

Technical Documentation

nov 2018

*Number of pages : 3*

*Version :*

# Document Information

## Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Status | Prepared by | Comments |
| 2018-11-14 | 1.0 | Draft | WFA | Initial Draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Document Control

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Name | E-mail | Telephone |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Approval

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Name | Signature | Sign-off Date |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Table of Contents

Document Information 2

Revision History 2

Document Control 2

Approval 2

Table of Contents 3

# Introduction

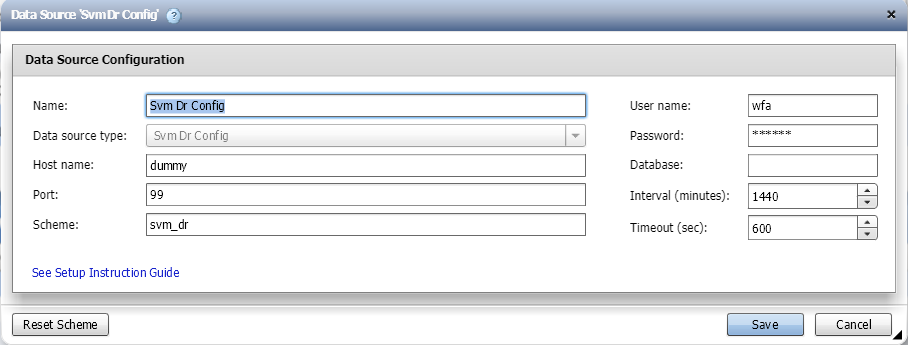
## Purpose

This document defines the datasource details, code for acquisition and scheme details.

## Description

To provide a flexible configuration, we’ve added a wfa.ini file in the svmtool directory.

Using this datasource, we read the ini-file (for example once a day) and import all configuration settings.



Note : non of the parameters (but the name) are important for this datasource, just use a dummy hostname and port. No credentials required. Interval = 1440 minutes (=1 day)

## Assumptions

For individuals reading this document please note that the following has been assumed:

* A familiarity with NetApp products and technologies
* A familiarity with PowerShell and/or Perl languages

# Scheme details

This datasource updates the scheme: svm\_dr .

Below are the tables used by the scheme.

|  |  |
| --- | --- |
| Name | Description |
| settings |  |

## Table: settings

### Fields

|  |  |  |
| --- | --- | --- |
| Name | Description | Notes |
| name | name of setting | Natural key. |
| value | value of the setting | Can be NULL. |
| type | type of the setting | Can be NULL. |

# Acquisition script

Below is the acquisition script for this datasource

#!/powershell  
#  
<# WFA Data Source  
   
 DESCRIPTION:  
 import ini-file settings  
   
 AUTHOR:  
 Mirko Van Colen (mirko@netapp.com)  
   
#>  
   
   
# ================================================================  
# == Scheme name : you must edit this  
# ================================================================  
   
$schemeName = "svm\_dr" # the scheme you want to 'datasource'  
   
# ================================================================  
# == Variables you may want to edit  
# ================================================================  
   
$interactive = $false # set this if your are running locally in the shell  
$sendmail = $false # mail the logfile & csv files  
$debug=$false # show debug info / note interactive enables this by default  
   
# IF sendmail is $true you should set the following mail variables, otherwise can be ignored  
   
$mailTo = "from@netapp.com" # who you want to send acquire debug emails to  
$mailFrom = "to@netapp.com" # who the email will appear to come from  
$mailSmtp = "mail.netapp.com" # a reachable and working smtp mail server  
   
# ================================================================  
# == Get Environment variables (script location, host & port)  
# == Note : set the datasource port to 1, to trigger debug mode  
# ================================================================  
   
if(!$interactive){  
 Set-Variable -Name SCRIPT\_PATH -Value (Split-Path (Resolve-Path $myInvocation.MyCommand.Path)) -Scope local  
}else{  
 # auto load wfa profile  
 $profilecheck = Get-Item function: | ?{$\_.Name -eq "Connect-WfaCluster"}  
 if(-not $profilecheck){  
 Write-Host "Loading WFA profile..." -ForegroundColor Yellow  
 cd 'C:\Program Files\NetApp\WFA\PoSH\'  
 . '.\profile.ps1'  
 }  
 Set-Variable -Name SCRIPT\_PATH -Value 'C:\Program Files\NetApp\WFA\jboss\standalone\tmp\wfa' -Scope local  
 cd "$SRIPT\_PATH"  
}  
  
