Use PowerShell to Import Group Policy Objects

The Scripting Guys January 5, 2014

Summary: Microsoft PFE, Ian Farr talks about using Windows PowerShell to import Group Policy Objects.

Microsoft Scripting Guy, Ed Wilson, is here. Yesterday, guest blogger, Ian Farr talked about backing up Group Policy Objects (GPOs) in his post Using PowerShell to Back Up Group Policy Objects. He continues his GPO series today...

The challenge

Welcome back! Part 1 looked at my script, Comprehensive Group Policy Backup Script. Part 2 is about a Group Policy import script. Previously done, you say? True, but I have yet to find a script that mirrors all of the following configuration items to a test domain:

- Group Policy settings
- Delegation
- Security filtering
- Scope-of-management (SOM)
- Block inheritance
- Enforced
- Link enabled
- Link order
- WMI filters

Until now. This script can also process a Migration Table. Let's discuss it...

The import

The import script flow is determined by the backup script's output. What do we have? A folder with the normal **Backup-GPO** cmdlet results. In the same folder, there's an XML file that has the details of any WMI filters from the source domain. The **Export-Clixml** cmdlet created this file. We need to import it with the **Import-Clixml** cmdlet:

```
$WmiFilters = Import-Clixml -Path $WmiXML
```

$WmiFIlters has the configuration information for each WMI filter from the source domain.

We also have another XML file, GpoDetails.xml, with information for reconstructing GPO settings that were not captured by **Backup-GPO**. This is imported too:

```
$CustomGpoInfo = Import-Clixml -Path $CustomGpoXML
```

In **$CustomGpoInfo**, there's an object for each backed-up GPO. Each object has the following properties:

- **BackupGuid** : A unique GUID identifying the backed-up GPO.
- **Name**: The GPO name.
- **GpoGuid**: The GUID of the GPO in Active Directory.
- **SOMs**: Details of where the GPO is linked to (SOM) and link configuration.
- **DomainDN** : the distinguished name of the source domain.

For example:

Follow Us

Search MSDN with Bing

Search this blog

Search all blogs

Top Server & Tools Blogs

ScottGu's Blog

Brad Anderson's "In the Cloud" Blog

Brian Harry's Blog

Steve "Guggs" Guggenheimer's Blog

Subscribe

Recent Posts

- PowerTip: Turn off the power to your computer with PowerShell July 16, 2018
- Parse HTML and pass to Cognitive Services Text-to-Speech July 16, 2018
- PowerTip: Determine your version of PowerShell and host operating system June 29, 2018
- Windows PowerShell and the Text-to-Speech REST API (Part 5) June 29, 2018

Tags

Scripting Guy! | Windows PowerShell | scripting techniques | PowerTip | guest blogger | VBScript | getting started | Weekend Scripter | Sean Kearney

```
BackupGuid : ae9d0129-c9a2-46eb-951d-a129ec9d659e
Name       : Capture Default Printer
GpoGuid    : 277f16e7-8b0c-4ee1-86f7-8b6d548c9aa5
SOMs       : {OU=User Accounts,DC=ninja,DC=net:False:True:3:False}
DomainDN   : DC=ninja,DC=net
```

Let's look at the **SOMs** property in detail. Scope-of-management refers to a site, domain, or organizational unite where a GPO is linked. Each **SOM** object in the **SOMs** array has that information and more:

"$SomDN:$SomInheritance:$LinkEnabled:$LinkOrder:$LinkEnforced"

For example:

```
OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=ninja,DC=net:True:True:1:True
```

There are five pieces of information for each entry. Each piece is separated by a colon. Here's what they represent:

- **$SomDN**: The distinguished name of the SOM.
- **$SomInheritance**: Whether Block Inheritance is enabled on the SOM (True or False).
- **$LinkEnabled**: Whether the GPO link is enabled or disabled (True or False).
- **$LinkOrder**: The precedence of the GPO link (order number).
- **$LinkEnforced**: Whether the GPO link is enforced (True or False).

