foreach* für das Cmdlet *Foreach-Object* gewählt, was zwar schon seit PowerShell 1.0 funktioniert, hier das Beispiel aber noch lesbarer macht.

Kapitel 2: Bedingungen überprüfen mit dem *if* Cmdlet

Als letztes Element fehlt uns nur noch das Überprüfen von Bedingungen. Auch hier gibt es mehrere Möglichkeiten für unterschiedliche Einsatzszenarien. In dieser Einführung beschränken wir uns jedoch auf den *IF* Befehl. Die Syntax ist wahrscheinlich schon von anderen Stellen bekannt und ist sehr einfach:

```
If (Bedingung) { dann die auszuführenden Befehle Block 1}  
elseif (Bedingung2) {auszuführende Befehle Block 2}  
else{auszuführende Befehle Block 3}
```

Elseif ist dabei optional und nicht immer nötig. Sollen mehrere Befehle im {} Bereich ausgeführt werden, so kann man sie entweder mit dem Semikolon trennen oder pro Befehl eine neue Zeile benutzen. Windows PowerShell wartet einfach auf die geschweifte Klammer } als Abschluss.

Zum Vergleichen kennt Windows PowerShell mehrere Vergleichsoperatoren. Sie fangen alle mit „-“ an und bestehen in der Regel aus einer englischen Abkürzung mit 2 Buchstaben: *-eq* steht z.B. für equals (gleich). Am wichtigsten für uns sind

-eq	Gleich
-ne	Ungleich
-gt oder -ge	Grösser als / Grösser oder gleich als
-lt oder -le	Kleiner als / Kleiner oder gleich als
-match	„Entspricht“ (nicht ganz so streng wie gleich)
-notmatch	„entspricht nicht“

Die letzte Übung für diesen Block kombiniert nun alle Punkte zu einer ersten Statusüberwachung eines Systems mit Powershell. Alle Services eines Systems werden zunächst sortiert nach ihrem Status und anschliessend in Farbe ausgegeben: Services mit dem Status „stopped“ sollen in Rot, Services mit dem Status „running“ in Grün ausgegeben werden.



A8: Rufen Sie eine Liste aller Services auf. Sortieren Sie diese Liste nach dem Status und färben Sie die Ausgabe entweder in Rot oder Grün, je nachdem, ob der Status des jeweiligen Service „stopped“ oder „running“ ist. Hinweis: Verwenden Sie zunächst *sort-object* von der vorherigen Übung. Verwenden Sie dann die *Foreach* Schleife, aber statt nur *write-host* zu nehmen, bauen Sie eine *If* Abfrage davor ein. Den Status eines Service bekommen Sie wie gewohnt mit *\$_status*, die möglichen Werte sind „stopped“ oder „running“. Zur Syntax: Die *If* Bedingung kommen in runde Klammern (), der Ausgabebefehl in {} Klammern. PowerShell überwacht die gesetzten Klammern und verlangt, dass Sie diese auch schließen. Vergessen Sie nicht die Schlussklammer } des *ForEach Cmdlets*! Wenn Sie am Ende in einer >> Zeile stehen, schliessen Sie diese mit 2x Return ab, um die umgebrochenen Zeilen auszuführen. Ignorieren Sie die weiteren Möglichkeiten neben „stopped“ und „running“ und verzichten Sie einfachheitshalber auf die *Elseif* Abfrage.

Führen Sie anschließend das Beispiel noch einmal aus, diesmal aber ohne das Cmdlet *sort-object*. Verwenden Sie hierzu auch die Cursortasten um nicht alles noch einmal eingeben zu müssen. So erhalten Sie eine bunte Ausgabe ähnlich wie hier gezeigt:

