Company Name

Tel: +44 1234 567 9898
Fax: +44 1234 545 9999
email: info@@company.com

THIS DOCUMENT CREATED

using

PScodePrint

Macro Variables Substitution Example

General Date	14/01/2017 9:47:06 AM
Long Date	Saturday, 14 January 2017
Short Date	14/01/2017
Long Time	9:47:06 AM
Short Time	09:47
Today	14/01/2017 9:47:06 AM

Prepared by
Joginder S Nahil
on
14/01/2017 9:47:06 AM

WWW.STARPRINTTOOLS.COM

```
#-----------------------------------------------------------------------
# Create Web Sites Script
#
# Created for MedAssets - April 2014
# ToDo:  Move the functions into a separate module.
#        Updating the functions to use Param construct (cleaning code)
#-----------------------------------------------------------------------
# History
#-----------------------------------------------------------------------
# Version 0.1 - Initial creation.
# Version 0.2 Enhancements
#     Use the current system's IP address rather than input argument.
#     Fixed the issue writing to a physical log file for feedback.
# Version 0.3 - Updates.
#  If a directory for a physical path does not exist, it will be created.
#  Minor tweaks to keep the script PowerShell 2.0 compatible for the
#  target server environment.
#  Added change to XML file to indicate whether to add a web application
#  or a virtual directory to the site.
#  Changed the check in the AddSSLBinding function when checking on the
#  existence of the SSL association with a specific site.
# Version 0.4 - Updates
#  Code was added to ensure the Web App Conversion took place ~line 500
#  Updated AddSSLBinding for the else condition of the SslInstall check
#  Added Check-CertificateLocation
#  The current script name is written in the log file
#  The location of the certificate file is read from the XML file
#  The log file name is always the same name.
# Version 0.5 - Updates
#  Changed the for loop to a for each loop.  This included changes where
#  the site's applications are assigned and the loop for the site bindings.
# Version 0.6 - Updates
#  Added new functions Add-HttpHeader, Add-HttpHeaders to the script.
#  Added the CustomHeaders XML elements to the SiteInfoInput file to allow
#  for including specific HTTP Headers to be added at the site or application
#  host level.
#-----------------------------------------------------------------------

#region Helper Functions
#-----------------------------------------------------------------------
# Functions to assist in facilitate processing.
#-----------------------------------------------------------------------

Function Add-ReportMessage(){
    Param(
    [string] $inputReportLine,
    [string] $ForegroundColor="White"
    )

     Write-Host -ForegroundColor $ForegroundColor $($inputReportLine)

     Out-File -FilePath FileSystem::"C:\$ReportFileName" -InputObject
    $inputReportLine -Encoding ASCII -Append
}

Function Check-FolderPath{
    Param([string] $TargetDir)

    if(!(Test-Path -Path $TargetDir )){
        Add-ReportMessage "$TargetDir Not Found"
        New-Item -ItemType directory -Path $TargetDir
        $true
    }
    else
    {
```

```
65          Add-ReportMessage "$TargetDir Found"
66          $true
67      }
68
69      if(!(Test-Path -Path $TargetDir ))
70      {
71          Add-ReportMessage "Unable to create folder: $TargetDir"
72          $false
73      }
74
75  }
76
77  Function Check-InputFile{
78      param(
79      [parameter(Position=0,Mandatory=$true,HelpMessage="Name of file to check")]
79
80      [string] $fileNamePath)
81      return Test-Path -Path $fileNamePath
82  }
83
84  function Check-CertificateLocations{
85      param($listItems)
86
87      [bool]$Continue = $true
88
89      for ($i=0;$i -lt $listItems.site.Count; $i++){
90
91          $site = $listItems.site[$i]
92          $workingSiteName = $site.name
93          $certificate = $site.Certificate
94          $workingLocation = $certificate.fileLocation
95
96          $Continue = Check-InputFile -fileNamePath $certificate.fileLocation
97
98          if(-not($Continue)){
99              Add-ReportMessage
99              "The Certificate for $workingSiteName cannot be found at $workingLo
99              cation"
100             Return $Continue
101         }
102     }
103
104     Return $Continue
105 }
106
107 Function Check-ModuleLoaded{
108     Param([string] $moduleName)
109
110     if( -not(Get-Module -name $moduleName))
111     {
112         Add-ReportMessage "$moduleName module is not loaded"
113         if(Get-Module -ListAvailable | Where-Object{$_.name -eq $moduleName})
114         {
115             Import-Module -Name $moduleName
116             Add-ReportMessage "$moduleName has been loaded"
117             $true
118         }
119         else
120         {
121             Add-ReportMessage "$moduleName is not available to load"
122             $false
123         }
124     }
125     else
126     {
```

```
127             Import-Module -Name $moduleName
128             Add-ReportMessage "$moduleName is loaded"
129             $true
130         }
131     }
132
133     Function Get-Time(){
134         # Capture the time
135         [string] $time = Get-Date -UFormat %T
136         Return $time
137     }
138
139     #endregion
140
141     #region IIS Functions
142
143     function Test-WebAppPool($Name) {
144         return Test-Path "IIS:\AppPools\$Name"
145     }
146
147     function Get-WebAppPool($Name) {
148         return Get-Item "IIS:\AppPools\$Name"
149     }
150
151     function Add-WebAppPool($Name, $Identity = "NetworkService", $RuntimeVersion =
151     "v4.0", $PipelineMode = "Integrated") {
152         if (Test-WebAppPool $Name) {
153             Add-ReportMessage -ForegroundColor Cyan
153             "Application pool $Name already exists"
154         }
155         else {
156             Add-ReportMessage -ForegroundColor Green
156             "Creating application poool $Name"
157
158             $appPool = New-WebAppPool -Name $Name
159             $appPool.managedRuntimeVersion = $RuntimeVersion
160             $appPool.managedPipelineMode = $PipelineMode
161             $appPool.processModel.identityType = $Identity
162
163             $appPool | Set-Item
164         }
165     }
166
167     function Test-Website($Name) {
168         return Test-Path "IIS:\Sites\$Name"
169     }
170
171     function Get-Website($Name) {
172         return Get-Item "IIS:\Sites\$Name"
173     }
174
175     function Add-Website($Name, $PhysicalPath, $ApplicationPool = "DefaultAppPool",
175     $Port = 80) {
176         if (Test-Website $Name) {
177             $webSite = Add-ReportMessage -ForegroundColor Cyan
177             "Web site $Name already exists`n"
178         }
179         else {
180             Add-ReportMessage -ForegroundColor Green "Creating web site $Name`n"
180
181             if((test-path $physicalPath) -eq $false){
182                 Add-ReportMessage
182                 "Creating new folder $physicalpath (Add-Website)"