The backup folder might also contain a migration table to translate domain-specific information from one domain to another. This should be manually updated after the Backup_GPOs script has completed.

To process the backup output, the import script needs five sections:

1. Import WMI filters
2. Import backed-up GPOs
3. Link WMI filters
4. Create GPO links
5. Configure GPO links

Logging on

The script also records key actions. A script log shows script activity and aids troubleshooting. For years, I've logged script output in a format that can be parsed by the SMS Trace application (part of the Systems Management Server 2003 Toolkit 2, which is now superseded by CM Trace). This allows me to match events to script components, highlight key information, filter, search, and even look up error codes.

Here are some sample log entries:

Processing GP link updates for Deny Logon...	Update_Links	18/09/2013 21:22:19	1 (0x1)
GPO link updated for 93764eb8-3f64-4cae-aebe-5bc13ce825b9 at OU=UAE,OU=Lab Accounts,DC=halo,DC=net	Update_Links	18/09/2013 21:22:19	1 (0x1)
GPO link set to "Enabled: Yes" "Order: 1" "Enforced: No"	Update_Links	18/09/2013 21:22:19	1 (0x1)
GPO link updated for 93764eb8-3f64-4cae-aebe-5bc13ce825b9 at OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab	Update_Links	18/09/2013 21:22:20	1 (0x1)
GPO link set to "Enabled: Yes" "Order: 1" "ENFORCED: YES"	Update_Links	18/09/2013 21:22:20	1 (0x1)
BLOCK INHERITANCE set on OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=halo,DC=net	Update_Links	18/09/2013 21:22:20	1 (0x1)
OU=Matlock,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=halo,DC=net does not exist in target domain	Update_Links	18/09/2013 21:22:20	1 (0x1)

You can download the entire function from the Script Center Repository: Log-ScriptEvent Function.

Now, let's look at the script sections in detail...

Import WMI filters

If the **WmiFilter** script switch is specified, this section is executed. A loop processes each filter in the **$WmiFilters** array. The **DomainDN** portion of the current filter's **DistinguishedName** property is updated to reflect the target domain, and then saved to a variable (**$SourceDomainDN** and **$TargetDomainDN** have been defined earlier in the script).

```
ForEach ($WMI in $WmiFilters) {

$TargetWmiDN = $WMI.DistinguishedName –Replace
$SourceDomainDN, $TargetDomainDN
```

For example:

```
CN={4E39859E-F1C7-406E-8E08-58FC207644C9},CN=SOM,CN=WMIPolicy,CN=System,DC=ninja,DC=net
```

Is updated to:

```
CN={4E39859E-F1C7-406E-8E08-58FC207644C9},CN=SOM,CN=WMIPolicy,CN=System,DC=halo,DC=net
```

This variable is then supplied to **Get-ADObject** to test whether the filter

Office
Active Directory
operating system
storage
WMI
files
text files
community
desktop management
2011 Scripting Games
2012 Scripting Games

Related Resources
Script Center Home
Scripting Library
Learn to Script
Script Repository
Scripting Forum
2012 Scripting Games

Archives
July 2018 (2)
June 2018 (4)
May 2018 (5)
March 2018 (3)
February 2018 (4)
December 2017 (7)
October 2017 (2)
All of 2018 (18)
All of 2017 (30)
All of 2016 (244)
All of 2015 (726)
All of 2014 (740)
All of 2013 (757)
All of 2012 (624)
All of 2011 (442)
All of 2010 (432)
All of 2009 (285)
All of 2008 (181)
All of 2007 (243)
All of 2006 (244)
All of 2005 (248)
All of 2004 (100)

already exists in the target domain.

```
TargetWMI = Get-ADObject -Identity $TargetWmiDN
```

