Cmdlets4SAS: SAS Skripting und Automatisierung

Andreas Menrath
HMS Analytical Software GmbH
Rohrbacher Straße 26
69115 Heidelberg
Andreas.Menrath@analytical-software.de

Zusammenfassung

In einer immer komplexer werdenden Welt besteht zunehmend der Wunsch, Routineaufgaben weitestgehend zu automatisieren. Dieser Artikel beschreibt die Tools, mit denen sich SAS Server über die Skriptsprache PowerShell ansteuern lassen, liefert einen Überblick über die Möglichkeiten und gibt Anreize, in welchen Anwendungsfällen eine Automatisierung einen echten Mehrwert generieren kann.

Schlüsselwörter: SAS Integration Technologies, PowerShell, Skripting, Automatisierung

1 Überblick

SAS ist nicht nur eine Programmiersprache, sondern spätestens seit Version 9 auch eine vollwertige verteilte Plattform mit einer Client-/Server-Architektur.

Auf der Client-Seite sind die Anwendungsmöglichkeiten jedoch stark eingeschränkt. Hier war man bislang auf die bestehenden SAS-Clients angewiesen oder man muss sich aufwändig selbst eine Clientanwendung bauen. Wenn sich ein Problem nicht über Enterprise Guide lösen lässt, hat man hier als Kunde das Nachsehen. Leider hat SAS es versäumt ein API (application programming interface) mitzuliefern, über das dem Endanwender ein einfacher und komfortabler Zugriff auf Serverressourcen (z.B. Logfiles, Reports und Daten) ermöglicht wird.

Diese Lücke kann jedoch mit der mächtigen Skriptsprache PowerShell aus dem Hause Microsoft und einem selbst entwickelten Plugin auf elegante Weise geschlossen werden. Dieser Beitrag stellt hierzu die Grundlagen der PowerShell und dem Zugriff auf das SAS System vor. Anhand von einigen ausgewählten Beispielen wird verdeutlicht, wie durch wenige Zeilen Skriptcode bereits ein komplettes Business Szenario abgedeckt werden kann.

2 Was ist PowerShell?

2.1 Einführung

PowerShell von Microsoft ist die kostenlose Weiterentwicklung der DOS Kommandozeile. Sie wurde entwickelt, um Power-Usern und vor allem Administratoren die Arbeit zu erleichtern und die alten Skriptsprachen VBScript und Batch (CMD.EXE) durch eine moderne, Unix ähnliche Shell zu ersetzen. PowerShell ist hierbei sowohl der Name der Skriptingsprache selbst, als auch der Name der Ausführungsumgebung für PowerShell-Skripte. Ähnlich SAS können Powershell-Skripte sowohl interaktiv in einer Shell ausgeführt werden als auch im Batch-Modus ohne Benutzeroberfläche. Der Aufgabenumfang, der mit Hilfe von PowerShell bewältigt werden kann, ist riesig. Nicht nur typische Administrationsaufgaben lassen sich mit wenigen Codezeilen automatisieren, sondern es lassen sich auch andere Rechner fernadministrieren, andere Anwendungen aufrufen oder der komplette Umfang des Microsoft .Net Frameworks verwenden.

PowerShell liegt aktuell in der Version 4 vor und ist für Microsoft strategischer Bestandteil aller aktuellen Server- und Betriebssystemprodukte. Seit Windows 7 ist die PowerShell bereits im Betriebssystem vorinstalliert. Für ältere Betriebssysteme kann PowerShell kostenlos bei Microsoft heruntergeladen und nachinstalliert werden. Dieser Artikel setzt keine PowerShell Kenntnisse voraus, kann andererseits aber auch keine umfassende Einführung in die Skriptsprache gewähren. PowerShell ist eine etablierte und sehr umfangreiche Programmiersprache, zu der Einführungsliteratur, Tutorials, Blogbeiträge und Schulungsmöglichkeiten massenweise verfügbar sind.

2.2 Erste Schritte: PowerShell starten

Nach der Installation befinden sich im Startmenü unter
Programme→Zubehör→Windows PowerShell die folgenden Programme:

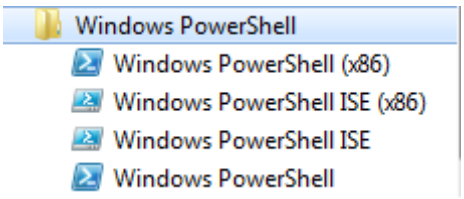


Abbildung 1: Ausschnitt Startmenü

Hinweis: die x86-Programme werden jedoch nur bei 64 Bit Betriebssystemen angezeigt. Die Anwendung **Windows PowerShell** startet ein neues Kommandozeilenfenster. Die Anwendung **Windows PowerShell ISE** startet ein neues Fenster mit einem PowerShell Skripteditor, der die Skripte auch gleich in einer Shell ausführen kann. Alle Beispiele in diesem Beitrag werden mit der ISE (steht für „Integrated Scripting Environment“) ausgeführt.

Standardmäßig ist die PowerShell so konfiguriert, dass keine Skripte ausgeführt werden dürfen. Diese Maßnahme soll den Anwender vor möglichem Missbrauch schützen. Daher muss PowerShell einmalig als Administrator gestartet werden und der folgende Code ausgeführt werden:

```
Set-ExecutionPolicy RemoteSigned
```

Bestätigen Sie die Nachfrage. Jetzt können Sie auf ihrem Rechner ihr erstes Skript schreiben und ausführen. Weitere Informationen zur Aktivierung von PowerShell finden Sie auch im Blog des „SAS Dummy“ Chris Hemedinger unter [1].