183                 New-Item -ItemType directory -Path $physicalpath
```

```
1  2  3
184             }
185               $webSite = New-Website -Name $Name -PhysicalPath $PhysicalPath
185             -ApplicationPool $ApplicationPool -Port $Port
186               $webSite.serverAutoStart = $true
187
188               $webSite | Set-Item
189           }
190       }
191
192       function Test-WebApplication($Site, $Name) {
193           return Test-Path "IIS:\Sites\$Site\$Name"
194       }
195
196       function Get-WebApplication($Site, $Name) {
197           return Get-Item "IIS:\Sites\$Site\$Name"
198       }
199
200       function Add-WebApplication($Site, $Name, $PhysicalPath, $ApplicationPool =
200       "DefaultAppPool") {
201           if (Test-WebApplication $Site $Name) {
202                 Add-ReportMessage -ForegroundColor Cyan
202               "Web application for this name: $Name already exists`n"
203               Remove-WebApplication -Site $Site -Name $Name
204               Add-ReportMessage -ForegroundColor Cyan
204               "Web application for this name: $Name Removed`n"
205           }
206           #else
207           #{
208            Add-ReportMessage -ForegroundColor Green
208           "Creating web application $Name`n"
209
210             if((test-path $physicalPath) -eq $false){
211                 Add-ReportMessage
211               "Creating new folder $physicalpath (Add-WebApplication)"
212                 New-Item -ItemType directory -Path $physicalpath
213              }
214
215            $webApp = New-WebApplication -Site $Site -Name $Name -PhysicalPath
215           $PhysicalPath -ApplicationPool $ApplicationPool
216           #}
217       }
218
219       function Test-WebVirtualDirectory($Site, $Name) {
220           return Test-Path "IIS:\Sites\$Site\$Name"
221       }
222
223       function Get-WebVirtualDirectory($Site, $Name) {
224           return Get-Item "IIS:\Sites\$Site\$Name"
225       }
226
227       function Add-WebVirtualDirectory($Site, $Name, $PhysicalPath, $Application =
227       $null) {
228           if (Test-WebVirtualDirectory -Site $Site -Name $Name) {
229                 Add-ReportMessage -ForegroundColor Cyan
229               "Web virtual directory for this name: $Name already exists`n"
230           }
231           else {
232                 if((test-path $physicalPath) -eq $false){
233                   Add-ReportMessage
233                   "Creating new folder $physicalpath (Add-WebVirtualDirectory)"
234                   New-Item -ItemType directory -Path $physicalpath
235                 }
236
237               Add-ReportMessage -ForegroundColor Green
1  2
```

```
237             "Creating web virtual directory $Name`n"
238               $webDir = New-WebVirtualDirectory -Site $Site -Name $Name
238             -PhysicalPath $PhysicalPath -Application $Application
239         }
240     }
241
242     ------------------------------------------------------------------------
242     Function Add-HttpHeader{
243         Param(
244             [parameter(Position=0,Mandatory=$true,HelpMessage=
244         "Name of the HTTPHeader to add")]
245             [string] $headerName,
246             [parameter(Position=1,Mandatory=$true,HelpMessage=
246         "Value of the HTTPHeader to add")]
247             [string] $headerValue
248          )
249         try{
250
251                 $addElement = $customHeadersCollection.CreateElement("add")
252                 $addElement["name"] = $headerName
253                 $addElement["value"] = $headerValue
254                 $customHeadersCollection.Add($addElement) | Out-Null
255                 $iis.CommitChanges()
256
257                 Add-ReportMessage
257           "`tAdded new HTTPHeader Name: $headerName, Value: $headerValue"
258         }
259         catch{
260                 Add-ReportMessage $Error[0].ToString()
261         }
262     }
263
264     ------------------------------------------------------------------------
264     Function Add-HttpHeaders{
265         Param(
266         [parameter(Position=0, Mandatory=$true, HelpMessage=
266         "Name of the site to add the HttpHeader")]
267         [string] $siteName,
268         [parameter(Position=1, Mandatory=$false, HelpMessage=
268         "Apply the HttpHeader to the application host level?")]
269         [bool] $applyToHost = $false,
270         [parameter(Position=2, Mandatory=$true, HelpMessage=
270         "Name of the HTTPHeader to add")]
271         [string] $headerName,
272         [parameter(Position=3, Mandatory=$true, HelpMessage=
272         "Value of the HTTPHeader to add")]
273         [string] $headerValue
274         )
275
276         #-----------------------------------------------------------------
277         # Note:  This step is IMPORTANT.
278         #  The $iis object this script refers to will not work properly without
279         #  first loading the Microsoft.Web.Administration assembly.
280         #-----------------------------------------------------------------
281         [System.Reflection.Assembly]::LoadWithPartialName(
281         "Microsoft.Web.Administration") | Out-Null
282
283         $iis = new-object Microsoft.Web.Administration.ServerManager
284
285         #-----------------------------------------------------------------
286         # If a site name is supplied then this HttpHeader will be added for
287
287         # that specific web site.  Otherwise it will be applied to the application h
287         ost
288         #-----------------------------------------------------------------
289         if ($applyToHost -eq $true){
290             Add-ReportMessage
```

```
290         "`n`t**************************************************"
291         Add-ReportMessage
291         "`t* Adding Custom HTTP Header to the application host"
292         Add-ReportMessage "`t* Header Name: $headerName"
293         Add-ReportMessage "`t* Header Value: $headerValue"
294         Add-ReportMessage
294         "`t**************************************************`n"
295
296         $config =$iis.GetApplicationHostConfiguration()
297     }
298     else{
299         Add-ReportMessage
299         "`n`t**************************************************"
300         Add-ReportMessage
300         "`t* Adding Custom HTTP Header to the site: $siteName"
301         Add-ReportMessage "`t* Header Name: $headerName"
302         Add-ReportMessage "`t* Header Value: $headerValue"
303         Add-ReportMessage
303         "`t**************************************************`n"
304
305         $config = $iis.GetWebConfiguration($siteName)
306     }
307
308     $httpProtocolSection = $config.GetSection("system.webServer/httpProtocol")
309     $customHeadersCollection = $httpProtocolSection.GetCollection(
309     "customHeaders")
310
311     # Flag to indicate whether to add the custom HTTP Header or not.