If the filter does exist, properties from the backup are added to a hash table with the **[Ordered] type** declaration. **[Ordered]** ensures that the properties stay in the defined order.

```
$Properties = [Ordered]@{
"msWMI-Author" = $WMI."msWMI-Author"
"msWMI-ChangeDate" = "$(Get-Date -Format yyyyMMddhhmmss).706000-000"
"msWMI-ID" = $WMI."msWMI-ID"
"msWMI-Name" = $WMI."msWMI-Name"
"msWMI-Parm1" = $WMI."msWMI-Parm1"
"msWMI-Parm2" = $WMI."msWMI-Parm2"
}  #End of $Properties
```

The **Set-ADObject** cmdlet is then used to update the filter with **$Properties** passed to the **Replace** parameter:

```
$UpdateWmiFilter = Set-ADObject -Identity $TargetWmiDN -Replace $Properties
```

If the filter doesn't exist, the hash table of properties is given an additional attribute: **"msWMI-CreationDate"**:

```
"msWMI-CreationDate" = "$(Get-Date -Format yyyyMMddhhmmss).706000-000"
```

The **New-ADObject** cmdlet is then called:

```
$NewWmiFilter = New-ADObject -Name $WMI."msWMI-ID" -Type $WMI.ObjectClass `
-Path "CN=SOM,CN=WMIPolicy,CN=System,$TargetDomainDN" `
-OtherAttributes $Properties
```

Here's an explanation of its parameters.

- **Name**: The AD name attribute of the new object.
- **Type**: The object class, for example, **msWMI-Som**.
- **Path**: Where to create the object in Active Directory (note the use of **$TargetDomainDN** in the string).
- **OtherAttributes**: Accepts a hash table of object attributes that are not covered by a cmdlet parameter.

When every WMI filter is imported, the parent loop is closed.

Import backed-up GPOs

A parent loop processes each object in the **$CustomGpoInfo** array:

```
ForEach ($CustomGpo in $CustomGpoInfo) {
```

The **Import-GPO** cmdlet is executed:

```
$ImportedGpo = Import-GPO -BackupId $CustomGpo.BackupGuid
-Path $BackupFolder
-CreateIfNeeded
-Domain $DomainFQDN
-TargetName $CustomGpo.Name
-MigrationTable $MigrationFile
```

The parameters:

- **BackupID**: The backup GUID property of the current object.
- **Path**: The backup folder (a script parameter).
- **CreateIfNeeded**: Creates a GPO if one doesn't exist.
- **Domain**: The domain into which the GPO is to be imported (a script parameter).
- **TargetName**: The name of the imported GPO, the name property of

the current object.
- **MigrationTable**: Included if the **MigTable** switch is specified at script execution. The migration table in the backup folder is referenced.

A backed-up GPO object is imported into the target domain. What about those additional GPO configuration settings?

Link WMI filters

Before the next GPO is processed, this section is executed only if the **WmiFilter** script switch is activated. If the current custom GPO object has a WMI filter property, the corresponding filter in the target domain is linked to the newly imported GPO.

Use the **WmiFilter** property of the current custom GPO to construct the path to the WMI filter in the target domain:

```
$TargetWmiDN =
"CN=$($CustomGpo.WmiFilter),CN=SOM,CN=WMIPolicy,CN=System,$TargetD
```

Test that the filter is present and include the **msWMI-Name** value in the properties that are returned:

```
$TargetWMI = Get-ADObject -Identity $TargetWmiDN -Property
"msWMI-Name"
```

If the filter is present, use the **Id** property of the imported GPO to construct the path to the corresponding Group Policy container in the target domain:

```
$TargetGpoDN = "CN=
{$($ImportedGpo.Id)},CN=Policies,CN=System,$TargetDomainDN"
```

What's a Group Policy container? A GPO is made up of two components. The first is the Group Policy container, which includes settings that are stored in Active Directory (there's an attribute for the linked WMI filter). The second is a Group Policy template, which has settings that are stored in the System Volume (SYSVOL). (This provides a file system share that is replicated to all domain controllers).