if(!$interactive){  
 $port = Get-WfaRestParameter "port"  
 $hostname = Get-WfaRestParameter "host"  
}else{  
 $port = 1 # port number 1 enables debugging  
 $hostname = "your\_custom\_hostname\_if\_interactive"  
}  
$IniFile = $SCRIPT\_PATH + "\..\..\..\..\PoSH\Modules\svmtool\wfa.ini"  
$dllLocation = $SCRIPT\_PATH + "\..\..\..\..\Posh\Modules\DataOntap\log4net.dll"  
$logfile = $SCRIPT\_PATH + "\..\..\log.war\jboss\" + $schemeName + ".log"  
if($port -eq 1){  
 $debug=$true  
}  
   
# ================================================================  
# == Prep logging - a log file is created by default, with your scheme name  
# ================================================================  
   
# Initialize log4net  
[void][Reflection.Assembly]::LoadFrom($dllLocation)  
$pattern="%d %w %-5p %c : %m%n"  
[log4net.LogManager]::ResetConfiguration()  
New-Item -Path $logFile -type file -ErrorAction SilentlyContinue  
$Appender = new-object log4net.Appender.FileAppender  
$Appender.File = $logFile  
$Appender.Layout = new-object log4net.Layout.PatternLayout($pattern)  
if($debug){  
 $Appender.Threshold = [log4net.Core.Level]::Debug  
}else{  
 $Appender.Threshold = [log4net.Core.Level]::Info  
}  
$Appender.ActivateOptions()  
[log4net.Config.BasicConfigurator]::Configure($Appender)  
$logg = [log4net.LogManager]::GetLogger("[$schemeName]")  
#  
   
# ================================================================  
# == 5 Public logging functions - Debug,Info,Warning,Error & Fatal  
# == If you set interacte true, you'll get host-feedback  
# ================================================================  
   
# LOG INFO  
function LogInfo($t){  
 if($interactive){  
 Write-Host $t -ForegroundColor Yellow  
 }  
 $logg.Info($t)  
}  
   
# LOG DEBUG  
function LogDebug($t){  
 if($interactive){  
 Write-Host $t -ForegroundColor Cyan  
 }  
 $logg.Debug($t)  
}  
   
# LOG WARN  
function LogWarn($t){  
 if($interactive){  
 Write-Warning $t  
 }  
 $logg.Warn($t)  
}  
   
# LOG ERROR  
function LogError($t){  
 if($interactive){  
 Write-Host $t -ForegroundColor Red  
 }  
 $logg.Error($t)  
}  
   
# LOG FATAL (throws error & exit)  
function LogFatal($t){  
 if($interactive){  
 Write-Host $t -ForegroundColor Magenta  
 }  
 $logg.Fatal($t)  
 throw $t  
}  
   
# ================================================================  
# == Private - WFA Datasource conversion helper functions  
# ================================================================  
   
# converts a psobject to wfa csv  
function ConvertTo-WfaCsv($psobj,$csvpath){  
 try {  
 New-Item -Path $csvpath -type file -force | Out-Null  
 } catch [System.Exception] {  
 $msg = "Data Source: Could not create output file path: $($\_.Exception)"  
 LogFatal($msg)  
 }  
 if($psobj){  
 $csv = $psobj | convertto-csv -NoTypeInformation -Delimiter "`t"  
 $csv = $csv | %{$\_ -replace '"'} | select -skip 1  
 $Utf8NoBomEncoding = New-Object System.Text.UTF8Encoding $False  
 $csv = $csv | %{[System.IO.File]::AppendAllText((resolve-path $csvpath), "$\_`n",$Utf8NoBomEncoding)}  
 }  
}  
   
# booleanstring to int  
function BoolToInt($b){  
 if($b -match "true|yes"){  
 return 1  
 }else{  
 return 0  
 }  
}  
   
# check if the scheme is ok  
function CheckScheme($schemeName){  
 # lets connect to mysql and check the scheme first  
 try{  
 LogInfo("Looking up the scheme '$schemeName' in MySql")  
 $rsTables = Invoke-MySqlQuery "SHOW tables FROM $schemeName"  
 }catch{  
 LogError("Could not find scheme $schemeName")  
 LogError("Maybe reset the scheme $schemeName ?")  
 LogFatal("Failed to verify the scheme $schemeName, Does it exist?")  
 }  
 [System.Collections.ArrayList]$tableListMySql = @()  
   