312     $foundHeader = $false
313
314     foreach($attrib in $customHeadersCollection){
315
316         $testHeader = $attrib.RawAttributes.name
317         $testAttribute = $attrib.RawAttributes.value
318
319         if($testHeader -eq $headerName){
320             if($attrib.RawAttributes.value -eq $headerValue){
321                 Add-ReportMessage
321                 "`tThe header $testHeader with the value $testAttribute already
321                  exists in this location."
322                 $foundHeader = $true
323                 break
324             }
325         }
326     }
327
328     if ($foundHeader -ne $true){
329         Add-HttpHeader -headerName $headerName -headerValue $headerValue
330     }
331 }
332
333 #endregion
334
335 #region Other IIS Functions
336
337 function AddSSLBinding{
338     Param(
339     [string]$thumbprint,
340     [string]$sitename,
341     [int]$port,
342     [string]$hostheader,
343     [string] $ipAddress)
344
345       # Check to see if the SSL has been installed previously
346     $SslInstalled = Get-ChildItem IIS:\sslbindings | ? {$_.Port -eq $port} |
346     ? {$_.Sites -eq $sitename}
347
```

```
347
348         if($SslInstalled -ne $null){
349             Add-ReportMessage "`nSSL binding already exists on port $port for $sitename"
350             Write-Warning "SSL binding already exists on port $port for $sitename"
351             #Add-ReportMessage "Adding SSL binding again to be sure.`n"
352
352             #New-Item IIS:\SslBindings\$ipAddress!$port -Value $cert | out-null
353         }
354         else{
355             Add-ReportMessage "Adding SSL binding on $port for $sitename.`n"
356             New-Item IIS:\SslBindings\$ipAddress!$port -Value $cert | out-null
356
357         }
358
359         if ($hostheader){
360
360             #--------------------------------------------------------------------
360
361               # This works well in PowerShell 2.0
362
362             #--------------------------------------------------------------------
362
363               New-ItemProperty $(join-path iis:\Sites $sitename) -name bindings
363             -value @{protocol="https";bindingInformation=
363             "$($ipAddress):$($port):$($hostheader)";certificateStoreName="My";
363         certificateHash=$thumbprint} | Write-host # out-null
364     }
365     else{
366           New-ItemProperty $(join-path iis:\Sites $sitename) -name bindings -value
366         @{protocol="https";bindingInformation="$($ipAddress):$($port):";
366 certificateStoreName="My";certificateHash=$thumbprint} | Write-host # out-null
367       }
368
369   }
370
371 Function Create-AppWebPool{
372       param(
373       [parameter(Position=0,Mandatory=$true,HelpMessage="Name of AppPool")]
374       [string]$PoolName,
375       [parameter(Position=1,Mandatory=$false)]
376       [string]$pipelineMode,
377       [parameter(Position=2,Mandatory=$false)]
378       [string]$RuntimeVersion,
379       [parameter(Position=3,Mandatory=$false)]
380       [bool]$enable32BitAppOnWin64)
381
382       new-webapppool -Name $PoolName
383
384       # Integrated is the default
385       if( $pipelineMode -ne $null -and $pipelineMode -eq "Classic")
386       {
387           set-ItemProperty $PoolName -name "managedPipelineMode" -Value 1
388       }
389
390       # The default value is the current .NET framework
391       if($RuntimeVersion.Length -ne 0)
392       {
393           set-ItemProperty $PoolName -name "managedRuntimeVersion" -Value
393           $RuntimeVersion
394       }
395       else
396       {
397           set-ItemProperty $PoolName -name "managedRuntimeVersion" -Value "v4.0"
397
```

```
1  2
398        }
399
400        # The default is false
401        if($enable32BitAppOnWin64)
402        {
403            set-ItemProperty $PoolName -name "enable32BitAppOnWin64" -Value true
404        }
405    }
406
407    ------------------------------------------------------------------------
407    Function Import-PfcCertificate{
408        param(
409           [String]$certPath,
410           [String]$certRootStore = "CurrentUser",
411           [String]$certStore = "My",$pfxPass = $null)
412
413        Add-ReportMessage "Import-PfcCertificate`t Certification Path: $certPath"
414        Add-ReportMessage "`tCertification Root Store: $certRootStore"
415
416        $pfx = new-object
416        System.Security.Cryptography.X509Certificates.X509Certificate2
417
418        if ($pfxPass -eq $null)
419        {
420            $pfxPass = read-host "Enter the pfx password" -assecurestring
421        }
422        $pfx.import($certPath,$pfxPass,"Exportable,PersistKeySet")
423
424        $store = new-object System.Security.Cryptography.X509Certificates.X509Store
424        ($certStore,$certRootStore)
425        $store.open("MaxAllowed")
426        $store.add($pfx)
427        $store.close()
428    }
429
430    #endregion
431
432    #region Process AppPoolsFile
433
434    ------------------------------------------------------------------------
434    Function Process-AppPoolsFile{
435        param(
436        [parameter(Position=0,Mandatory=$true,HelpMessage=
436        "Name of XML file (apppools.xml)")]
437        [string] $xmlFileName)
438
439        if(!(test-path $xmlFileName))
440        {
441            Add-ReportMessage "`n`nThe file $xmlFileName could not be found"
442            Add-ReportMessage "Exiting the function"
443        }
444        else
445        {
446            Add-ReportMessage "Processing $xmlFileName`n"
447
448            #Import-Module WebAdministration # Should already be loaded
449
450            # Create a list of the desired app pools
451
451            #$inputFile =   "C:\Users\tgoodwin\Documents\apppools.xml" #Testing fil
451            e
452
453            [xml] $xmlInput = (Get-Content $xmlFileName)
454            $listItems = $xmlInput.appcmd.APPPOOL
455
456            # Point to the AppPools location
457            cd IIS:\AppPools
458
1  2
```

```
459             for($j=0; $j -lt $listItems.Count; $j++)
460             {
461
462                 $PoolName = $listItems[$j]."APPPOOL.NAME"
462               # Attribute of parent node
463                 $pipelineMode = $listItems[$j]."PipelineMode"
463               # Attribute of parent node
464                 $RuntimeVersion = $listItems[$j]."RuntimeVersion"
464               # Attribute of parent node
465                 $enable32BitAppOnWin64 = $listItems[$j].add.