2.3 Erste Schritte: PowerShell Hello World

Starten Sie die Windows PowerShell ISE und schreiben den folgenden Code:

```
Write-Host "Hello $env:USERNAME"
```

Nach einem Klick auf die Schaltfläche mit dem grünen Pfeil oder per Tastaturbefehl F5 wird das Skript sofort ausgeführt und im unteren Konsolenfensterbereich wird die Begrüßung für Ihren angemeldeten Windows-Benutzernamen ausgegeben:

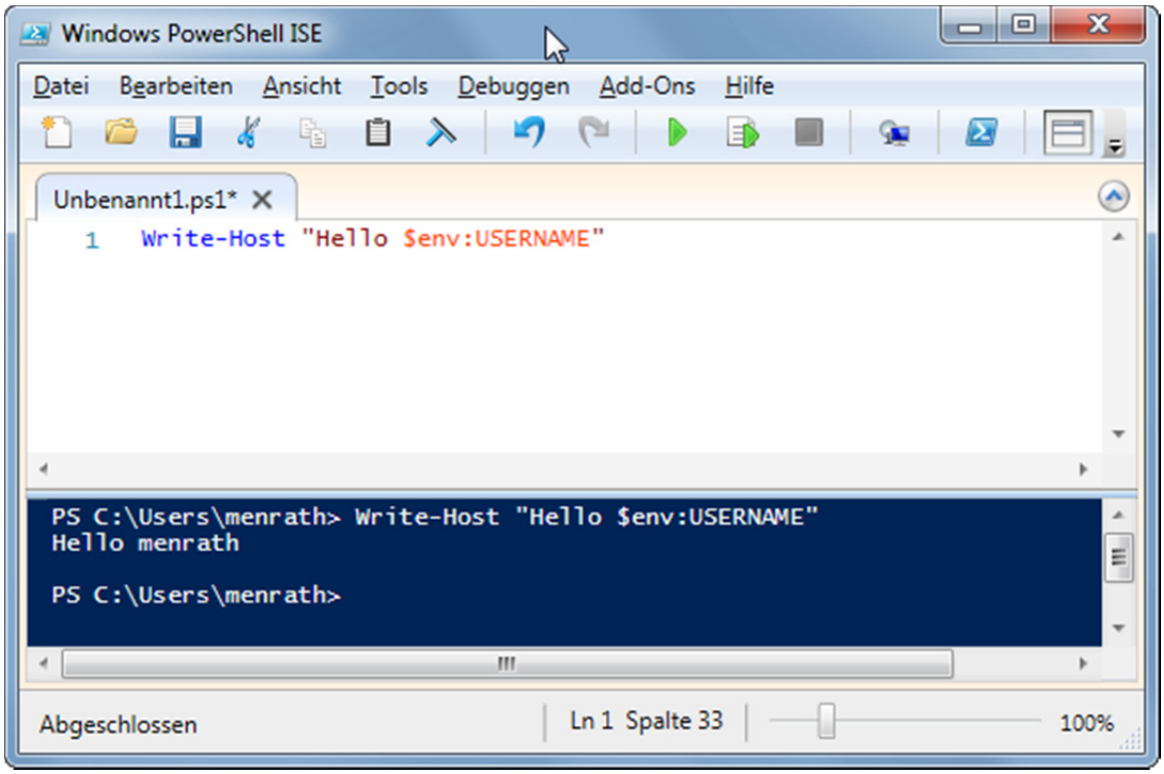


Abbildung 2: PowerShell ISE mit Skript und Konsolenfenster

2.4 Cmdlets

Mit dem Hello-World Beispiel haben Sie schon einen wesentlichen Aspekt von PowerShell kennengelernt: **Cmdlets** (Kurzform für Commandlets). Ein Commandlet ist ein vordefinierter Befehl, der bestimmte Aufgaben ausführt. Im obigen Beispielcode hat das Cmdlet „Write-Host“ bewirkt, dass ein Text auf der Kommandozeile ausgegeben

wird. Ein Commandlet folgt dabei immer dem Verb-Substantiv Namensschema [2]. Dadurch lassen sich die Befehlsnamen leichter merken und Skriptcode lässt sich leichter lesen. Die PowerShell selbst bringt bereits Hunderte dieser Commandlets mit. Darüber hinaus können noch weitere spezialisierte Cmdlets nachinstalliert werden, die z.B. die Administration von Microsoft SharePoint, Microsoft Exchange Server, Microsoft SQL Server oder virtuellen Maschinen von VMWare erlauben.

Da alle Cmdlets nach einem bestimmten Schema aufgebaut sind, lassen Sie sich nach dem Baustein-Prinzip miteinander kombinieren und somit zu sehr komplexen Gesamtanwendungen zusammenführen.

3 Cmdlets4SAS

3.1 Motivation

Leider gibt es aus dem Hause SAS Institute Inc. keine PowerShell Commandlets, mit denen sich SAS Aufgaben automatisieren ließen. SAS bietet jedoch mit dem kostenlosen Produkt **SAS Integration Technologies** die IOM-Schnittstelle[1] an, mit der SAS Server über die COM-Schnittstelle anprogrammiert werden können. Diese Schnittstelle wird auch intern von den SAS eigenen Clients, wie z.B. Enterprise Guide verwendet und ist daher auf jedem Rechner mit mindestens einem SAS Client bereits installiert.

