

Kurzübersicht Microsoft PowerShell 1.0 RC

Bei der MS PowerShell (im folgenden nur noch kurz PS genannt) handelt es sich um den grossen Bruder der allseits bekannten Command Shell welche in Ihren Grundzügen bis in die Zeiten von MS-DOS zurückreichen.

Ursprünglich war geplant die PS zusammen mit Microsoft Windows Vista an Endkunden auszuliefern, diese Pläne wurden jedoch verworfen, die PS ist ein eigenständiges Tool welches für Windows XP, 2003 und Vista von MS heruntergeladen und installiert werden kann – wenn man denn das Microsoft.NET Framework in Version 2.0 installiert hat.

Der erste Eindruck

Nach der Installation und dem erstmaligen Starten der PS dauert es einige Momente bis die erste Interaktion mit dieser möglich wird. Anscheinend werden beim Starten sehr viele Klassen des .NET Frameworks geladen und instanziiert, welches zu diesem doch sehr langen Startverhalten führt. Dies wird auch später nicht mehr besser, der start der PS dauert selbst bei einem Warm-Start ungefähr 3 Sekunden – wesentlich länger als der ordinäre Command Prompt.

Das erste Reinschnuppern mit altbekannten Befehlen wie beispielsweise ‚dir‘, ‚copy‘ und ‚cd‘ gestaltet sich als wenig problematisch. Die meisten Befehle welche man unter der Command Prompt benutzte wurden per Default als Aliase auf die neuen Befehle (wie diese im Detail heissen, dazu später mehr) gelegt.

Ein etwas tieferer Blick in die neue Umgebung

Auch wenn man zu jenen Leuten gehört die altgewohntes lieben – so wie ich – so kann es sicherlich nicht schaden, einen Blick auf die neuen Befehl und deren Leistungsfähigkeit zu werden. Um einen Überblick über die aktuell installierten Cmdlets zu bekommen, reicht die einfache eingabe von:

```
get-command | more
```

... und schon bekommt man eine schöne pro Bildschirmseite zu scrollende Liste aller Befehle mit denen man sich in Zukunft anfreunden kann.

Auf den ersten Blick kann man bei den Befehlen schon ein schönes Muster erkennen nach welchem diese Aufgebaut sind:

```
<aktion>-<gegenstand der aktion>
```

Also beispielsweise „get-location“ um das aktuelle Verzeichnis angezeigt zu bekommen, respektive „set-location“ um das aktuelle Verzeichnis zu wechseln.

Einer der Befehle welchen ich auf meiner Reise durch die PS zu schätzen gelernt habe, ist der „get-help“, welcher als Parameter einen beliebigen anderen Befehl entgegennimmt und für diesen eine mehr oder minder kurze Beschreibung anzeigt (In der Form der UNIX Manual Pages).

Alles Objektorientiert oder was?

Beinahe alle heutzutage bekannten und auch benutzten Shells implementieren die Funktionalität des Pipelining. Darunter versteht man dass die Ausgabe eines Befehls als Eingabe für einen weiteren, durch das „|“ Symbol angehängten Befehl benutzt.

Bisher war dieses Pipelining allerdings strikt Stringorientiert – eine Ausgabe wurde 1:1 so wie diese im Normalfall am Terminal ausgegeben würde an das angepiped Programm übergeben. Mit der PS gibt es hierbei einen Paradigmenwechsel hin zur vollkommenen Objektorientierung. Die Ausgabe von Befehlen sind nämlich keine Strings mehr sondern vollwertige Objekte aus dem .NET Framework.

Am beispiel des Cmdlets „get-childitem“ kann man dies verdeutlichen. Dieses Cmdlet liefert eine Reihe von Objekten zurück, welche folgende Properties besitzen:

- Name
- CreationTime
- LastWriteTime
- LastAccessTime

Sobald der Befehl ausgeführt wird, wird dieses Objekt von der PS (oder einem angepipedten, spezifischen Formatter) aufbereitet und ausgegeben. Ein einfaches Beispiel:

```
get-childitem | format-table  
Liefert einen ‚DIR‘ ähnlichen Output
```

```
get-childitem | format-list  
Zeigt einfach nur alle Properties der Objekte an.
```

Diese Methodik ist in der ganzen PS durchgängig, weshalb insgesamt ein sehr runder Eindruck dieser neuen Art des Pipelinings entsteht.

Applikationen, Cmdlets, Funktionen und Scripts

Die PS selbst kann eine Vielzahl von verschiedenen Befehlstypen ausführen. Dazu gehören beispielsweise altbekannte Applikationen die man schon zu Zeiten des Command Promptes verwendete wie z.B. „xcopy“. Neben diesen gibt es aber auch noch einige fix eingebaute und durch den Benutzer erweiterbare, ausführbare Typen. Zu diesen zählen:

Cmdlets

Unter einem Cmdlet versteht man eine Erweiterung des Basisbefehlssatzes der PS. Cmdlets werden in einer .NET kompatiblen Sprache (C#, VB.NET, usw.) geschrieben, als Library kompiliert und anschliessend mittels eines mehrstufigen Mechanismus in der PS verankert. Nach dieser Prozedur steht das Cmdlet dem Benutzer wie ein „normaler“, integrierter Cmdlet zur Verfügung. Im übrigen sind alle Befehle die man mit „get-command - CommandType Cmdlet“ angezeigt bekommt normale Cmdlets welche sich in DLL's innerhalb des PS Installationsverzeichnis befinden

Funktionen

Eine Funktion wird im Normalfall für das abstrahieren von Programmabläufen innerhalb von Scripts verwendet, findet allerdings auch innerhalb der PS eine gewisse Bedeutung. So wird beispielsweise der von der PS angezeigte Prompt über eine Funktion mit dem Namen „prompt“ erzeugt.