465             "enable32BitAppOnWin64" # Embedded within the child node
466
467
468                 if (test-WebAppPool $listItems[$j]."APPPOOL.NAME")
469                 {
470                     Add-ReportMessage "Delete and Create: $PoolName"
471                     Add-ReportMessage "`tPoolName: $PoolName"
472                     Add-ReportMessage "`tpipelineMode: $pipelineMode"
473                     Add-ReportMessage "`tRuntimeVersion: $RuntimeVersion"
474                     Add-ReportMessage
474               "`tenable32BitAppOnWin64: $enable32BitAppOnWin64`n"
475
476                     remove-WebAppPool $PoolName
477
478                 }
479                 else
480                 {
481                     Add-ReportMessage "****Create new AppPool****"
482                     Add-ReportMessage "`tPoolName: $PoolName"
483                     Add-ReportMessage "`tpipelineMode: $pipelineMode"
484                     Add-ReportMessage "`tRuntimeVersion: $RuntimeVersion"
485                     Add-ReportMessage
485               "`tenable32BitAppOnWin64: $enable32BitAppOnWin64`n"
486                 }
487
488                 if($enable32BitAppOnWin64 -ne "true")
489                 {
490                     Create-AppWebPool -PoolName $PoolName -pipeline
490               $pipelineMode -RuntimeVersion $RuntimeVersion
490               -enable32BitAppOnWin64 $false
491                 }
492                 else
493                 {
494                     Create-AppWebPool -PoolName $PoolName -pipeline
494               $pipelineMode -RuntimeVersion $RuntimeVersion
494               -enable32BitAppOnWin64 $true
495                 }
496             }
497         }
498 }
499
500   #endregion
501
502   ____________________________________________________________________
    Function Main{
503
504       # ensure we are at the correct starting point
505       cd $scriptPath
506
507       $Continue = $true
508
509       # Check for and delete any existing Log file.
510       if(Test-Path -Path "C:\$ReportFileName"){
511           Remove-Item -Path "C:\$ReportFileName"
512       }
513
514       # Check for input files
```

```
515     Add-ReportMessage "Script Started: $startTime"
516
517     if (-not (Test-Path -Path ".\SiteInfoInput.xml")){
518         Add-ReportMessage
518         "The SiteInfoInput.xml could not be found.`nExiting the script"
519         Return
520     }
521
522     if (-not (Test-Path -Path ".\apppools.xml")){
523         Add-ReportMessage
523         "The apppools.xml could not be found.`nExiting the script"
524         Return
525     }
526
527     $SitesInput = ".\SiteInfoInput.xml"
528     $AppPoolsInput = ".\apppools.xml"
529
530     [xml] $xmlInput = (Get-Content $SitesInput)
531     $listItems = $xmlInput.Sites
532     $certLocation = $listItems.Sites.Certificate
533
534     # Check for certificate file(s)
535     $Continue = Check-CertificateLocations $listItems
536
537     if(!$Continue){
538         Add-ReportMessage
538         "At least one Certificate file could not be found.`nExiting the script"
538
539         Return
540     }
541
542     # Check that module is loaded
543     $Continue = Check-ModuleLoaded "WebAdministration"
544
545     if (!$Continue){
546         Add-ReportMessage
546         "The WebAdministration module could not be found.`nExiting the script"
546
547         Return
548     }
549
550     # Process application pools
551     Add-ReportMessage("`n***** Creating the App Pools *****`n")
552     Process-AppPoolsFile -xmlFileName $AppPoolsInput
553
554     # Create web sites
555     Add-ReportMessage("`n***** Creating the web sites *****`n")
556     Import-Module WebAdministration
557
558     cd C:
559
560     # Point to the Web Sites location
561     cd IIS:\Sites
562
563     #Loop throught the list of Sites from the XML file
564     foreach ($site in $listItems.ChildNodes){
565         $siteName = $site.name
566         $physicalpath = $site.physicalPath
567         $applicationPool = $site.ApplicationPool
568         $customerHeaders = $site.CustomHeaders
569
570         if(Test-Website -Name $siteName){
571             Add-ReportMessage
571             "The site $sitename already exists on this server, so it will be re
571             moved."
572             Remove-Website -Name $siteName
```

```
573             }
574
575             # Check to see if the physical path exists
576             if((Test-Path $physicalpath) -eq $false){
577                 $physicalpath
578                 Add-ReportMessage
578                 "The physical Path $physicalpath does not exist and will be created
578                 ."
579                 New-Item -ItemType directory -Path $physicalpath
580             }
581
582             if((Test-path $physicalpath) -eq $false){
583                 Add-ReportMessage
583                 "The physical Path $physicalpath does not exist and was not created
583                 ."
584             }
585             else {
586                 Add-ReportMessage
586                 "Creating new web site -Name $siteName -PhysicalPath $physicalpath
586                 -ApplicationPool $applicationPool"
587                 New-Website -Name $siteName -PhysicalPath $physicalpath
587                 -ApplicationPool $applicationPool
588
589
589                 # Remove the default binding, since we are going to add it in the n
589                 ext step
590                 remove-webBinding -Name $siteName
591
592                 # Create applications and Virtual Directories
593                 $webApplications = $site.applications
594
595                 for($k=0; $k -lt $webApplications.ChildNodes.Count; $k++){
596                     $webApp = ""
597                     $webAppPool = ""
598                     $webAppVirtDir = ""
599                     $VirtDirpath = ""
600                     $VirtDirPhyPath =""
601                     $ConvertToApp = $false
602                     $applicationName = ""
603
604                     #----------------------------------------------------
605                     # This works in PowerShell 3.0.  This information is
606                     # left for future reference
607                     #----------------------------------------------------
608                     # $bindings.ChildNodes.Item($j).protocol
609                     #----------------------------------------------------
610                     # $webApp = $webApplications.ChildNodes[$k]
611                     #----------------------------------------------------
612                     # PowerShell 3.0 compatible approach commented out.
613                     #----------------------------------------------------
614
615                     $webAppPath = $webApplications.ChildNodes.Item($k).path
615                 #$webApp.path
616                     $webAppPool = $webApplications.ChildNodes.Item($k).
616                 applicationPool #$webApp.applicationPool
617                     $ConvertToApp = $webApplications.ChildNodes.Item($k).
617                 converttoApplication #$webApp.converttoApplication
618                     $webAppVirtDir = $webApplications.ChildNodes.Item($k).