Es besteht jedoch die Möglichkeit, selbst Commandlets für die PowerShell zu programmieren und somit fehlende Funktionalität selbst nachzurüsten. Im Blog des „SAS Dummy“[2] gibt es seit einiger Zeit auch Beispiele, wie sich SAS durch PowerShell Skripte steuern lässt. Die Einzelbeispiele sind jedoch nicht aufeinander abgestimmt, inhaltlich unvollständig und auch nicht immer zur Wiederverwendung geeignet. Darüber hinaus setzen alle Beispiele ein fortgeschrittenes Know-how der IOM-Schnittstelle und COM voraus.

Da der Autor dieses Beitrags bereits jahrelange Erfahrung mit der Verwendung der IOM-Schnittstelle hatte und mit allen Tücken und Problemen der Schnittstelle vertraut ist, war es naheliegend die wichtigsten SAS Funktionalitäten als PowerShell Commandlets nachzuprogrammieren, um über die PowerShell einen vereinfachten Zugriff auf SAS Server zu ermöglichen. Bislang sind viele Ideen des Autors für eine SAS Automatisierung bereits in der Entwurfsphase daran gescheitert, dass der Aufwand für die Lösung eines Problems zu hoch erschien, da immer gleich eine komplette Anwendung hätte erstellt werden müssen. Durch wiederverwendbare Cmdlets und eine flexible Kombinationsmöglichkeiten der Cmdlets im Skriptcode rückt dieses Hindernis jedoch in weite Ferne.

[1] IOM: Integrated Object Model
[2] http://blogs.sas.com/content/sasdummy/tag/powershell/

3.2 Vorstellung des Projekts

Unter dem Projekt- und Produktnamen Cmdlets4SAS (sprich: Commandlets for SAS) wurde ein Open Source Projekt ins Leben gerufen, das sich als Ziel gesetzt hat, den Zugriff auf SAS Serverressourcen mit einigen Cmdlets sowohl zu vereinheitlichen, als auch zu vereinfachen.
Auf der Projektseite unter http://sourceforge.net/projects/cmdlets4sas/ lässt sich sowohl der Quellcode (inkl. Unit Tests) als auch der Binärcode als ZIP Archiv kostenlos herunterladen.

Die Wikiseite des Projekts unter http://sourceforge.net/p/cmdlets4sas/wiki/Home/

- enthält die Dokumentation des Projekts und der einzelnen Commandlets,
- gibt Hilfestellungen bei der Installation von PowerShell und den Cmdlets4SAS,
- enthält einige lauffähige Beispiele und Anwendungsfälle.

Um die Cmdlets4Sas nutzen zu können, müssen einige Dateien in einen bestimmten Ordner im Dateisystem kopiert werden. Hierzu finden Sie weitere Informationen auf der Seite des Projekts oder in der mitgelieferten Datei InstallationInstructions.txt.

Nach der Installation von Cmdlets4SAS aus dem ZIP-Archiv, muss das PowerShell Modul mit den Cmdlets erst noch in der PowerShell Sitzung geladen werden. Hierzu dient der folgende Befehl:

```
Import-Module Cmdlets4Sas
```

Diese eine Zeile genügt, um alle Commandlets zu laden und in der aktuellen Sitzung verfügbar zu machen.

3.3 Steckbrief der wichtigsten Cmdlets

In der aktuellen, stabilen Version 1.0 sind bereits **24 voll funktionsfähige Cmdlets** enthalten. Jedes Cmdlet vorzustellen würde den Rahmen dieses Beitrags sprengen. Daher sollen an dieser Stelle nur die wichtigsten Funktionalitäten vorgestellt werden.

3.3.1 Verbindungsaufbau

Das wichtigste Feature ist natürlich erst einmal eine Verbindung zu einem SAS Ausführungsserver herzustellen. Aktuell werden die folgenden Servertypen unterstützt:

- SAS Workspace Server
- SAS Pooled Workspace Server
- SAS Stored Process Server

Am Einfachsten funktioniert es, einen lokalen SAS Workspace Server auf dem eigenen Rechner zu starten. Damit der Server von weiteren Commandlets verwendet werden

kann, ist es erforderlich, diesen in einer Variablen zu speichern[3]. Hierzu genügt der folgende Aufruf:

```
$ws = Connect-SasWorkspaceLocal
```

Möchte man auf einen entfernten SAS Server zugreifen, muss man zunächst ermitteln, unter welchen Verbindungsdaten der Server erreichbar ist. Diese Information lässt sich in der SAS Management Console oder einem anderen SAS Client recherchieren. Verbinden Sie sich z.B. im SAS Enterprise Guide mit einem Verbindungsprofil am Metadatenserver und öffnen über das Menü Ansicht→Serverliste.

