DDE mit Unix SAS?

Tim Schwarz
PAREXEL International
Spandauer Damm 130
14050 Berlin
tim.schwarz@parexel.com

Zusammenfassung

Das Finalisieren von SAS Outputs mit MS Office Produkten gehört im CRO Betrieb zur täglichen Arbeit. Je nach Grad der Automatisierung kann dieser Arbeitsschritt sogar einer der aufwendigsten sein. Beim PC SAS System bekommt der Nutzer mit dem DDE Protokoll ein elegantes Werkzeug an die Hand, um diesen Prozess aus SAS heraus zu steuern. Was aber tun bei einem Umstieg von PC auf Unix SAS, wo dieses Werkzeug nicht mehr verfügbar ist? Dieser Beitrag soll Möglichkeiten aufzeigen, diese Schnittstelle im Rahmen von Unix SAS, Samba und dem Windows Scheduler zu ersetzen, um eine annähernd gleichwertige Effizienz wie mit dem PC SAS System zu erreichen.

Schlüsselwörter: DDE, Unix, Automatisierung, SAMBA

1 Problemstellung

Das Finalisieren von SAS Outputs mit MS Office, Acrobat oder anderen Produkten gehört im CRO Betrieb (Auftragsforschung) zur täglichen Arbeit. Je nach Grad der Automatisierung kann dieser Arbeitsschritt sogar einer der aufwendigsten sein. Beim PC SAS System bekommt der Nutzer mit dem DDE (dynamic data exchange) Protokoll ein elegantes Werkzeug an die Hand, um diesen Prozess aus SAS heraus zu steuern.

Bei der Nutzung von Unix SAS steht diese Schnittstelle nicht zur Verfügung. Die Finalisierung der Dokumente erfolgt aber i.d.R. weiterhin unter Windows, so dass hier ein wichtiger Automatisierungsschritt und damit Effizienz verloren geht.

Ziel dieses Beitrages ist es, einen alternativen Ansatz zu entwickeln, welcher kostengünstig und für den Endanwender verhältnismäßig einfach zu implementieren ist.

2 Lösungsansatz

Nachfolgend wird mit der Verzeichnisüberwachung eine einfach zu implementierende Alternative zur DDE Schnittstelle beschrieben. Dies wird gezeigt im Rahmen von Samba als Basis, dem Windows Task Scheduler als Hilfswerkzeug und Windows Skriptsprachen zur Überwachung und Automatisierung.

2.1 Samba

Samba ist eine weit verbreitete Software unter der GNU General Public License [http://de.wikipedia.org/wiki/Samba_%28Software%29]. Das Programm läuft unter Unix und emuliert Datei- und Druckdienste von Microsoft Windows und erlaubt somit einen Zugriff auf das Unix Dateisystem aus Windows heraus. Dadurch fungiert das Dateisystem als Alternativschnittstelle zwischen den beiden Plattformen. Natürlich ist es keine direkte Schnittstelle zwischen verschiedenen Anwendungen (wie beim DDE Protokoll), sondern eher eine indirekte über den einzelnen Ordner.

Das Vorhandensein eines gemeinsam nutzbaren Ordners bewirkt an sich natürlich überhaupt nichts. D.h. diese indirekte Schnittstelle muss nun mit Funktionalität versehen werden. Eine naheliegende Idee an dieser Stelle ist, eine Verzeichnisüberwachung zu implementieren.

2.2 Windows Task Scheduler

Für die Aufgabe der Verzeichnisüberwachung könnte man eine Applikation zu eben diesem Zweck schreiben oder erwerben. Aber genau jene Zeit / Kostengründe wären wahrscheinlich für die meisten SAS Programmierer ein Argument, diesen Ansatz zu verwerfen.

Glücklicherweise nimmt an dieser Stelle der Windows Task Scheduler einen entscheidenden Teil der Arbeit ab. Nachfolgend ein paar Punkte, welche dieses Werkzeug für die Verzeichnisüberwachung als geeignet erscheinen lassen:

- standardmäßig auf jedem Windows Rechner installiert,
- läuft im Hintergrund ohne viele System Resourcen zu benötigen,
- Prozesse lassen sich zyklisch automatisiert starten (parallel/in Reihe),
- flexible Abbruchbedingungen,
- Profile lassen sich als XML Datei speichern und so leicht weitergeben.

Das heißt, das Hauptproblem des zyklischen Startens eines Prozesses ist mit dieser Anwendung bereits implementiert. Was nun noch fehlt ist der zu startende Prozess: ein Programm, welches ein Verzeichnis ausliest und ein darin befindliches Skript (welches mit Unix SAS erstellt wurde) ausführt.

Das macht das Problem schon sehr viel übersichtlicher. Zudem lässt sich dies mit verschiedensten Skriptsprachen bewerkstelligen. Zum Beispiel Java Skript, Visual Basic Skript, Power Shell, etc.