618                 virtualDirectory #$webApp.virtualDirectory
619                     $VirtDirpath = $webAppVirtDir.path
620                     $VirtDirPhyPath = $webAppVirtDir.physicalPath
621                     $applicationName = $webAppPath.Substring(1)
621                 # Added for Web App Conversion
622
623
623                 #Add-ReportMessage $siteName `t $webAppPath `t $webAppPool `t $
```

```
1   2   3   4
623             webAppVirtDir `t $VirtDirpath `t $VirtDirPhyPat
624
625             if($ConvertToApp -eq $true){
626                 if((Test-WebApplication -Site $siteName -Name
626               webAppPath) -eq $false){
627                     Add-ReportMessage
627                   "`tNew Web Application -Site $siteName -Name $webAppPa
627                   th -PhysicalPath $VirtDirPhyPath -ApplicationPool $web
627                   AppPool"
628                     Add-WebApplication -Site $siteName -Name
628                   $webAppPath -PhysicalPath $VirtDirPhyPath
628                   -ApplicationPool $webAppPool
629
629                   # Ensure conversion to a Web Application from a Virtua
629                   l Directory.
630                     #if($applicationName.Length -gt 0){
631
632                      #   $iisPath = ""
633
633                   #   $iisPath = "IIS:\Sites\" + $siteName + "\" + $appl
633                   icationName
634
634                   #  ConvertTo-WebApplication -ApplicationPool $webAppPo
634                   ol -PSPath $iisPath
635                      #  }
636
637                 }
638                 else{
639                     Add-ReportMessage
639                   "The Web Application $webAppPath for the $siteName sit
639                   e already exists."
640                  }
641             }
642             else{
643                 if((Test-WebVirtualDirectory -Site $siteName) -eq
643               $false){
644                     Add-ReportMessage
644                   "`tNew Virtual Directory -Site $siteName -Name $webApp
644                   Path -PhysicalPath $VirtDirPhyPath -Application $webAp
644                   pPath"
645                     Add-WebVirtualDirectory -Site $siteName -Name
645                   $webAppPath -PhysicalPath $VirtDirPhyPath
645                   -Application $webAppPath
646                 }
647                 else{
648                     Add-ReportMessage
648                   "The Virtual Directory for the $siteName site already
648                   exists."
649                  }
650             }
651         }
652
653         # Install the associated Certificate for this site
654         $certificate = $site.Certificate
655
656         $CertificatInput = $certificate.fileLocation
657         $CertificatePassword = $certificate.password
658
659         Add-ReportMessage
659         "Installing the certificate found at $CertificatInput`n"
660
661         # Install the SSL certificate
662         certutil -p $CertificatePassword -importpfx $CertificatInput
663
664         # Get the thumbprint for the SSL certificate
665         $Cert = dir cert:\localmachine\my | Where-Object {$_.Subject -like
1   2   3
```

```
665             '*.medassets.com*' }
666         $Thumbprint = $Cert.Thumbprint.ToString()
667
668         # Add the Bindings
669         $bindings = $site.bindings
670         $cCount = $bindings.ChildNodes.Count
671
672         for($j=0; $j -lt $cCount; $j++){
673             #$binding = $bindings.ChildNodes[$j]
674             #$binding = $bindings.Item[$j]
675
676             $protocol = $bindings.ChildNodes.Item($j).protocol
676         #$binding.protocol
677             $port = $bindings.ChildNodes.Item($j).port #$binding.Port
678             $hostHdr = $bindings.ChildNodes.Item($j).HostHeader
678         #$binding.HostHeader
679
680             #Add-ReportMessage  $protocol `t $port `t $hostHdr
681             #Add-ReportMessage `n
682             if($protocol -eq "https")
683             {
684                 Add-ReportMessage
684               "`tNew SSL Binding -Name $siteName -IPAddress $WorkingIPad
684               dress -Port $port -HostHeader $hostHdr -SslFlags 0"
685                 AddSSLBinding -sitename $siteName -port $port
685               -hostheader $hostHdr -thumbprint $Thumbprint -ipAddress
685               $WorkingIPaddress
686             }
687             else
688             {
689                 Add-ReportMessage
689               "`tNew Web Binding -Name $siteName -IPAddress $WorkingIPad
689               dress -Port $port -HostHeader $hostHdr"
690                 New-WebBinding -Name $siteName -IPAddress
690               $WorkingIPaddress -Port $port -HostHeader $hostHdr
691             }
692         }
693
694         $hdrCount = $customHeaders.ChildNodes.Count
695
696         if($hdrCount -gt 0){
697
698             Add-ReportMessage (
698             "`nThere are $hdrCount Custom HTTP Headers to add.`n")
699
700             # Add any Custom HTTP Headers
701             foreach ($customHeader in $customHeaders.ChildNodes){
702
703                 [string] $wrkgHeaderName = $customHeader.headerName
704                 [string] $wrkgHeaderValue = $customHeader.headerValue
705
706                 if($customHeader.applyToHostLevel.ToString().ToUpper()
706               -eq "TRUE"){
707                     $wrkgHostLevel = $true
708                 }
709                 else{
710                     $wrkgHostLevel = $false
711                 }
712
713                 if(($wrkgHeaderName.length -gt 0) -and ($wrkgHeaderValue.
713               length -gt 0)){
714                     Add-HttpHeaders -siteName $siteName -applyToHost
714                     $wrkgHostLevel -headerName $wrkgHeaderName
714                     -headerValue $wrkgHeaderValue
715                 }
716             }
```

```
1    2    3    4
717                 }
718                 else{
719
720                     Add-ReportMessage (
720                     "`nThere are no Custom HTTP Headers to add.`n")
721                 }
722
723             } #Bottom of if statement (Test-path $physicalpath)
724         }
725     }
726
727     #--------------------------------------------------------------------
728     # Script Entry point.
729     #
730     # This process expects the following files in the same physical folder:
731     #
732     #    apppools.xml - The list of application pools to create
733     #    SiteInfoInput.xml - the list of the web sites, applications, and
734     #        virtual directories to create.
735     #    Medassets_Server_Certificate.pfx - The server certificate file
736     #    The Certificate password is currently hard coded as "test"
737     #
738     # If the files are not found the script will end.