In der Serverliste wählen Sie nun den Server aus und lassen sich die Eigenschaften anzeigen. In den Eigenschaften finden Sie wichtigen Einstellungen zu Serververbindung und Port:

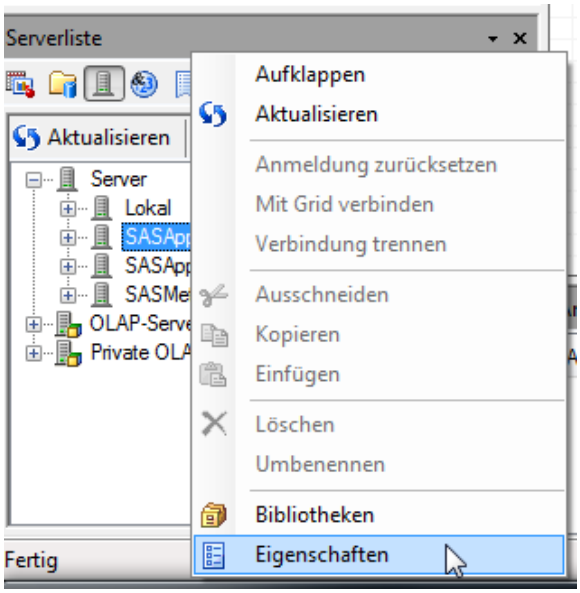


Abbildung 3: Ausschnitt Enterprise Guide

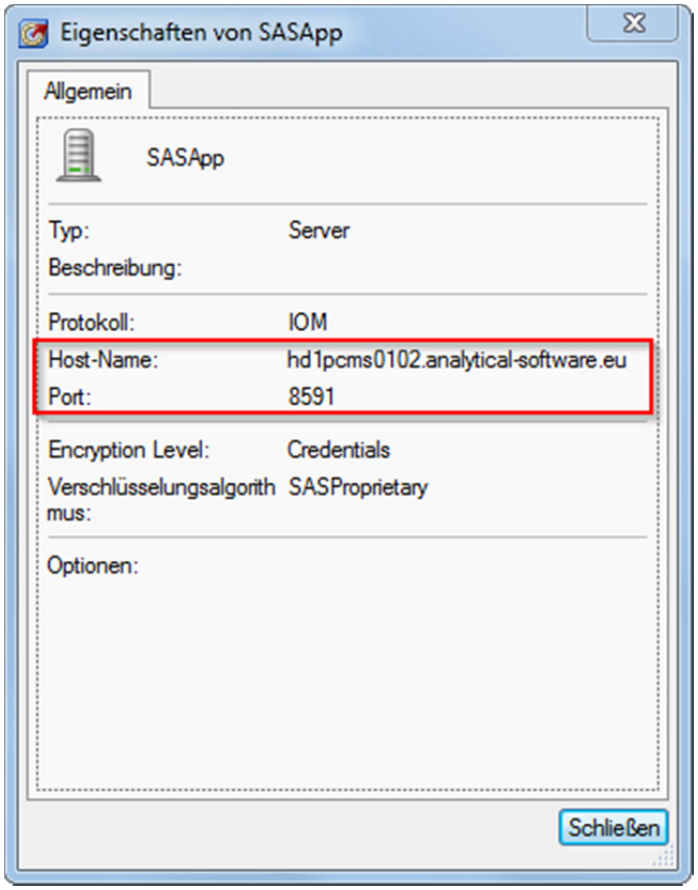


Abbildung 4: Enterprise Guide - Servereigenschaften

Aus diesen Informationen können Sie nun den Verbindungsstring in Form einer URI zusammenstellen[4]. Ein Standard URI hat den folgenden Aufbau:

```
iom://<Server.Unternehmen.com>:<Port>;Bridge
```

[3] Als Default-Variablenname wird durchgängig **$ws** als Abkürzung für Workspace verwendet.
[4] Nähere Informationen zu dem Format finden Sie unter: http://support.sas.com/documentation/cdl/en/oledbpr/63701/HTML/default/viewer.htm#p016vf6x8ord2nn1pm0dg3fjsuo1.htm

Ersetzen Sie nun den Servernamen und die Portnummer und schon können Sie eine Verbindung zu dem Server aufbauen:

```
$ws = Connect-SasWorkspaceServer "iom://hd1pcms0102.analytical-
software.eu:8591;Bridge" "MyUser" "MyPassword"
```

Dieses Beispiel enthielt eine direkte Anmeldung am Server mit einem Benutzer und Passwort. Üblicherweise möchte man seine Anmeldedaten jedoch nicht in seinem Programmcode speichern. Hierzu gibt es die Möglichkeit die Anmeldedaten zur Laufzeit über einen PowerShell Prompt abzufragen.

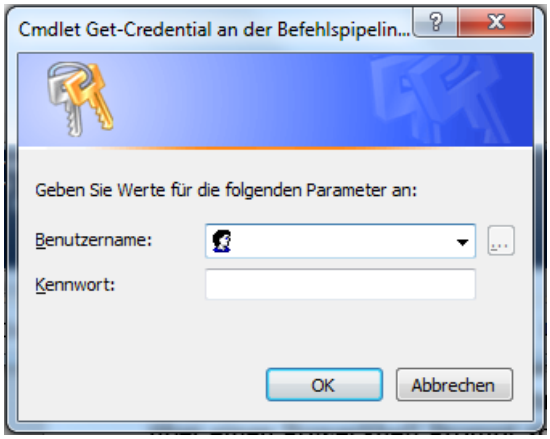


Abbildung 5: PowerShell Anmeldedialog

Das modifizierte Skript mit Anmelde-Prompt lautet nun:

```
$anmeldung = Get-Credential
$ws = Connect-SasWorkspaceServer "iom://hd1pcms0102.analytical-
software.eu:8591;Bridge" $anmeldung
```

Neben dem vorgestellten Anmeldeverfahren mit Benutzername und Passwort werden auch die Anmeldung über die **Integrierte Windows Authentifizierung (IWA)** und über **SAS Token Authentizierung**[5] unterstützt. Details können in der Cmdlets4SAS Dokumentation nachgelesen werden.

