

Automating Group Policy Management

*Using SDM Software solutions to manage your Group Policy Infrastructure with
PowerShell*



Written by **Darren Mar-Elia**
CTO & Founder
SDM Software, Inc.
www.sdmsoftware.com

March, 2010

Table of Contents

Table of Contents	2
Introduction	3
Creating and Deleting GPOs.....	4
Linking and Un-Linking GPOs	5
Getting GPO Information	7
Modifying GPO Settings	8
Modifying Administrative Template Policy.....	10
Example: Automating Windows Firewall Exception lists	12
Modifying Security Policy.....	12
Modifying Drive Mapping Policy.....	13
Writing Settings with GPAE – A Summary	14
Reading Settings.....	15
Call to Action	16
To Evaluate.....	16

Introduction

With the advent of Microsoft's PowerShell scripting environment, new opportunities have arisen for the automation and command-line management of Windows infrastructures. Windows' Group Policy technology is no exception here. And, with SDM Software's **GPEXpert® Group Policy Automation Engine (GPAE)**, you can now automate the management of settings within Group Policy Objects (GPOs) in addition to simply managing the creation and linking of GPOs.

In this whitepaper, we'll look at how you can use Microsoft's new PowerShell scripting environment, **SDM Software's [free GPMC cmdlets](#)** and the **[GPEXpert™ Group Policy Automation Engine](#)** to automate the management of your Group Policy environment. Each offering provides a different set of functionality to enable Group Policy management using PowerShell (***Note that the current version of the GPAE supports both PowerShell v1 and v2***). The GPMC cmdlets, for example, are designed to wrap the functionality provided by GPMC's scripting APIs. These APIs typically operate at the "whole-GPO" level (i.e. they let you create and delete GPOs rather than edit the settings within them). This is where the **GPEXpert® Group Policy Automation Engine** comes into play. The GPAE provides a scripting model for modifying the settings **within** GPOs. The combination of both tools makes it possible to automate almost all tasks related to Group Policy management. Let's take a look at how some of these tasks can be automated using PowerShell.

The first thing we want to do is list out all of the cmdlets available in the free GPMC cmdlets from SDM Software. It's easy to do that using PowerShell's **get-command** cmdlet. **Figure 1** shows all the GPMC cmdlets available in version 1.3 –25 to be exact, as of this writing.