 # are there any tables in the scheme ?  
 if($rsTables[0] -gt 0){  
 $rsTables | select -Skip 1 | %{$tableListMySql+=$\_[0]}  
 LogDebug("Found $($rsTables[0]) tables in MySql")  
 $tableListMySql | %{$i=0}{$i++;LogDebug("($i) $\_");}  
 }else{  
 LogFatal("No tables found in this scheme")  
 }  
 LogInfo("Scheme $schemeName is ok")  
 return $tableListMySql  
}  
   
# ================================================================  
# == Public - Master Conversion Function  
# ================================================================  
   
# check if a dictionary is matching your psobject, and reformat if needed for WFA  
function CheckDictionary($Dictionary,$Table){  
   
 LogInfo("Processing dictionary $Dictionary")  
   
 $fieldList = @()  
   
 # get the table fields  
 LogDebug("Looking up the fields for table $schemeName.$Dictionary")  
 $rsFields = Invoke-MySqlQuery "SHOW fields FROM $schemeName.$Dictionary"  
   
 # are there any fields in the table ?  
 if($rsFields[0] -gt 0){  
 $fields = $rsFields |select -Skip 1  
 $fieldList = $fields | %{$\_.Field}  
 LogDebug("Found $($rsFields[0]) fields")  
 $fields | %{$i=0}{$i++;LogDebug("($i) $($\_.Field) ($($\_.Type))");}  
 }else{  
 LogFatal("No fields found in table $schemeName.$Dictionary")  
 }  
 $selectOrderedId = @()  
 # make a correct ordered select  
 foreach($f in $fields){  
 if($f.Type -eq 'TinyInt(1)'){  
 $exp = [scriptblock]::Create('BoolToInt($\_."' + $f.Field + '")')  
 }else{  
 $exp = [scriptblock]::Create('$\_."' + $f.Field + '"')  
 }  
 $selectOrderedId += @{name=$f.Field;e=$exp}  
 }  
 $selectOrderedNoId = @()  
 # make a correct ordered select  
 foreach($f in $fields){  
 if($f.Field -eq "id"){  
 $exp = [scriptblock]::Create("`"\N`"")  
 }else{  
 if($f.Type -eq 'TinyInt(1)'){  
 $exp = [scriptblock]::Create('BoolToInt($\_."' + $f.Field + '")')  
 }else{  
 $exp = [scriptblock]::Create('$\_."' + $f.Field + '"')  
 }  
 }  
 $selectOrderedNoId += @{name=$f.Field;e=$exp}  
 }  
 $outFile = ".\$Dictionary.csv" # write results to intermediate CSV file  
 $global:attachList += $outFile # add CSV file to list of debug attachments  
   
 # check fields  
 if($Table){  
 LogDebug("Comparing fields with MySql table fields")  
 $tableFields = ($Table | gm -MemberType NoteProperty | select Name | %{$\_.Name})  
 if($tableFields -and $fieldList){  
 $comp = Compare-Object -ReferenceObject $fieldList -DifferenceObject $tableFields  
 if($comp){  
 $logg.Warn("Possible mismatch found between fields (if missing id, we'll add NULL-values)")  
 $comp | ?{($\_.SideIndicator -eq '<=') -and ($\_.InputObject -ne 'id')} | %{$i=0}{$i++;LogFatal("($i) Missing field '$($\_.InputObject)' in $dictionary")}  
 $comp | ?{($\_.SideIndicator -eq '=>')} | %{$i=0}{$i++;$logg.Warn("($i) Extra field '$($\_.InputObject)' in $dictionary")}  
 if($comp | ?{($\_.SideIndicator -eq '<=') -and ($\_.InputObject -ne 'id')}){  
 LogFatal("Mismatch found between fields. Check the log in the log-viewer for more info.")  
 }  
 }else{  
 LogDebug("All fields match")  
 }  
 }else{  
 LogFatal("There are no fields in table $Dictionary")  
 }  
   
 # we check if it has a primary key ("id")  
 $haspk = $Table | gm -Name "id"  
 if($haspk){  
 $table = @($Table | select -Property $selectOrderedId)  
 LogDebug("We found an id, starting to hash it")  
 # if it does, we'll hash it  
 for($i=0;$i -lt $table.Count;$i++){  
 $table[$i].id = $table[$i].id.GetHashCode()  
 }  
 }else{  
 # if not, we add \N  
 LogDebug("No id found, we'll add one with \N")  
 $table = @($Table | select -Property $selectOrderedNoId)  
 }  
   