739     #--------------------------------------------------------------------
740
741     # Get current IP address of the current system
742     [string]$WorkingIPaddress = (Get-WmiObject -class
742     win32_NetworkAdapterConfiguration -Filter 'ipenabled = "true"').ipaddress[0]
743     [string] $scriptName = $MyInvocation.MyCommand.Name
744
745     # Initialization of script variables
746     $startTime = Get-Date -UFormat %T
747     $workingDate = Get-Date
748     #$f = "-reportLog.txt"
749     $ReportFileName = "CreateWebSites-reportLog.txt"
749     #Get-Date -UFormat $d%Y%m%d%H%M$f
750     [string]$Thumbprint = " "
751
752     # Capture starting point
753     $scriptPath = split-path -parent $MyInvocation.MyCommand.Definition
754
755     Add-ReportMessage "****************************************"
756     Add-ReportMessage "* Log Report for Creating Web Sites"
757     Add-ReportMessage "* Script Name: $scriptName"
758     Add-ReportMessage "* Execution Date: $workingDate"
759     Add-ReportMessage "****************************************`n"
760
761     # Check that an IP address is passed as an argument to this script
762     if($WorkingIPaddress.Length -ne 0)
763     {
764         Main
765     }
766     else
767     {
768         Add-ReportMessage "No IP address was found for this system."
769     }
770
771     $endTime = Get-Date -UFormat %T
772     $duration = New-TimeSpan -Start $startTime -End $endTime
773     $msg = $duration.Minutes.ToString() + " minutes,"
774     $msg = $msg + $duration.Seconds.ToString() + " seconds"
775
776     Add-ReportMessage "`n****************************************"
777     Add-ReportMessage "* Script started: $startTime"
778     Add-ReportMessage "* Script Ended : $endTime"
779     Add-ReportMessage "* Script Lasted: $msg"
```

```
780     Add-ReportMessage "****************************************"
781
782     Write-Host "***** The log file can be found at C:\$ReportFileName *****"
783
784     cd $scriptPath # Return to our starting point
784
784
784
784     ********************************************************************************
        *
784     **********************
784     *** The script has the following errors:
784
784     At line 410 column 33:  Missing expression after '='..
784     At line 410 column 31:  Missing ')' in function parameter list..
784     At line 410 column 47:  Missing argument in parameter list..
784
784     At line 407 column 31:  Missing closing '}' in statement block or type definitio
784     n..
784     At line 411 column 51:  Unexpected token ')' in expression or statement..
784     At line 422 column 36:  Missing expression after ','..
784
784     At line 422 column 36:  Unexpected token '?Exportable' in expression or statemen
784     t..
784     At line 422 column 47:  Missing argument in parameter list..
784     At line 422 column 62:  Unexpected token ')' in expression or statement..
784     At line 425 column 17:  Missing ')' in method call..
784
784     At line 425 column 17:  Unexpected token '?MaxAllowed?' in expression or stateme
784     nt..
784     At line 425 column 29:  Unexpected token ')' in expression or statement..
784     At line 428 column 1:  Unexpected token '}' in expression or statement..
784
784     ********************************************************************************
        *
784     **********************
```

X:\Dev\Products\PScodePrint\4.0\bin\scripts\DTW.PS.PrettyPrinterV1.psd1

```
1   @{ ModuleVersion     = "1.0.0"
2     Author            = "Dan Ward"
3     CompanyName       = "DTWConsulting.com"
4     Copyright         = "Copyright 2012"
5     Description       =
5   "PowerShell code cleaner / pretty printer v1 for PowerShell v2"
6     GUID              = "{222A4EF8-9A04-4240-AE0C-18A0CDED5248}"
7     PowerShellVersion = "2.0"
8     CLRVersion        = "2.0"
9     NestedModules     = "DTW.PS.FileSystem.Encoding.psm1",
10                      "DTW.PS.PrettyPrinterV1.psm1"
11  }
```

X:\Dev\Products\PScodePrint\4.0\bin\scripts\Show-ColorizedContent.ps1

```
#requires -version 2.0

param(
   $filename = $(throw "Please specify a filename."),
   $highlightRanges = @(),
   [System.Management.Automation.SwitchParameter] $excludeLineNumbers)


# [Enum]::GetValues($host.UI.RawUI.ForegroundColor.GetType()) | % { Write-Host -Fore $_ "$_" }
$replacementColours = @{
   "Command"="Yellow";
   "CommandParameter"="Yellow";
   "Variable"="Green" ;
   "Operator"="DarkCyan";
   "Grouper"="DarkCyan";
   "StatementSeparator"="DarkCyan";
   "String"="Cyan";
   "Number"="Cyan";
   "CommandArgument"="Cyan";
   "Keyword"="Magenta";
   "Attribute"="DarkYellow";
   "Property"="DarkYellow";
   "Member"="DarkYellow";
   "Type"="DarkYellow";
   "Comment"="Red";
}
$highlightColor = "Green"
$highlightCharacter = ">"

## Read the text of the file, and parse it
$file = (Resolve-Path $filename).Path
$content = [IO.File]::ReadAllText($file)
$parsed = [System.Management.Automation.PsParser]::Tokenize($content, [ref] $null) |
   Sort StartLine,StartColumn

function WriteFormattedLine($formatString, [int] $line)
{
    if($excludeLineNumbers) { return }

    $hColor = "Gray"
    $separator = "|"
    if($highlightRanges -contains $line) { $hColor = $highlightColor;
    $separator = $highlightCharacter }
    Write-Host -NoNewLine -Fore $hColor ($formatString -f $line,$separator)
}

Write-Host

WriteFormattedLine "{0:D3} {1} " 1

$column = 1
foreach($token in $parsed)
{
    $color = "Gray"

    ## Determine the highlighting colour
    $color = $replacementColours[[string]$token.Type]
    if(-not $color) { $color = "Gray" }

    ## Now output the token
    if(($token.Type -eq "NewLine") -or ($token.Type -eq "LineContinuation"))
    {
        $column = 1
    1    2
```

```
1  2
62          Write-Host
63
64           WriteFormattedLine "{0:D3} {1} " ($token.StartLine + 1)
65      }
66      else
67      {
68          ## Do any indenting
69          if($column -lt $token.StartColumn)
70          {
71              Write-Host -NoNewLine (" " * ($token.StartColumn - $column))
72          }
73
74          ## See where the token ends
75          $tokenEnd = $token.Start + $token.Length - 1
76
77          ## Handle the line numbering for multi-line strings
78          if(($token.Type -eq "String") -and ($token.EndLine -gt $token.