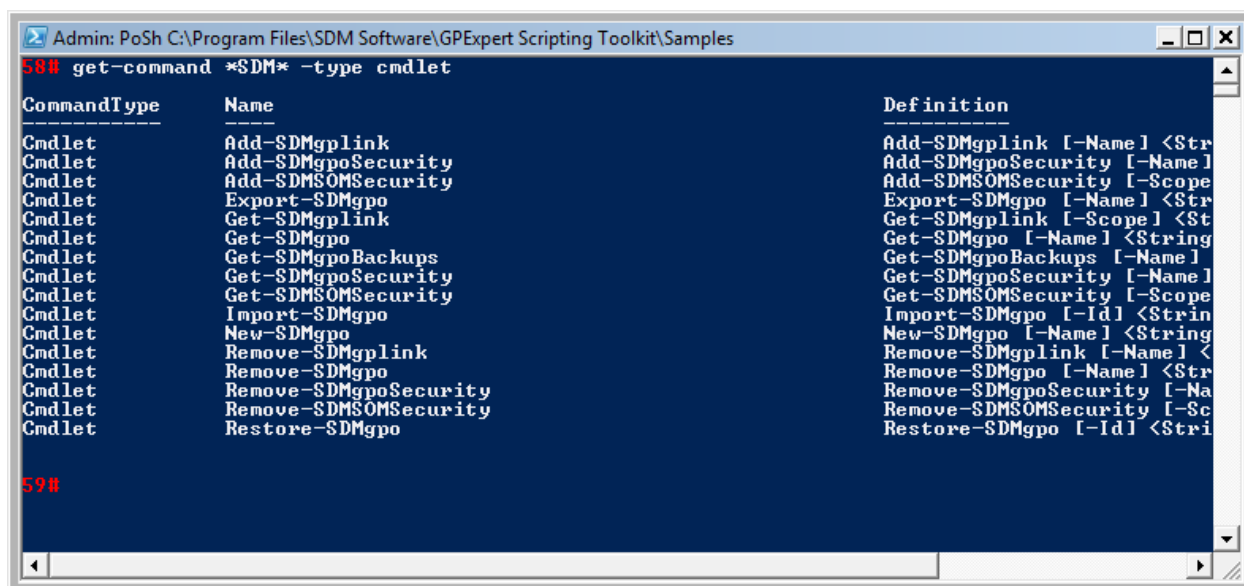


Figure 1: Viewing all of the GPMC cmdlets

Creating and Deleting GPOs

Using the free SDM GPMC cmdlets, creating and deleting GPOs is very simple. To create a GPO from PowerShell, you'll use the **new-SDMgpo** cmdlet as follows:

```
> New-SDMgpo "HR Lockdown Policy"
```

```

DisplayName           : HR Lockdown Policy
Path                  : cn={715E4C2F-4787-457D-AB5E-
3FD1B20DD2BE},cn=policies,cn=system,DC=cp
                        andl,DC=com
ID                    : {715E4C2F-4787-457D-AB5E-3FD1B20DD2BE}
DomainName            : cpandl.com
CreationTime           : 1/9/2008 9:11:04 AM
ModificationTime       : 1/9/2008 9:11:04 AM
UserDSVersionNumber    : 0
ComputerDSVersionNumber : 0
UserSysvolVersionNumber : 0
ComputerSysvolVersionNumber : 0

```

As the command above shows, we've created a new GPO called "HR Lockdown Policy". When you issue the command in PowerShell, it returns information about the newly created GPO, as shown above. The one assumption being made, but not explicitly spelled out here, is that this cmdlet is creating a GPO in the current domain. There is a "DomainName" parameter available for all of the GPMC cmdlets that let you operate against domains other than the one you are running the cmdlet from.

Now that we've created a GPO, let's look at how we can delete one. It is a very straightforward task to perform using the **remove-sdmgpo** GPMC cmdlet. The syntax is as follows:

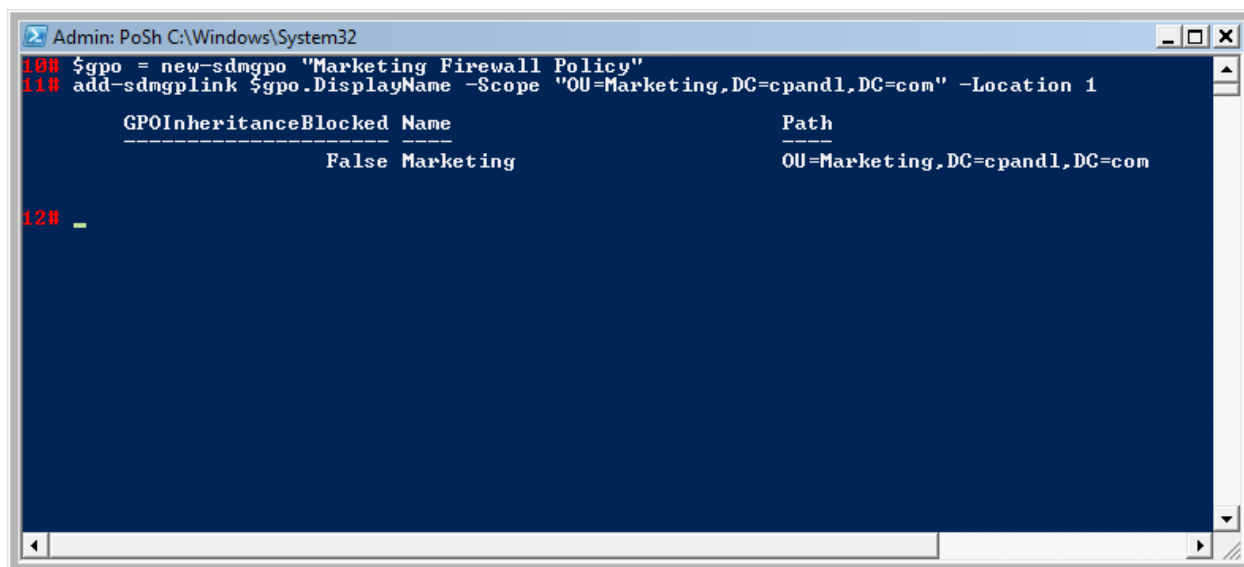
```
Remove-SDMgpo "HR Lockdown Policy"
```

If the GPO deletes successfully, you will receive no message—otherwise an error message is returned if the deletion operation fails.

Now let's look at how you can link and unlink GPOs from AD containers.