To finish, use the path to the Group Policy container to update the WMI filter attribute **gPCWQLFilter** on the Group Policy container object:

```
$UpdateGpoFilter = Set-ADObject $TargetGpoDN -Replace
@{gPCWQLFilter = "[$TargetDomainFQDN;$($TargetWMI.Name);0]"}
```

The **Replace** parameter accepts a hash table of attributes. In this instance, the value that is associated with the **gPCWQLFilter** attribute has an interesting format constructed by using the FQDN of the target domain and the name of the target WMI filter (the **msWMI-Name** value). Here's an example:

```
[halo.net;{25F58EA4-D2F5-49D9-A81C-51A610ED3E28};0]
```

Create GPO links

The information in this section is executed only if the **SomInfo** script switch is activated. If the current custom GPO object has a populated **SOMs** property, each **SOM** entry from the array is evaluated in a child loop. The aim is to link the imported GPO to corresponding **SOMs** in the target domain.

```
$SOMs = $CustomGpo | Select-Object -ExpandProperty SOMs
        ForEach ($SOM in $SOMs) {
```

To obtain the original SOM distinguished name, the **$SOM** string is split by using the "**:**" delimiter, and the first element of the array is assigned to a variable:

```
$SomDN = ($SOM –Split ":")[0]
```

For example:

```
OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=ninja,DC=net
```

The **DomainDN** of the string is then replaced with the distinguished name of the target domain:

```
$SomDN = $SomDN –Replace $CustomGpo.DomainDN, $DomainDN
```

For example:

```
OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=halo,DC=net
```

Next, check that the updated **SOMDN** exists by using the **Get-ADObject** cmdlet:

```
$TargetSom = Get-ADObject -Identity $SomDn
```

Note It is assumed that a matching path already exists in the target test domain.

If the distinguished name doesn't exist, an error is logged, and we move to the next **SOM**. If the distinguished name exists, create a new link to it by using the **New-GPLink** cmdlet:

```
$SomLink = New-GPLink -Guid $ImportedGpo.Id -Domain
$DomainFQDN -Target $SomDN
```

Remember **$ImportedGPO**? It contains the result of the **Import-GPO** cmdlet. Its **Id** property is the GPO GUID of the imported policy in the target domain.

Onward…

Each **SOM** is processed and the **SOM** child loop is exited. The current custom GPO from the **$CustomGpoInfo** array is updated. The GPO GUID of the newly imported policy is added as a property called **NewGpoGuid**:

```
$CustomGpo | Add-Member -MemberType NoteProperty -Name
NewGpoGuid -Value $ImportedGpo.Id
```

For example:

```
BackupGuid : 51c55b48-5810-4e7f-88ea-2423c3935fac
Name       : Deploy New York Printer
GpoGuid    : fa04fd62-79a8-4dee-90f0-3870838b44f7
SOMs       :
DomainDN   : DC=ninja,DC=net
NewGpoGuid : 66f601c3-1a2d-428c-8180-360e5da171a1
```

The process is repeated for each custom GPO object. The parent loop is then exited.

Configure GPO links

Time to process the block inheritance, link enabled, link order, and link enforced details. As in the *Import Backed-Up GPOs* section, a parent loop processes each object in the **$CustomGpoInfo** array. The **SOMs** property is expanded, and a child loop processes each entry.

Again, the **$SomDN** value is obtained by splitting the **$SOM** string. This time, though, we're also interested in the other elements, for example:

```
OU=Belper,OU=Derbyshire,OU=United Kingdom,OU=Lab Accounts,DC=ninja,DC=net:True:True:1:True
```

The **Set-GPLink** cmdlet doesn't understand Boolean values, so these have to be converted to **Yes** or **No**. Here's how the **$LinkEnabled** value is processed. The **$SOM** string is split at the "**:**", and the second index of the array is tested for **$True** or **$False**:

```
Switch ($SOM.Split(":")[2]) {

$True {

#Set the GP link enabled variable to Yes

$LinkEnabled = "Yes"

}  #End of $True

$False {

#Set the GP link enabled variable to No

$LinkEnabled = "No"

}  #End of $False

}  #End of Switch ($SOM.Split(":")[2])
```

Note The **Switch** statement is very powerful. It accepts wildcard characters, regular expressions, or the content of a file as input. It can even loop. Have a look at Get-Help about_Switch.