 # we check if it has one or more primary keys  
 $hasfk = $table | gm -Name "\*\_id"  
 if($hasfk){  
 LogDebug("1 or more foreign keys found, starting to hash")  
 # if it does, loop them and hash them  
 $fkcount = 0  
 foreach($fk in $hasfk){  
 $fkcount++  
 LogDebug("($fkcount) fk = $($fk.Name)")  
 for($i=0;$i -lt $table.Count;$i++){  
 $table[$i]."$($fk.Name)" = ($table[$i]."$($fk.Name)").GetHashCode()  
 }  
 }  
 }  
 }else{  
 LogWarn("This table is empty")  
 }  
 # convert to WFa readable csv  
 LogDebug("Start converting the table to csv '$outFile'")  
 ConvertTo-WfaCsv -psobj $table -csvpath $outFile  
   
}  
   
   
# Ensure that dates are always returned in English  
[System.Threading.Thread]::CurrentThread.CurrentCulture="en-US"  
   
# Start & check scheme  
LogInfo("===============================================================")  
LogInfo("BEGINNING Acquisition (set portnumber to '1' for debug logging)")  
LogInfo("===============================================================")  
   
$global:attachList = @()  
$global:attachList += ".\datasource.log" # make the first debug-email attachement be the log-file itself  
   
[System.Collections.ArrayList]$tableListMySql = @(CheckScheme -SchemeName $schemeName)  
   
# ==============================================================================================  
# ======================================== START CUSTOM CODE ===================================  
# ==============================================================================================  
   
 # Ini Content Helpers  
Function GetIniContent {   
 [CmdletBinding()]   
 Param(   
 [ValidateNotNullOrEmpty()]   
 [ValidateScript({(Test-Path $\_) -and ((Get-Item $\_).Extension -eq ".ini")})]   
 [Parameter(ValueFromPipeline=$True,Mandatory=$True)]   
 [string]$FilePath   
 )   
   
 Begin   
 {Write-Verbose "$($MyInvocation.MyCommand.Name):: Function started"}   
   
 Process   
 {   
 Write-Verbose "$($MyInvocation.MyCommand.Name):: Processing file: $Filepath"   
   
 $ini = @{}   
 switch -regex -file $FilePath   
 {   
 "^\[(.+)\]$" # Section   
 {   
 $section = $matches[1]   
 $ini[$section] = @{}   
 $CommentCount = 0   
 }   
 "^(;.\*)$" # Comment   
 {   
 if (!($section))   
 {   
 $section = "No-Section"   
 $ini[$section] = @{}   
 }   
 $value = $matches[1]   
 $CommentCount = $CommentCount + 1   
 $name = "Comment" + $CommentCount   
 $ini[$section][$name] = $value   
 }   
 "(.+?)\s\*=\s\*(.\*)" # Key   
 {   
 if (!($section))   
 {   
 $section = "No-Section"   
 $ini[$section] = @{}   
 }   
 $name,$value = $matches[1..2]   
 $ini[$section][$name] = $value   
 }   
 }   
 Write-Verbose "$($MyInvocation.MyCommand.Name):: Finished Processing file: $FilePath"   
 Return $ini   
 }   
   
 End   
 {Write-Verbose "$($MyInvocation.MyCommand.Name):: Function ended"}   
}  
   
$ini = GetIniContent -FilePath $IniFile  
$settings = @()  
foreach($i in $ini.Keys){  
 foreach($s in $ini[$i].Keys){  
 $o = "" | select name,value,type  
 $o.name = $s  
 $o.value = $ini[$i][$s]  
 $o.type = $i  
 $settings += $o  
 }  
}  
   
CheckDictionary -Dictionary "settings" -Table $settings  
   
# ==============================================================================================  
# ======================================== END CUSTOM CODE ===================================  
# ==============================================================================================  
   
# close and send mail if needed  
LogInfo("Acquisitions COMPLETE")  
if ($sendmail)  
{  
 Copy-Item $logFile ".\datasource.log"  
 $bodyText = "See attachments.`n`nNOTE: Scroll to the end of the attached datasource.log file to see the log of this execution of the $schemeName Data Source.`n`n"  
 Send-MailMessage -to $mailTo -from $mailFrom -subj "WFA $schemeName Data-Source: Log and CSV Attachments" -body $bodyText -attachments $global:attachList -smtpServer "$mailSmtp"  
}

# Apendix

## Datasource versions

|  |  |
| --- | --- |
| Version | Comments |
| 1.0.0 | Initial version |