Linking and Un-Linking GPOs

The GPMC cmdlets also support linking and un-linking GPO to and from Active Directory sites, domains and OUs. The **Add-SDMgplink** and **Remove-SDMgplink** cmdlets serve this purpose. Let's look at how they can be used. Suppose you want to create and then link a GPO all in one fell swoop. It's easy to write a script to do just that. Let's call our new GPO "Marketing Firewall Policy" and let's link it to the Marketing OU. The following two commands, shown in **Figure 2**, will do just that.



```
Admin: PoSh C:\Windows\System32
10# $gpo = new-sdmgpo "Marketing Firewall Policy"
11# add-sdmgplink $gpo.DisplayName -Scope "OU=Marketing,DC=cpan1,DC=com" -Location 1

      GPOInheritanceBlocked Name                Path
      -----
      False Marketing                OU=Marketing,DC=cpan1,DC=com

12#
```

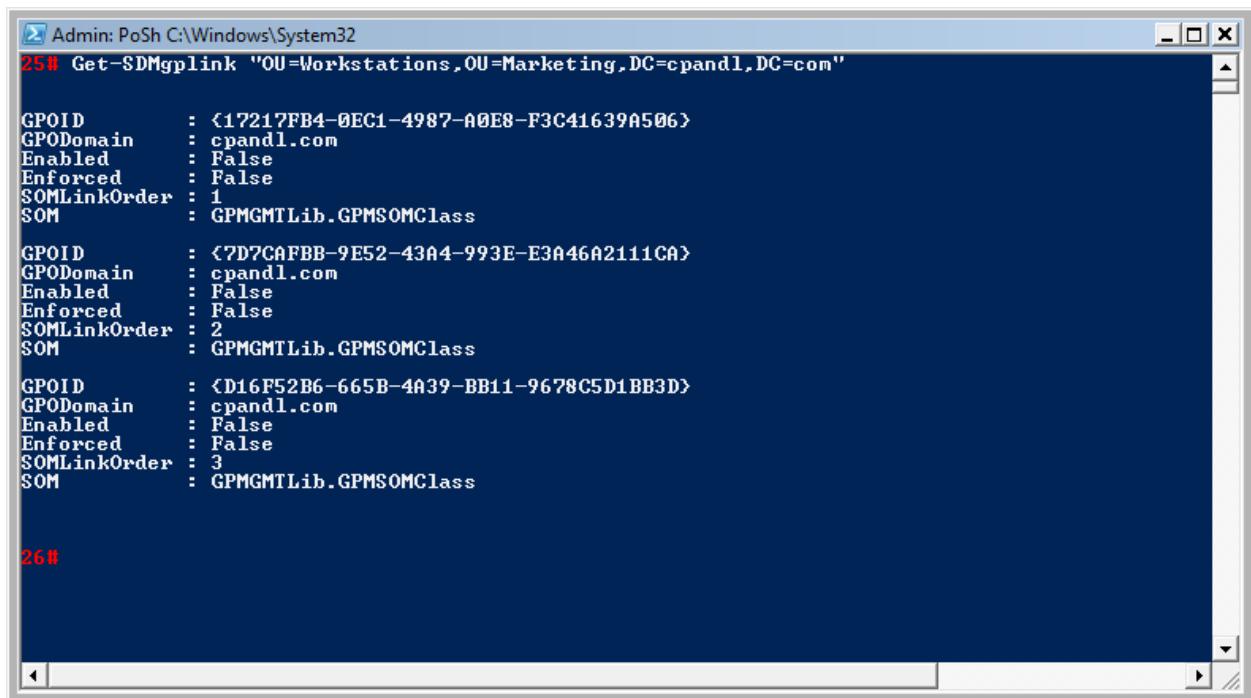
Figure 2: Creating and Linking a New GPO.

Note that in this figure, we call the **new-SDMgpo** cmdlet to create the new GPO and assign it to a variable called *\$gpo*. Next, we call **add-SDMgplink** and pass in the name of the GPO we just created (*\$gpo.DisplayName*) along with the distinguished name (OU=Marketing, DC=cpan1,DC=com) of the AD container we wish to link to. Finally, the Location parameter tells the cmdlet where to put the link, if there is more than one link on that container. In this case, choosing 1 puts the link at the top of the list on the Marketing OU. The output we see is the **add-SDMgplink** cmdlet reporting back the link that it just created.