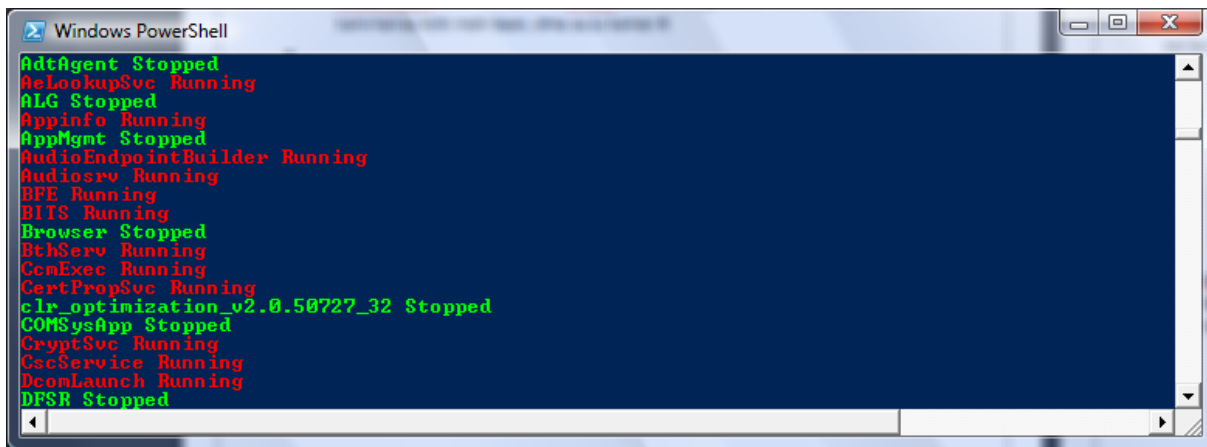


Abbildung 4: Farbige Ausgabe der Services

Foreach-Object kann man wie bereits erwähnt mit *ForEach* abkürzen. Es geht sogar noch kürzer, aber dann kann man es nicht mehr lesen, ohne das Abkürzungszeichen (%) zu kennen. Daher bleiben wir zunächst bei *ForEach-Object* oder *ForEach*.

Mit dem Cmdlet *ForEach* können wir nun auch noch das oben erwähnte Problem mit der Hilfetextdatei unter der Windows PowerShell 3.0 lösen. Sie erinnern sich: unter Windows PowerShell 1.0 und 2.0 kann man zur Erstellung einer eigenen Hilfetextdatei folgendes Skript verwenden:

```
Get-help * | get-help -full > c:\HilfeTextdatei.txt
```

Unter Windows PowerShell 3.0 verursachen leider einige Cmdlets in der Liste noch Fehlermeldungen, die zu einem Abbruch der Schleife führen. Daher muss man hier eine Alternativlösung verwenden, die das Cmdlet *ForEach* verwendet und nun für Sie leicht zu verstehen sein sollte:

```
Get-help * | Foreach { get-help $_.name -full >> c:\HilfeTextdatei.txt }
```

Das zweite *get-help* wird mit dem Namen des jeweiligen Cmdlet Objekts aufgerufen und nicht mit dem Objekt an sich. Das ist leicht zu verstehen, denn genau so tippen Sie es auch in der Kommandozeile ein: *get-help get-service* nutzt den Namen des Cmdlets *get-service*. Oder wie tippen Sie das Objekt *get-service* über die Tastatur ein? Eben.

Der Ausgabebefehl *>>* fügt dabei den Ausgabetext am Ende der existierenden Datei hinzu, so dass diese mit jedem Schleifendurchlauf wächst anstatt immer wieder neu angelegt und überschrieben wird. Wenn die Datei bisher nicht existierte, legt es die Datei dabei auch an. Denken Sie daran, dass Sie das Beispiel daher nur einmal aufrufen, sonst haben Sie den Inhalt mehrfach in der Datei.

Die *Foreach* Lösung funktioniert auch in den vorherigen Versionen der Windows PowerShell einwandfrei. Die Fehlermeldungen in 3.0 treten zwar immer noch für einige Cmdlets auf, führen jedoch nicht mehr zum Abbruch, sondern die Schleife geht einfach zum nächsten Objekt über.