78          StartLine))
79          {
80              $lineCounter = $token.StartLine
81              $stringLines = $(-join $content[$token.Start..$tokenEnd] -split
81              "`r`n")
82              foreach($stringLine in $stringLines)
83              {
84                  if($lineCounter -gt $token.StartLine)
85                  {
86                      WriteFormattedLine "`n{0:D3} {1}" $lineCounter
87                  }
88                  Write-Host -NoNewLine -Fore $color $stringLine
89                  $lineCounter++
90              }
91          }
92          ## Write out a regular token
93          else
94          {
95              Write-Host -NoNewLine -Fore $color (-join $content[$token.Start..
95              $tokenEnd])
96          }
97
98          ## Update our position in the column
99          $column = $token.EndColumn
100     }
101 }
102
103 Write-Host "`n"
```

X:\Dev\Products\PScodePrint\4.0\bin\scripts\VerifyBatchSettings.ps1

```
#-----------------------------------------------------------------------
# Verification Batch Execute SQL Scripts
#
# This script checks that all the components needed by the Batch Execution
# SQL Script are available and in place. It will also validate the input
# XML file settings.
#-----------------------------------------------------------------------
# History
#-----------------------------------------------------------------------
# Version 0.1 - Initial creation. T. Goodwin - 07/16/2014
#-----------------------------------------------------------------------

Function Get-SQLVersion{
    Param(
    [string] $ServerName,
    [string] $DbName
    )
       $q = 'select @@VERSION as versionInfo'
       $ver =  Invoke-Sqlcmd -Query $q -serverinstance "$ServerName"
    -database "$DbName"

       return $ver.versionInfo.Substring(0,25)
}

Function Main(){

    #region Step 1. Verify the existence of the input file
    Write-Host
    "`n$shortLine`nVerify the existence of the input file`n$shortLine`n"

    # Does the XML file exist?
    if(-not(Test-Path("$scriptpath\$XmlFileInputName"))){
        Write-Host "Couldn't find $scriptpath\$XmlFileInputName"
        Write-Host "Exiting the script."
        Return
    }
    else{
        Write-Host "The input file $scriptpath\$XmlFileInputName exists."
    }

    #region Read XML file configuration values.
    [xml] $xmlInput = (Get-Content $scriptpath\$XmlFileInputName)
    $targetServerName = $xmlInput.Inputs.targetServerName
    $targetDbName = $xmlInput.Inputs.targetDbName
    $sourceFolderName = $xmlInput.Inputs.sourceFolderName

    $inputMainPath = $xmlInput.Inputs.inputMainPath

    # Check for linked server
    $requiredLinkedServer = $xmlInput.Inputs.RequiredLinkedServer

    # Check for the Source Db information
    $sourceServerName = $xmlInput.Inputs.sourceServerName
    $sourceDbName = $xmlInput.Inputs.sourceDbName

    # which type of scripts to execute
    $executeCreateFk = $xmlInput.Inputs.FolderTypes.FkCreate
    $executeDropFk = $xmlInput.Inputs.FolderTypes.FkDrop
    $executeTable = $xmlInput.Inputs.FolderTypes.Table
    $executeUserFunctions = $xmlInput.Inputs.FolderTypes.UserDefinedFunction
    $executeView = $xmlInput.Inputs.FolderTypes.View
    $executeStoreProcs = $xmlInput.Inputs.FolderTypes.StoredProcedure
    $executeDropsOnly = $xmlInput.Inputs.FolderTypes.DropsOnly
    $executeModified = $xmlInput.Inputs.FolderTypes.Modified
    #endregion
1
```

```
1
64
65          Write-Host "`n$shortLine"
66          Write-Host "* Processing Information"
67          Write-Host "* Target ServerName: $targetServerName"
68          Write-Host "* Target Database: $targetDbName"
69          Write-Host "* Main Folder Path: $inputMainPath"
70          Write-Host "* Sub Folder Name: $sourceFolderName"
71          Write-Host "* Required Linked Server Name: $requiredLinkedServer"
72          Write-Host "$shortLine"
73          Write-Host "* Executing table/synonym scripts: $executeTable"
74          Write-Host "* Executing view scripts: $executeView"
75          Write-Host "* Executing stored procedure scripts: $executeStoreProcs"
76          Write-Host
76          "* Executing user defined function scripts: $executeUserFunctions"
77          Write-Host "* Executing drop FK scripts: $executeDropFk"
78          Write-Host "* Executing create FK scripts: $executeCreateFk"
79          Write-Host "* Executing drop only scripts: $executeDropsOnly"
80          Write-Host "* Executing modify only scripts: $executeModified"
81          Write-Host "$shortLine`n"
82
83          #endregion
84
85
85          #region Step 2. Verify the existence of the Folders that contain the SQL obj
85          ect files
86          Write-Host
86          "`n$shortLine`nVerify the existence of the Folders that contain the SQL obje
86          ct files`n$shortLine`n"
87          if (Test-Path("$inputMainPath\$sourceFolderName")){
88              Write-Host "`nThe $inputMainPath\$sourceFolderName was found`n"
89          }
90          else{
91              Write-Host "`nThe $inputMainPath\$sourceFolderName was NOT found`n"
92          }
93
94          if($executeTable -eq $true){
95              if (Test-Path("$inputMainPath\$sourceFolderName\Table")){
96                  Write-Host
96                  "The $inputMainPath\$sourceFolderName\Table was found`n"
97              }
98              else{
99                  Write-Host
99                  "The $inputMainPath\$sourceFolderName\Table was NOT found`n"
100             }
101         }
102
103         if($executeView -eq $true){
104             if (Test-Path("$inputMainPath\$sourceFolderName\View")){
105                 Write-Host "The $inputMainPath\$sourceFolderName\View was found`n"
105
106             }
107             else{
108                 Write-Host
108                 "The $inputMainPath\$sourceFolderName\View was NOT found`n"
109             }
110         }
111
112         if($executeStoreProcs -eq $true){
113             if (Test-Path("$inputMainPath\$sourceFolderName\StoredProcedure")){
114                 Write-Host
114                 "The $inputMainPath\$sourceFolderName\StoredProcedure was found`n"
114
115             }
116             else{
117                 Write-Host
117                 "The $inputMainPath\$sourceFolderName\StoredProcedure was NOT found
1   2   3
```

```
117             `n"
118         }
119     }
120
121     if($executeUserFunctions -eq $true){
122         if (Test-Path("$inputMainPath\$sourceFolderName\UserDefinedFunction"))
122         {
123             Write-Host
123             "The $inputMainPath\$sourceFolderName\UserDefinedFunction was found
123             `n"
124         }
125         else{
126             Write-Host
126             "The $inputMainPath\$sourceFolderName\UserDefinedFunction was NOT f