The **$LinkEnforced** value is obtained by using the same **Switch** statement. This time, the fourth index of the array is used. **$LinkOrder** is grabbed from the third index.

Here's what the **Set-GPLink** syntax looks like:

```
$SomLink = Set-GPLink -Guid $CustomGpo.NewGpoGuid
 -Domain $DomainFQDN
 -Target $SomDN
 -LinkEnabled $LinkEnabled
 -Order $LinkOrder
 -Enforced $LinkEnforced
```

The value that is fed to the **Guid** parameter is the **NewGpoGuid** property that was recently added to the current custom GPO object. After executing **Set-GPLink**, almost all of the additional data is restored.

Here's the last piece of the puzzle. The **Set-GPInheritance** cmlet is only called to configure block inheritance when the corresponding value is **True**:

```
If ($SomInheritance -eq $True) {
#Set block inheritance
$SetInheritance = Set-GPInheritance -Target $SomDn -IsBlocked Yes
```

Note In PFE, we don't recommend the use of the enforced or the block inheritance options because they can complicate troubleshooting. If the script enables either of these options, a warning will be logged (highlighted in yellow in the following example):

```
GPO link set to "Enabled: Yes" "Order: 1" "Enforced: No"
BLOCK INHERITANCE set on OU=Costa Rica,OU=Lab Accounts,DC=halo,DC=net
GPO link updated for 3685849c-810c-43ba-aa90-da8dd1f3f541 at OU=France,OU=Lab Accounts,DC=halo,DC=net
```

Then, on to the next **SOM** entry until all link details are processed. Then, on to the next GPO import until all backed-up GPOs are processed.

Now inspect the script log. It will be in the directory from which the import script was executed. Target any red. Finally, fire up the GPMC for the test domain…Cool, eh?

Well, I enjoyed that and I hope you did too. Have fun!

You can download the entire script from the Script Center Repository: Comprehensive Group Policy Import Script.

~Ian

Thank you, Ian. Join me tomorrow as the Scripting Wife reveals the top five Hey, Scripting Guy! Blog posts of 2013. It is cool stuff—you will not want to miss it.

I invite you to follow me on Twitter and Facebook. If you have any questions, send email to me at scripter@microsoft.com, or post your questions on the Official Scripting Guys Forum. See you tomorrow. Until then, peace.

Ed Wilson, Microsoft Scripting Guy

Tags Group Policy guest blogger Ian Farr Scripting Guy! Weekend Scripter Windows PowerShell

Comments (4)

Name *

Email *

Website

Post Comment

Ian Farr [MSFT]
July 24, 2018 at 8:12 am
The script now processes gPLink info on site objects – BIG thanks to Mark Renoden! I've a number of other suggestions from the community to implement… watch this space.

Reply

Ian Farr [MSFT]
July 24, 2018 at 8:12 am
Here's a couple of companion scripts to ensure we have a matching OU structure in the test environment:

Dump AD OU Structure Script
https://gallery.technet.microsoft.com/scriptcenter/Dump-Active-Directory-OU-345e54d2

Mirror AD OU Structure Script
https://gallery.technet.microsoft.com/scriptcenter/Mirror-AD-OU-Structure-d9acfa6e

Reply

Jonathan
February 4, 2014 at 6:32 pm
Here is the error I'm getting: Get-ADObject : Directory object not found At D:scriptsImport_GPOs.ps1:523 char:46 + $TargetWMI = Get-ADObject <<<< -Identity $TargetWmiDN -Property "msWMI-Name" -Server $TargetPDC

Reply

jsmith
April 21, 2016 at 8:27 pm
Is it possible to modify the script so that it can restore just one GPO? Of course I can us the GPMC to selectively restore a single GPO but it misses all the extra data (i.e. links). Thanks.

Reply