Kapitel 3: Ausgabe von HTML

Zur Überwachung von Servern kann das Beispiel A8 schon etwas helfen. Schön wäre es nun, wenn die Ausgabe leichter weiterverwendet werden könnte. Die Ausgabe als CSV und XML haben Sie schon kennengelernt. Es gibt jedoch noch eine weitere Ausgabe, die manchmal sinnvoll sein kann: die Ausgabe als HTML. Das Cmdlet *convertto-html* ist dafür zuständig. Wie der Name vermuten lässt, wird die Ausgabe nur in HTML konvertiert, aber dabei nicht zusätzlich in eine Datei geschrieben, so dass man den HTML Code noch bearbeiten kann. Am Ende sollte man daher die HTML-Ausgabe in eine Datei umlenken, damit man sie z.B. mit einem Webbrowser betrachten kann. Anhand einer Reihe von kleinen Beispielen werden wir einige Möglichkeiten von *convertto-html* ausloten.



A9 Konvertieren Sie die Ausgabe von *get-service* in HTML. Verwenden Sie dazu das Cmdlet *convertto-html*, welches direkt mit einer List von Objekten arbeiten kann. Hinweis: die umfangreiche Ausgabe können Sie mit CTRL-C abbrechen.



A10 Leiten Sie diesmal die Ausgabe am Ende mit den bekannten Befehlen in die Datei „.A10.html“ um. Betrachten Sie die Datei. Hinweis: Sie können mit *invoke-item .\a10.html* direkt aus Powershell Ihren default-Browser mit der Ausgabedatei starten. Vergessen Sie jedoch nicht den korrekten Pfad zu A10.html mit einzugeben! Wenn es Ihnen lieber ist, öffnen Sie die Datei mit dem Dateieexplorer.

ConvertTo-HTML erlaubt die Wertausgabe einzuschränken, so dass die Liste nicht zu unübersichtlich wird. Hierzu wird *ConvertTo-HTML* einfach mit der Liste der auszugebenden Objekteigenschaften aufgerufen, z.B. ... | *convertto-html -property name, status* .