Nach einer erfolgreichen Anmeldung besteht natürlich auch die Möglichkeit, die Verbindung wieder zu trennen, um Hardware-Ressourcen zu schonen. Hierzu dient das Commandlet Disconnect-SasServer, das als einzigen Eingabeparameter eine Variable mit einem SAS Server Objekt erwartet:

```
Disconnect-SasServer $ws
```

3.3.2 SAS Code ausführen und Log prüfen

Nach einer erfolgreichen Anmeldung an einem SAS Ausführungsserver, können nun eine Vielzahl von Aktionen ausgeführt werden. Am häufigsten möchte der Anwender jedoch SAS Quellcode ausführen und anschließend das SAS Log auf Fehler überprüfen.

[5] Dieses Verfahren wird bei der Anmeldung am Pooled Workspace Server und Stored Process Server zwingend vorausgesetzt.

Zur Ausführung von SAS Code dient das Commandlet **Invoke-SasCode**, das als Eingabeparameter ein Serverobjekt und einen Text mit dem Quellcode erwartet:

```
Invoke-SasCode $ws "%put hello world;"
```

Für ein längeres Programm sollte der Code in einer PowerShell Variablen gespeichert werden. Für einen mehrzeiligen Text bietet die PowerShell einen sog. **Here-String** an, der mit **@“** beginnt und mit **“@** abgeschlossen wird. Auf diese Weise lassen sich auch elegant Zeilenumbrüche in den Text einbringen. Das angepasste Skript bekommt nun den Quellcode als Variable $code übergeben:

```
$code = @"
    data x;
      set sashelp.class;
    run;
    proc print data=x;
    run;
"@

Invoke-SasCode $ws $code
```

Ebenso leicht, wie die Codeausführung lässt sich nun auch das SAS Log mit dem Commandlet **Read-SasLog** ausgeben:

```
Read-SasLog $ws
```

Als einziger Parameter wird ein Serverobjekt erwartet. Die Standardausgabe erfolgt direkt auf die Konsole. Hier lässt sich auch die Mächtigkeit und Flexibilität der PowerShell demonstrieren: möchte man das Log permanent in einer Datei speichern, kann man die Ausgabe mit dem > Operator ganz einfach in eine Datei umleiten:

```
Read-SasLog $ws > C:\temp\my_sas.log
```

Zuletzt soll noch erwähnt werden, dass über den –AdvancedLog Schalter für das Read-SasLog Commandlet auch Loganalysen zum Kinderspiel werden. Über den Schalter wird jeder Logzeile auch ein Typ zugewiesen. Hierdurch kann man sich z.B. nur die Logzeilen ausgeben lassen, die einen Fehler oder eine Warnung enthalten. Im Wikibereich „Samples“ ist hierzu ein lauffähiges Beispiel hinterlegt[6].

3.3.3 Datenanalyse

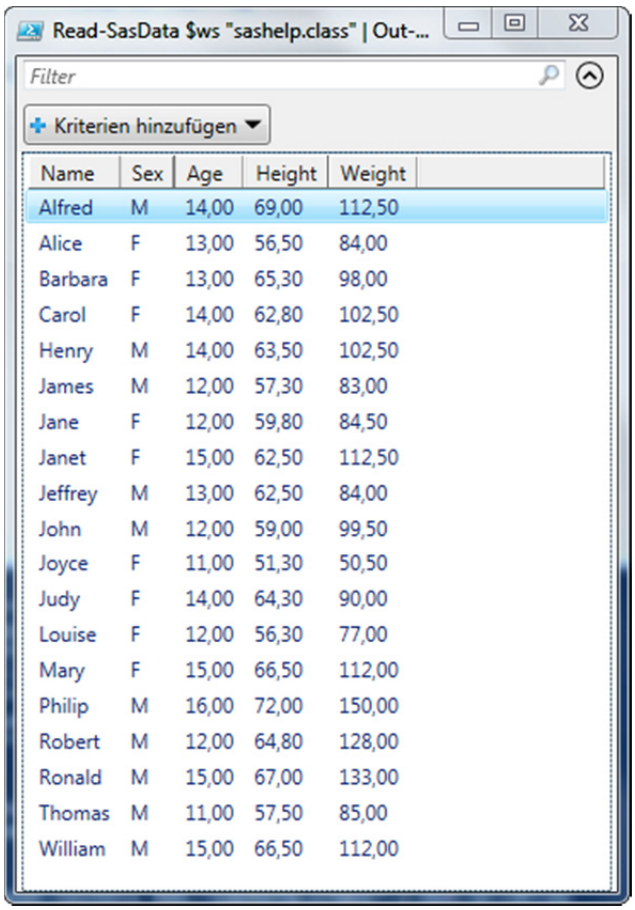
SAS besteht jedoch nicht nur aus Programmen, sondern auch aus Daten. Mit dem Commandlet **Read-SasData** lassen sich Tabellendaten anzeigen und Ad-hoc Abfragen ausführen. Zur Darstellung der Inhalte wird empfohlen das mächtige PowerShell GridView Steuerelement zu verwenden.