2.3 Verzeichnisüberwachung mit VBS

Ein Lösungsvorschlag wird nachfolgend mit VB Skript dargestellt.

```
'--- intialize variables and objects ---
Dim objFSO, objFolder, objFile, colFiles, strFolder, rc, timeStamp
Dim dtmCur, dtmOldestDate, strOldestFile, strOldestFileName
Dim objShell
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objShell = CreateObject("wscript.shell")

'--- generate timestamp (YYYYMMDDtHHMMSS) ---
dtmCur = Now
timeStamp = Year(dtmCur) & Right("0" & Month(dtmCur), 2) & _
Right("0" & Day(dtmCur), 2) & "t" & Right("0" & Hour(dtmCur), 2) & _
Right("0" & Minute(dtmCur), 2) & Right("0" & Second(dtmCur), 2)

'--- set monitoring folder ---
strFolder = "\\kennet\hlrag110604\stats\global\uxwin_interface"

Set objFolder = objFSO.GetFolder(strFolder)
Set colFiles = objFolder.Files

'--- check for first (oldest) file in monitoring folder ---
dtmOldestDate = Now
For Each objFile In colFiles
  If objFile.DateCreated < dtmOldestDate Then
    dtmOldestDate = objFile.DateCreated
    strOldestFile = objFile.Path
    strOldestFileName = objFile.Name
  End If
Next

'--- run script ---
If strOldestFile <> "" Then
  rc = objShell.Run(strOldestFile, 1, true)
  '-- move processed file to temporary folder and add time stamp --
  objFSO.MoveFile strOldestFile, strFolder & "\_processed_\" & _
  timeStamp & "_" & strOldestFileName
End If

'--- cleanup ---
Set objShell = Nothing
Set objFSO = Nothing
```

Die Funktionalität des obigen Skriptes wurde im letzten Abschnitt bereits angedeutet.

Hier noch einmal im Detail:

1. auslesen des zu überwachenden Verzeichnisses,
2. selektieren der ältesten Datei (die Erste in der Warteschlange),

3. ausführen der selektierten Datei,
4. mit Zeitstempel versehen und in „bearbeitet“ Ordner verschieben.

Diese 30 Zeilen Code stellen die Grundfunktionalität dar. Sie lassen sich natürlich beliebig erweitern. Zum Beispiel mit adäquater Fehlerbehandlung, ein Logfile ließe sich erstellen oder auch komplexere Protokolle implementieren. Mit diesen beiden Komponenten – dem Skript und dem Scheduler, welcher periodisch jenes kleine Skript ausführt – haben wir mit einfachen Mitteln eine funktionierende Verzeichnisüberwachung implementiert.

2.4 Windows Skript mit Unix SAS

Nun kann damit begonnen werden mit Unix SAS Skripte zu erstellen, welche von der in Abschnitt 2.2/2.3 eingeführten Verzeichnisüberwachung windowsseitig automatisiert ausgeführt werden. Diese Skripte müssen wiederum in einer Skriptsprache verfasst werden, welche die Windows Shell versteht (BAT, JS, VBS, WSH, PS, etc.).

Nachfolgend einige Beispielprogramme:

1. Kopieren von Dateien vom Unix Server auf ein lokales Laufwerk

```
FILENAME WINFILE "/proj123456/uxwin_interface/copytlf.bat";
DATA _NULL;
  FILE WINFILE;
  PUT "REM copy tables from Unix folder to local drive" /
      "copy \\proj123456\tables\*.rtf ""c:\#123456\tables 1.0\""";
RUN;
```

2. Starten von MS Word und darin ein VBA Macro mit Parameter Übergabe

```
FILENAME WINFILE "/proj123456/uxwin_interface/startBatchRun.vbs";
DATA _NULL_;
  FILE WINFILE;
  PUT "'--- initialize ---" /
      "Dim wrdApp" /
      "Set wrdApp = CreateObject(""Word.Application"")" /

      "'--- show Word GUI, otherwise Word runs in background ---" /
      "wrdApp.visible = True" /

      "'--- pass parameters with vbs to MS Word ---" /
      "wrdApp.Run ""BatchRun"", ""C:\test\"", False" /

      "'--- cleanup ---" /
      "wrdApp.Quit" /
      "Set wrdApp = Nothing";
RUN;
```

3. Starten von MS Excel, Add-Ins einladen und ein VBA Macro ausführen

```
FILENAME WINFILE "/proj123456/uxwin_interface/startAutoFormat.vbs";
DATA _NULL_;
  FILE WINFILE;
  PUT "'--- initialize ---" /
      "Dim exlApp" /
      "Set exlApp = CreateObject(""Excel.Application"")" /
      "Dim CurrAddin" /

      "'--- show GUI, otherwise Excel runs in background ---" /
      "exlApp.visible = true" /

      "'--- force Add-ins to be loaded automatically ---" /
      "For Each CurrAddin In exlApp.AddIns" /
      "    If CurrAddin.Installed Then" /
      "       CurrAddin.Installed = False" /
      "       CurrAddin.Installed = True" /
      "    End If" /
      "Next" /

      "'--- start macro AutoFormat ---" /
      "exlApp.Run ""AutoFormat"" "/

      "'--- cleanup ---" /
      "exlApp.Quit" /
      "Set exlApp = Nothing";
RUN;
```

Da die Skriptsprachen unter Windows sehr mächtige Werkzeuge sind, sind die Möglichkeiten, welche dieser Ansatz bietet, sehr weitreichend und gehen zumindest in Teilen über die DDE Funktionalität noch hinaus.