Um sich eine Liste aller aktuell definierten Funktionen anzeigen zu lassen, reicht es folgenden Befehl abzusetzen „get-command -CommandType Function“.

Ein kleines Beispiel für eine simple Funktion:

```
function MyFunction
{
    (new-object Random).Next()
}
```

Ruft man nun in der PS „MyFunction“ auf, so wird eine Zufallszahl ausgegeben. Es ist zu beachten, dass Funktionen welche innerhalb der Shell zur Verfügung stehen sollen vor deren Aufruf in ein Benutzer oder Rechnerspezifisches Profil eingetragen werden müssen.

Scripte

Ein Script ist nichts weiteres als eine Datei welche eine Sammlung von Befehlen für diverse Zwecke (Eingabe, Ausgabe, Flusskontrolle, etc.) enthält und von der PS abgearbeitet wird. Die erweiterung für PS Scripte ist „PS1“. Untenstehend wieder ein kleines Beispiel:

```
function MySecondTest
{
    Write-Output( "Hello, World!" )
}

MySecondTest
```

Wenn man nun das Script ausführen will, kann es vorkommen, dass man eine Fehlermeldung bekommt, welche einen darüber informiert dass das ausführen von Nicht signiertem Code nicht aktiviert ist (Je nachdem wie man beim ersten start der PS dies konfiguriert hat).

Um das Problem zu beseitigen gibt man folgenden Befehl ein:

```
„set-executionpolicy unrestricted“
```

Ein weiterer Stolperstein über den man vielleicht fällt ist jener, dass die PS per Default keine Applikationen und Scripte im aktuellen Verzeichnis ausführt ,so wie dies die Command Shell noch tat. Wie unter Unixoiden Betriebssystemen muss man das aktuelle Verzeichniss zur ausführung mitangeben. Beispiel: (annahme dass obiges Script unter test1.ps1 gespeichert wurde)

```
„.\test1.ps1“
```

Die Scriptsprache und deren Keywords

Die Scriptsprache der PS enthält wie jede andere Programmiersprache auch eine Reihe von Keywords mit deren Hilfe komplexe Programme erstellt werden können. Diese sind:

Keywords				
process	throw	finally		
break	continue	do	else	elseif
exit	filter	for	foreach	function
	if	in	begin	return

Keywords				
switch	trap	until	end	
while				param

Wer bereits eine Hochsprache (C,C++,C#,Java,etc...) beherrscht, der wird sich leicht tun, sich diese Scriptsprache anzueignen, da viele der Konstrukte sehr ähnlich sind. Eine weitere Erläuterung derselben wird hier nicht vorgenommen, ich verweise dazu an den „PowerShell User Guide“

Die Filesystem Provider

Ein sehr nettes Feature der PS besteht darin eine beliebige, logische oder physikalische Einheit als Laufwerk benutzen zu können (sofern dafür ein entsprechendes Snapin vorhanden ist). So lassen sich per Default beispielsweise die Registry, die Umgebungsvariablen, die installierten Zertifikate und die geladenen Funktionen als Laufwerk ansprechen.

Um einen Überblick zu bekommen, welche Filesystem Provider in der PS installiert sind, reicht folgendes aus:

```
„get-psprovider“
```

Damit man die derzeit gemappten Laufwerke zu Gesicht bekommt, kann man sich mittels „get-psdrive“ eine Listung anzeigen lassen und per „set-location“ auf dieses wechseln. Beispiel:

```
„set-location Env:“
```

Es ist als kleine Draufgabe sogar möglich sich per „set-location“ direkt auf einen UNC Pfad zu verbinden:

```
„set-location \\myfileserver\myshare\mydirectory“
```

Zusammenfassung des ersten Überblicks

Die PS macht einen sehr soliden und durchdachten Eindruck, auch wenn ich mich des Gefühls nicht widersetzen kann vor einer Mischung aus BASH, C#, IronPython und der Command Shell zu sitzen.

Die Integration in .NET und die Möglichkeit Klassen derselben direkt zu instanzieren und zu benutzen (siehe Random Beispiel von oben) sucht Ihresgleichen und wird vermutlich in Zukunft eine grosse Reihe von zusätzlichen Cmdlets und anderen Providern hervorbringen.

Allerdings muss man auch sagen, dass das Gesamtkonzept einer Shell wesentlich komplexer geworden ist, als dies bei der Commandshell der Fall war und vermutlich einige User davon abhalten wird auf die PS umzusteigen. Was auch nicht weiter tragisch ist, dass die Command Shell nach wie vor die Default Shell für Windows bleiben wird.