126             ound`n"
127         }
128     }
129
130     if($executeDropsOnly -eq $true){
131         if (Test-Path("$inputMainPath\$sourceFolderName\DropsOnly")){
132             Write-Host
132             "The $inputMainPath\$sourceFolderName\DropsOnly was found`n"
133         }
134         else{
135             Write-Host
135             "The $inputMainPath\$sourceFolderName\DropsOnly was NOT found`n"
136         }
137     }
138
139     if($executeModified -eq $true){
140         if (Test-Path("$inputMainPath\$sourceFolderName\Modified")){
141             Write-Host
141             "The $inputMainPath\$sourceFolderName\Modified was found`n"
142         }
143         else{
144             Write-Host
144             "The $inputMainPath\$sourceFolderName\Modified was NOT found`n"
145         }
146     }
147
148     if($executeCreateFk -eq $true){
149         if (Test-Path("$inputMainPath\$sourceFolderName\FkCreate")){
150             Write-Host
150             "The $inputMainPath\$sourceFolderName\FkCreate was found`n"
151         }
152         else{
153             Write-Host
153             "The $inputMainPath\$sourceFolderName\FkCreate was NOT found`n"
154         }
155     }
156
157     if($executeDropFk -eq $true){
158         if (Test-Path("$inputMainPath\$sourceFolderName\FkDrop")){
159             Write-Host
159             "The $inputMainPath\$sourceFolderName\FkDrop was found`n"
160         }
161         else{
162             Write-Host
162             "The $inputMainPath\$sourceFolderName\FkDrop was NOT found`n"
163         }
164     }
165
166     #endregion
167
168
168     #region Step 3. Verify that the SQL snappins or SQL Modules are available an
```

```
168         d loaded
169         Write-Host
169         "`n$shortLine`nVerify that the SQL snappins or SQL Modules are available and
169          loaded`n$shortLine`n"
170
171         $sqlModules = Get-Module -ListAvailable | ?{$_.Name -like "SQL*" }
172
173         if($sqlModules -eq $null){
174             Write-Host "There are NO PowerShell SQL modules available`n"
175              $flagLoadSQLModules = $false
176         }
177         else{
178             Write-Host "There are PowerShell SQL modules available"
179             Write-Host "The following modules are available: $sqlModules`n"
180             Write-Host "Loading the sqlps module`n"
181             #-----------------------------------------------------------------
181
182             # load the SQL PowerShell module.  The DisableNameChecking suppresses
183
183             # the message that warns you when you import a cmdlet or function whose
183
184             # name includes an unapproved verb or a prohibited character.
185             # The module will still be loaded.
186             # refer to http://technet.microsoft.com/en-us/library/hh849725.aspx
187             #-----------------------------------------------------------------
187
188             Import-Module "sqlps" -DisableNameChecking
189             $flagLoadSQLModules = $true
190         }
191
192         if($flagLoadSQLModules -eq $false){
193             Write-Host "Attempting to Load SQL Snappins"
194
195             # Verify that the sqlps module is not loaded
196             if ((Get-Module -Name "sqlps") -eq $null){
197
198                 if ((Get-PSSnapin -Name SqlServerProviderSnapin100 -ErrorAction
198                 SilentlyContinue) -eq $null ){
199                     Add-PsSnapin SqlServerProviderSnapin100
200                 }
201
202                 if ((Get-PSSnapin -Name SqlServerCmdletSnapin100 -ErrorAction
202                 SilentlyContinue) -eq $null ){
203                     Add-PsSnapin SqlServerCmdletSnapin100
204                 }
205
206                 if ((Get-PSSnapin -Name SqlServerProviderSnapin100 -ErrorAction
206                 SilentlyContinue) -eq $null ){
207                     Write-Host "Could not load SqlServerProviderSnapin100`n"
208                 }
209                 else{
210                     Write-Host "SqlServerProviderSnapin100 Loaded`n"
211                 }
212
213                 if ((Get-PSSnapin -Name SqlServerCmdletSnapin100 -ErrorAction
213                 SilentlyContinue) -eq $null ){
214                     Write-Host "Could not load SqlServerCmdletSnapin100`n"
215                 }
216                 else{
217                     Write-Host "SqlServerCmdletSnapin100 Loaded`n"
218                 }
219
220             }
221         }
222
223         #TODO - Check for SMO assembly?
```

```
1
224         #endregion
225
226
227         #region Step 4. Verify that SQL commands to the source and target SQL server
227         s can be run
228         Write-Host
228         "`n$shortLine`nVerify that SQL commands to the source and target SQL servers
228          can be run`n$shortLine`n"
229
230         Write-Host
230         "Executing a SQL command against the servers to check SQL Version`n"
231         $sourceDbVersion = Get-SQLVersion -ServerName $sourceServerName -DbName
231         $sourceDbName
232         $targetDbVersion = Get-SQLVersion -ServerName $targetServerName -DbName
232         $targetDbName
233
234         Write-Host
234         "The SQL version on the source SQL server, $sourceServerName, is $sourceDbVe
234         rsion`n"
235         Write-Host
235         "The SQL version on the target SQL server, $targetServerName, is $targetDbVe
235         rsion`n"
236
237         #endregion
238
239     }
240
241     #-----------------------------------------------------------------------
242     # Programming Notes
243     # ASSUMPTIONS: The XML input file is located in the same folder as this script.
243
244     #-----------------------------------------------------------------------
245     # Capture the dynamic script information
246     [string] $scriptName = $MyInvocation.MyCommand.Name
247     [string] $scriptpath = $MyInvocation.MyCommand.Path
248
249     # Capture the physical path where the script is running.
250     $scriptPath = split-path -parent $MyInvocation.MyCommand.Definition
251
252     $XmlFileInputName = "BatchScriptInput.XML"
253     $targetServerName = $null
254     $inputMainPath = $null
255     $targetDbName = $null
256     $targetServerName = $null
257     $sourceServerName = $null
258     $sourceDbName = $null
259     $RequiredLinkedServer = $null
260
261     # Set the $ErrorActionPreference to 'Stop' for development/Testing
262     $ErrorActionPreference = 'Stop'
263     #$ErrorActionPreference = 'Continue'
264
265     $border = '*'
266     $line = $border * 100
267     $shortLine = $border * 50
268     [bool] $flagLoadSQLModules = $false
269
270     main
271
272     Write-Host "Verification script execution complete"
```

X