3 Zusammenfassung / Fazit

Nachfolgend ist der gesamte Prozess noch einmal schematisch dargestellt:

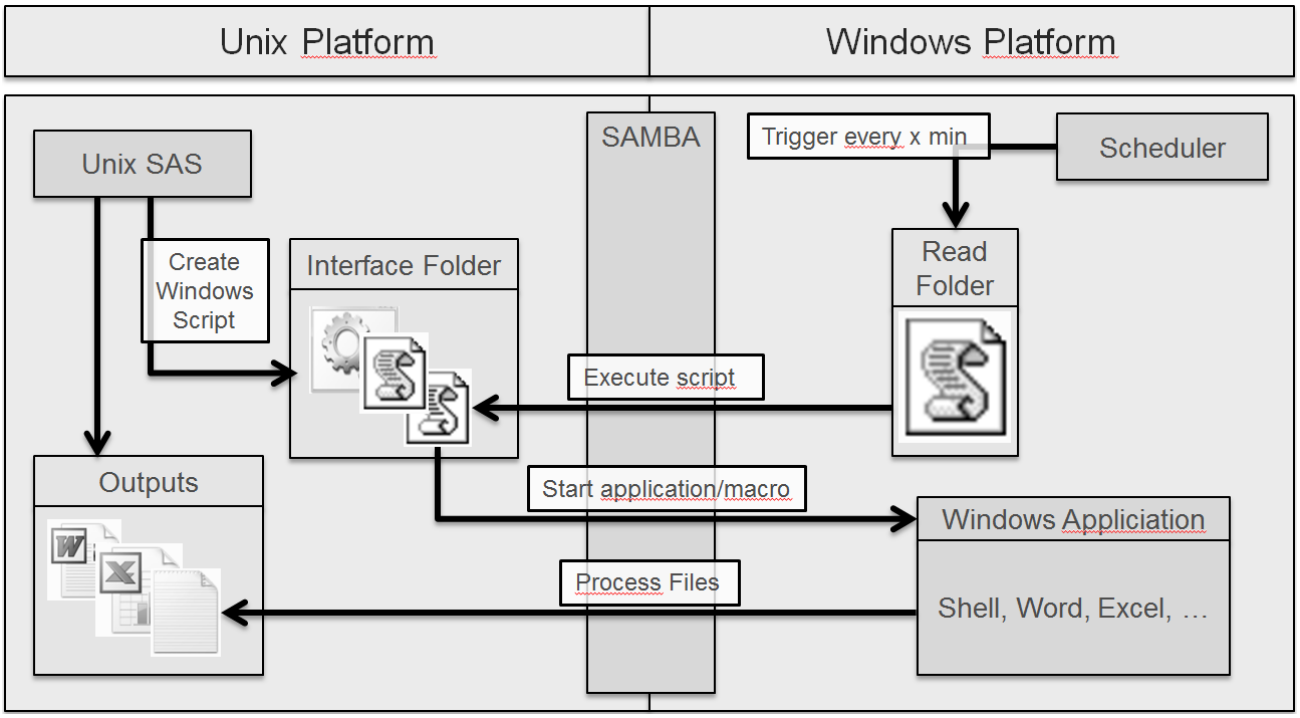

Abbildung 1: Flussdiagramm

Somit ist Samba (o.ä.) die Grundlage, mit welcher das Unix Datei System aus Windows heraus zugänglich gemacht wird. Man richtet sich einmalig den Windows Scheduler ein, welcher dann zyklisch das Skript zum Auslesen des Schnittstellen Verzeichnisses startet (siehe Abschnitt 2.3). Danach kann man dann wie in Abschnitt 2.4 beschrieben mit Unix SAS (unter Windows ausführbare) Skripte in diesem Ordner ablegen, welche dann automatisiert ausgeführt werden.

Das einzige kleine Hindernis für den SAS Programmierer bleibt lediglich, dass er ein wenig über den Tellerrand hinaus schauen und sich mit einer Skriptsprache seiner Wahl beschäftigen muss. Was sich aber in Zeiten von Google und unzähligen Foren als überschaubare Herausforderung darstellt.

Zusammenfassend lässt sich konstatieren, dass auch aus Unix SAS, und ohne die Möglichkeit der direkten Kommunikation zwischen Anwendungen, der gesamte Produktionsprozess steuerbar bleibt.