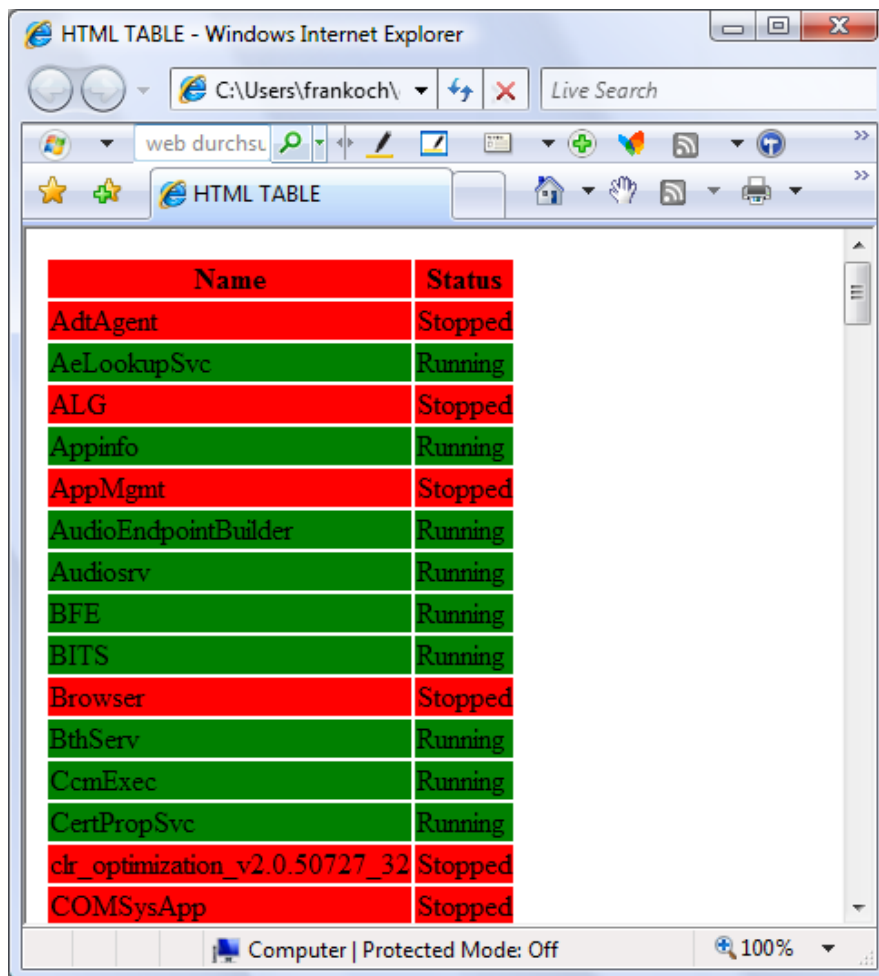


A11: Aufbauend auf A10: Erzeugen Sie eine schönere Webseite und listen Sie lediglich den Namen und den Status aller Services auf. Sie können die Ausgabe auch **vor** der Konvertierung nach dem Status sortieren. Hinweis: Ihre Befehlszeile besteht damit aus 4 Befehlen:
Liste alle Services, sortiere nach Status, konvertieren in HTML, Ausgabe in eine Datei.

Da *ConvertTo-HTML* einen HTML Text erzeugt, kann dieser mittels HTML Kenntnissen angepasst werden. Dies ist dann nicht mehr Aufgabe von Windows PowerShell, kann hiermit jedoch unterstützt werden. Versuchen Sie den folgenden Code zu verstehen, bevor Sie ihn in die Windows PowerShell kopieren oder abtippen und ausführen lassen:

```
get-service | ConvertTo-Html -Property Name,Status | foreach {  
  if ($_ -like "*<td>Running</td>*" ) {$_ -replace "<tr>", "<tr bgcolor=green>"}  
  else {$_ -replace "<tr>", "<tr bgcolor=red>"} } > .\get-service.html
```

Die Ausgabedatei sollte etwa so aussehen wie unten dargestellt. Prinzipiell funktioniert das Beispiel wie die *write-host* Ausgabe, jedoch werden hier die einzelnen Zeilen der HTML Datei umgeschrieben: dem HTML Befehl für die Tabellenspalte wird die Hintergrundfarbe *bgcolor Green* oder *bgcolor red* angefügt. Da nicht jeder HTML kann, wurde das Beispiel ausnahmsweise mit dem Lösungscode direkt angegeben.



Name	Status
AdtAgent	Stopped
AeLookupSvc	Running
ALG	Stopped
Appinfo	Running
AppMgmt	Stopped
AudioEndpointBuilder	Running
Audiosrv	Running
BFE	Running
BITS	Running
Browser	Stopped
BthServ	Running
CcmExec	Running
CertPropSvc	Running
clr_optimization_v2.0.50727_32	Stopped
COMSysApp	Stopped

Abbildung 5: Farbige HTML Ausgabe durch HTML Manipulation

Mit zusätzlichem HTML Wissen können Sie z.B. auch CCS, cascading style sheets, angeben, um das Layout an die eigenen Bedürfnisse anzupassen. Wer nicht fit in HTML ist, kann aber auch bereits mit dem Cmdlet `ConvertTo-HTML` selber etliche Anpassungen vornehmen. Die Hilfe zum Cmdlet zeigt diese auf, eine möchte ich hier noch kurz aufzeigen. So kann in die Lösung oben sehr einfach das Erstellungsdatum, Titel oder Hinweise angegeben werden:

```
get-service | ConvertTo-Html -title "MailServer EX01" -body (get-date) `
-pre "<P>Generated by Corporate IT</P>" -post "<P>For details, contact Corp.IT at `
<a href=http://infoweb> http://infoweb </a>.</P>." -Property Name, Status | foreach {
if ($_ -like "*<td>Running</td>*") {$_ -replace "<tr>", "<tr bgcolor=green>"} `
elseif ($_ -like "*<td>Stopped</td>*") {$_ -replace "<tr>", "<tr bgcolor=red>"} else
{ $_ } } > .\ServiceStatus.html
```