Mit nur einer einzigen Zeile Code lässt sich in diesem Beispiel die Tabelle SASHELP.CLASS abfragen und per Pipe-Operator an das GridView übergeben:

```
Read-SasData $ws "sashelp.class" | Out-GridView
```

[6] http://sourceforge.net/p/cmdlets4sas/wiki/Samples/

Als “Ergebnis” erscheint ein interaktives Steuerelement, in dem man dynamisch Filtern und Sortieren kann:



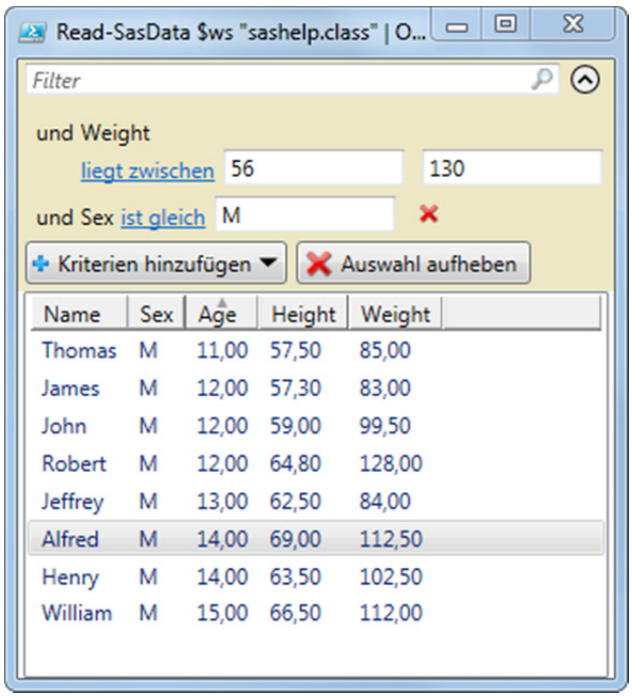


Abbildung 6: PowerShell GridView mit Filter

Abbildung 7: PowerShell GridView

Die Ausgaben enthalten standardmäßig die unformatierten Rohdaten. Wenn die hinterlegten SAS Formate auf die Spalten angewendet werden soll, so muss der Schalter –ApplyFormats gesetzt werden:

```
Read-SasData $ws "sashelp.prdsale" -ApplyFormats | Out-GridView
```

Zuletzt soll noch darauf hingewiesen werden, dass nicht nur statische Tabellen und Views unterstützt werden, sondern auch beliebig komplexe Ad-hoc Abfragen in Form einer SAS SQL Abfrage:

```
Read-SasData $ws "SELECT DISTINCT country FROM sashelp.prdsale WHERE
REGION='EAST'" | Out-GridView
```

3.3.4 Dateiübertragung

Mit den beiden Commandlets **Read-SasFile** und **Write-SasFile** lassen sich sowohl Text- wie auch Binärdateien zwischen dem eigenen Rechner und dem SAS Server austauschen. Zum Download einer Textdatei genügt etwa der folgende Aufruf:

```
Read-SasFile $ws "/usr/remote/test.log" "C:\temp\test.log"
```

Als Parameter werden hier das Serverobjekt übergeben, sowie der Quellpfad auf dem Server und der Zielpfad auf dem eigenen Rechner.

Um die Datei wieder zurück auf den Server zu speichern, genügt der folgende Aufruf:

```
Write-SasFile $ws "C:\temp\test.log" "/usr/remote/test.log"
```

Über den Schalter **–Binary** lässt sich noch definieren, dass die Daten binär übertragen werden sollen. Bei der Übertragung von sehr großen Dateien kann es vorkommen, dass der Arbeitsspeicher für die Zwischenspeicherung der Daten nicht ausreicht. Hierzu kann mit dem **–Bulk** Schalter gesteuert werden, dass die Datei in kleineren Paketen übertragen werden soll. Die Übertragung einer großen Binärdatei lässt sich wie folgt durchführen:

```
Read-SasFile $ws "/usr/huge.pdf" "C:\temp\huge.pdf" -Binary -Bulk
```

3.4 Weitere Cmdlets

Neben den vorgestellten Cmdlets existieren noch einige weitere Cmdlets z.B. zur Abfrage von SAS Makrovariablen, Abrufen der ODS LISTING Ausgabe, asynchroner Ausführung von SAS Anfragen, Abfragen gegen das Dateisystem des SAS Servers, u.v.m.

4 Anwendungsbeispiel: Versionsverwaltung

Zum Abschluss wird nun noch ein gekürztes Skriptbeispiel vorgestellt, das sämtliche SAS Autocall Makros (aus allen Ordnern in der SAS Fileref „SASAUTOS“) in einen Ordner kopiert und anschließend alle Dateien und Änderungen an das Versionsverwaltungstool SVN weiterleitet:

```
# path to SVN checked out folder
$workingPath = "D:\Daten\Commandlets4Sas\Demo_SVN\Checkout"
Import-Module Cmdlets4Sas

# start SAS
$ws = Connect-SasWorkspaceLocal

$macrofolders = Read-SasData $ws "select xpath from
dictionary.extfiles where fileref = 'SASAUTOS'" | Select-Object -
ExpandProperty "xpath"

# download all automatic SAS macros
foreach ($folder in $macrofolders)
{
  $files = Get-SasFiles $ws $folder
  foreach ($file in $files)
  {
    $targetPath = "$workingPath\$($file.Name)"
    Write-Host "Downloading $($file.FullName) to $targetPath"
    Read-SasFile $ws -From $file.FullName -To $targetPath
  }
}

# close SAS session
```

```
Disconnect-SasServer $ws

### execute multiple SVN commands to add files to source control ###
# add all files to svn
svn add "$workingPath\*"
svn commit --message "this is the initial commit"

# change some file
"my new line" | Out-File -FilePath "$workingPath\left.sas" -Append -
Encoding ascii

# commit changes
svn commit --message "this is the second commit with changes"
```