We can just as easily unlink this GPO using the **remove-SDMgplink** cmdlet, using the following format:

```
Remove-SDMgplink "Marketing Firewall Policy" -Scope "OU=Marketing,DC=cpandl,DC=com"
```

In addition to the two cmdlets we mentioned above, you can also retrieve information about links using the **get-SDMgplink** cmdlet. This cmdlet lets you retrieve information about a particular AD container with respect to its GP links. For example, the following figure shows what happens when we issue this command against an OU with multiple GPO links:



```
Admin: PoSh C:\Windows\System32
25# Get-SDMgplink "OU=Workstations,OU=Marketing,DC=cpandl,DC=com"

GPOID      : <17217FB4-0EC1-4987-A0E8-F3C41639A506>
GPODomain  : cpandl.com
Enabled    : False
Enforced   : False
SOMLinkOrder : 1
SOM        : GPMGMTLib.GPMSOMClass

GPOID      : <7D7CAFBFB-9E52-43A4-993E-E3A46A2111CA>
GPODomain  : cpandl.com
Enabled    : False
Enforced   : False
SOMLinkOrder : 2
SOM        : GPMGMTLib.GPMSOMClass

GPOID      : <D16F52B6-665B-4A39-BB11-9678C5D1BB3D>
GPODomain  : cpandl.com
Enabled    : False
Enforced   : False
SOMLinkOrder : 3
SOM        : GPMGMTLib.GPMSOMClass

26#
```

Figure 3: Viewing GPO links on an OU

In this figure, we are using **get-SDMgplink** to retrieve the list of GPOs linked to the Workstations OU, within the Marketing OU.

We can also get “fancy” with this cmdlet. For example, let’s suppose I wanted to set the “Enabled” property on a particular GPO link on a particular AD container to false—i.e. we want to set the link to not enabled. We can use this cmdlet and some PowerShell loop statements to iterate through all the links on the container, and when we find the one we want, we can set its Enabled property to false. The following script section does just that:

```
foreach ($link in (get-sdmgplink -Scope "dc=cpandl,dc=com")){if ($link.GPOID
-eq "{68FCB643-A3B7-45D1-BA23-3D4D23BEA7A2}"){$link.Enabled = $false}}
```

So, what’s going on here? Well, in this command, we’re using the Foreach-Object cmdlet (whose alias is ‘foreach’) to iterate all the objects returned from a call to **get-SDMgplink** on the domain container. We assign each of those links to the \$link variable, and pass that to the ‘if’ statement that tests whether the

GUID (GPOID) of that link equals a particular GPO GUID that we know. If it is does, then we set the Enabled property on \$link to False (\$false) and we've just disabled this GPO's link on the domain!

Getting GPO Information

We've shown you some ways you can create and link GPOs. Now how about we get some information about existing GPOs? You can do that using the **get-SDMgpo** cmdlet. This cmdlet can take two different inputs. We can either pass it the name of the GPO we wish to retrieve information on, or we can pass it '*', as in:

```
Get-SDMgpo *
```

In this case, the cmdlet returns information about all GPOs in the current domain. But let's stick with the simple case. When we issue the following command:

```
Get-SDMgpo "Default Domain Policy"
```

We get the following output:

```
DisplayName           : Default Domain Policy
Path                  : cn={31B2F340-016D-11D2-945F-00C04FB984F9},cn=policies,cn=system,DC=cpandl,DC=com
ID                    : {31B2F340-016D-11D2-945F-00C04FB984F9}
DomainName            : cpandl.com
CreationTime          : 11/11/2004 4:20:24 PM
ModificationTime      : 12/31/2007 11:01:54 AM
UserDSVersionNumber   : 10
ComputerDSVersionNumber : 36
UserSysvolVersionNumber : 10
ComputerSysvolVersionNumber : 36
```

This returns some useful information about the GPO, such as its GUID (ID property), when it was created and when it was last modified, as well as version information. But let's take it a step further. We can also call methods on the output of this cmdlet. For example, what if we wanted to output the settings of this GPO to an HTML or XML file? Simple-- we call the `GenerateReportToFile()` method on this cmdlet and we have what we're after, as follows:

```
$gpo = get-SDMgpo "Default Domain Policy"
$gpo.GenerateReportToFile(1,"c:\report\ddp.html")
```

So in this two line script, we've got a reference to the Default Domain Policy using **get-SDMgpo** and then we've called the `GenerateReportToFile()` method on it, passing in the first parameter, which tells the method that we want an HTML report (use 0 for XML) and the second parameter, which tells it what file to store the report in.

There are lots of other things we can do with this cmdlet as well. Some of them, like backing up and importing a GPO, are also supported explicitly within other SDM Software GPMC cmdlets. But just for a sample, type the following command after having assigned `$gpo` to a GPO.

```
$gpo | get-member
```

This command will return a list of all methods and properties available to us when we use this cmdlet to get a reference to a GPO.