Nach der Ausführung des Skripts befinden sich alle Dateien im Zielordner unter Versionsverwaltung und auch die Historie mit den Dateiänderungen ist aktualisiert worden:

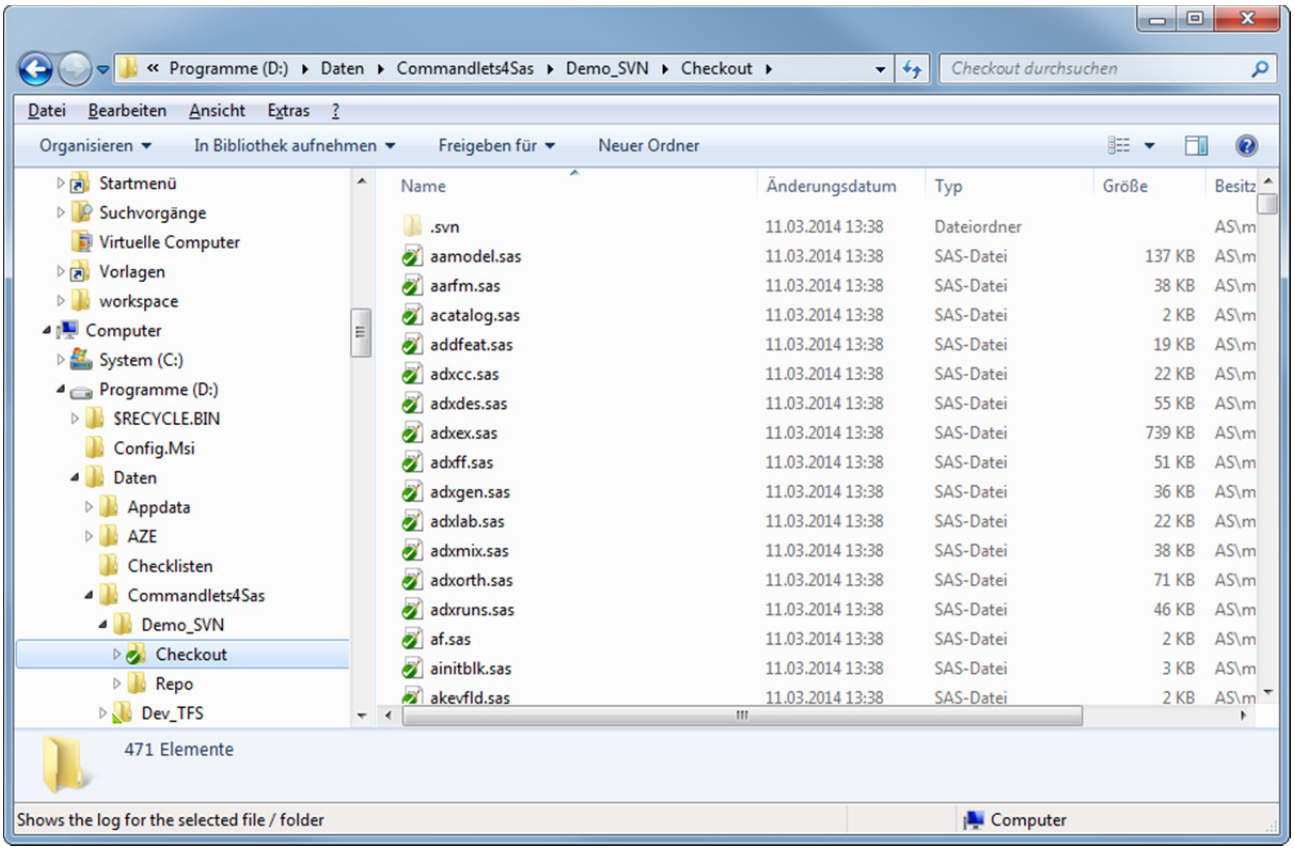


Abbildung 8: Ansicht im Dateisystem

Dieses Anwendungsbeispiel sollte verdeutlichen, wie relativ einfach und mit nur wenigen Zeilen Quellcode sich eine Vielzahl von Aufgaben automatisieren lassen. Das Beispiel soll stellvertretend für viele weitere Anwendungsszenarien dienen. So lassen sich etwa genauso gut in die Gegenrichtung alle aktuellen Makros aus einem Versionsverwaltungssystem auschecken und auf einen oder gleich mehrere SAS Server ausrollen: ein komplettes Programmeinsatzverfahren mit Rollout Prozess auf unterschiedliche Server lässt sich nun mit nur wenig Aufwand realisieren.

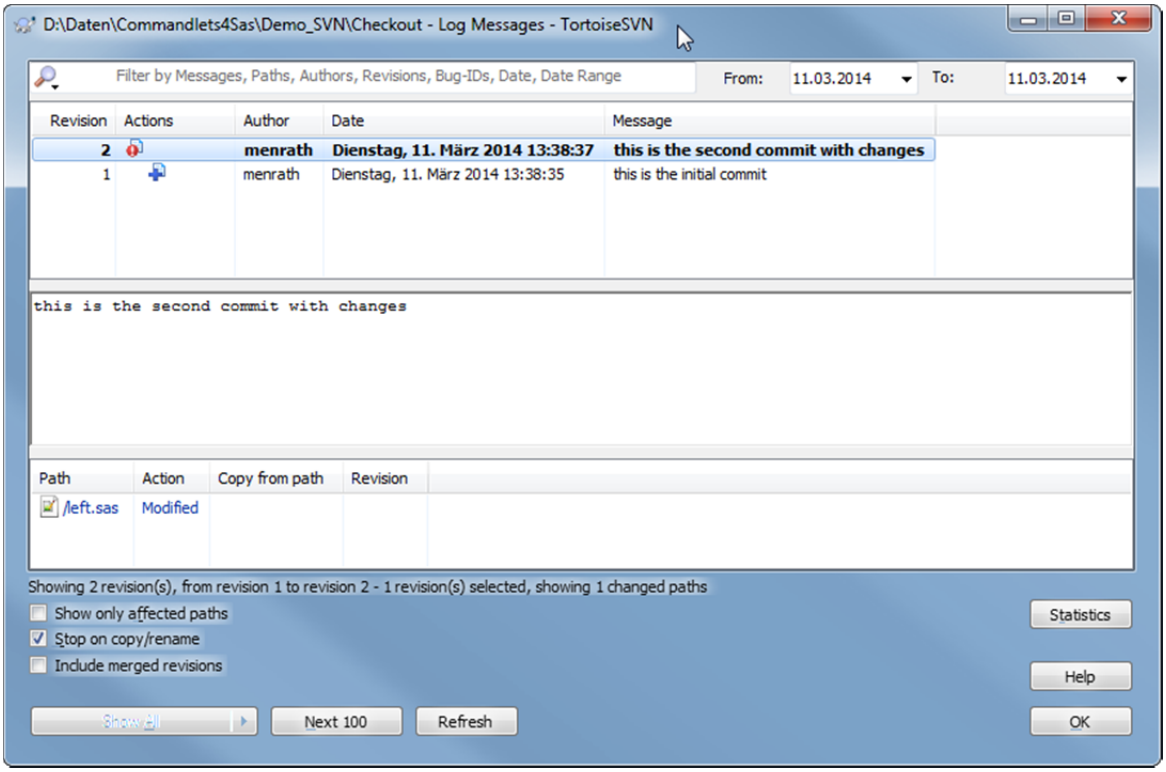


Abbildung 9: Ansicht der SVN Historie

Darüber hinaus lassen sich viele weitere Anwendungsszenarien denken. Hier einige Anreize:

- PowerShell Skripte lassen sich über einen **Scheduler** regelmäßig oder ereignisgesteuert aufrufen. Routineaufgaben lassen sich damit weitgehend automatisieren.
- **Deploymentverfahren** z.B. von SAS Formaten auf eine Vielzahl von SAS Anwendungsservern.
- **Tests von SAS Makros**: über die Cmdlets4Sas lassen sich in nur einem einzigen Skript SAS Makros sowohl auf einem SAS Workspace Server, Pooled Workspace Server als auch Stored Process Server ausführen und die Ergebnisse vergleichen.
- Unterstützung in **Migrationsszenarien**: führen Sie SAS Code in der alten Produktivumgebung (z.B. mit SAS 9.2) aus, rufen den gleichen Quellcode in der neuen Umgebung unter SAS 9.4 auf und vergleichen die Ergebnisse. Da sich die Cmdlets4Sas gegen alle SAS Versionen verbinden können, werden Regressionstests zum Kinderspiel.

5 Ausblick

PowerShell ist eine mächtige Skriptsprache mit einer Vielzahl an Schnittstellen und Anwendungsmöglichkeiten. Wird diese Skriptsprache um spezialisierte Commandlets ergänzt, können mit wenigen Zeilen Code viele Aufgaben automatisiert werden, die mit den SAS Standard-Tools nicht realisiert werden können.

Aktuell sind in der Version 1.0 von Cmdlets4Sas bereits 24 Cmdlets enthalten. Es ist jedoch absehbar, dass in Zukunft noch weitere Cmdlets hinzukommen werden, um beispielsweise die Streaming Möglichkeiten von Stored Processes zu nutzen, die Realisierung von Tests (in Form von Unit- oder Regressionstests) zu erleichtern oder um SAS Metadatenobjekte verwalten zu können.

Der Autor würde sich auch über ein Feedback freuen und möchte erfahren, in welchen Anwendungsszenarien die Cmdlets4SAS eingesetzt würden bzw. welche Prozesse in der SAS-Welt in Zukunft am ehesten automatisiert werden sollten.

Literatur

[1] SAS Dummy Blog:
http://blogs.sas.com/content/sasdummy/2011/09/12/running-windows-powershell-scripts/

[2] PowerShell Benutzerhandbuch:
http://technet.microsoft.com/de-de/library/dd315315.aspx