The GPMC cmdlets have quite a bit of other functionality as well, that is outside of the scope of this whitepaper, but we encourage everyone to check them out and see how they can be useful in your environment.

Now let's shift gears a little bit and show how we can use the **GPEXpert® Group Policy Automation Engine** from **SDM Software** to automate the modification of settings **within** GPOs!

Modifying GPO Settings

The **GPAE** provides a method of using PowerShell to access GP settings within most of the policy areas provided by Microsoft, including Administrative Templates, Security, Software Installation, Folder Redirection, Group Policy Preferences and more. But, what are the scenarios where you'd want to use the GPAE?

The GPAE is designed to automate any repetitive Group Policy operations. In addition, because we can capture changes to Group Policy settings within scripts, we can create and test those scripts offline against test GPOs and then implement them at the time of our choosing, either manually by running the script or in an automated fashion using a utility like the Windows Task Scheduler. The GPAE also lets us read settings from GPOs, which lets us verify the settings that exist in a given GPO.

So, let's get started looking at how the GPAE works. The first thing to note is that the GPAE can operate against both local and AD-based GPOs. So, if we have machines that are not in an AD domain or we just need to modify the local GPO, the GPAE can handle it. The first thing we need to do when using the GPAE, is to get a reference to a GPO. We can do that simply by calling the **get-SDMGPOObject** cmdlet from the GPAE and passing it a reference to the GPO we wish to access. In the following figure, we're actually calling the cmdlet with no parameters, which gets a reference to the local GPO on the machine where the GPAE is installed.

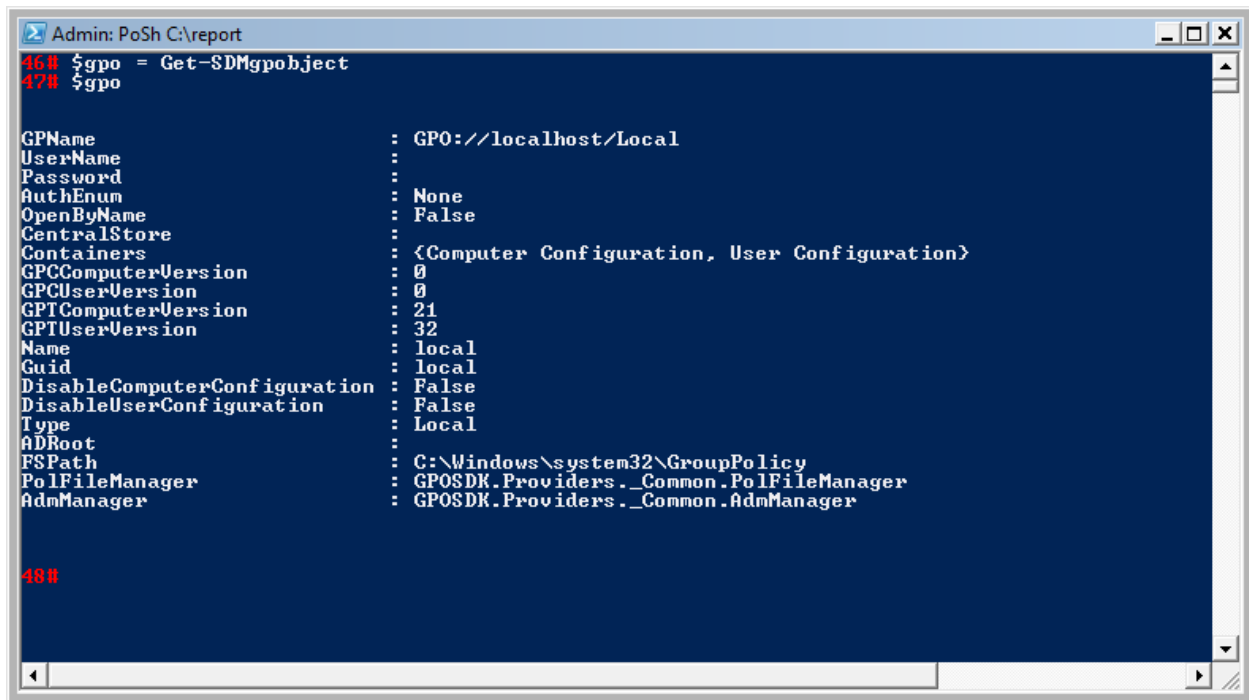


Figure 4: Getting a reference to the local GPO.

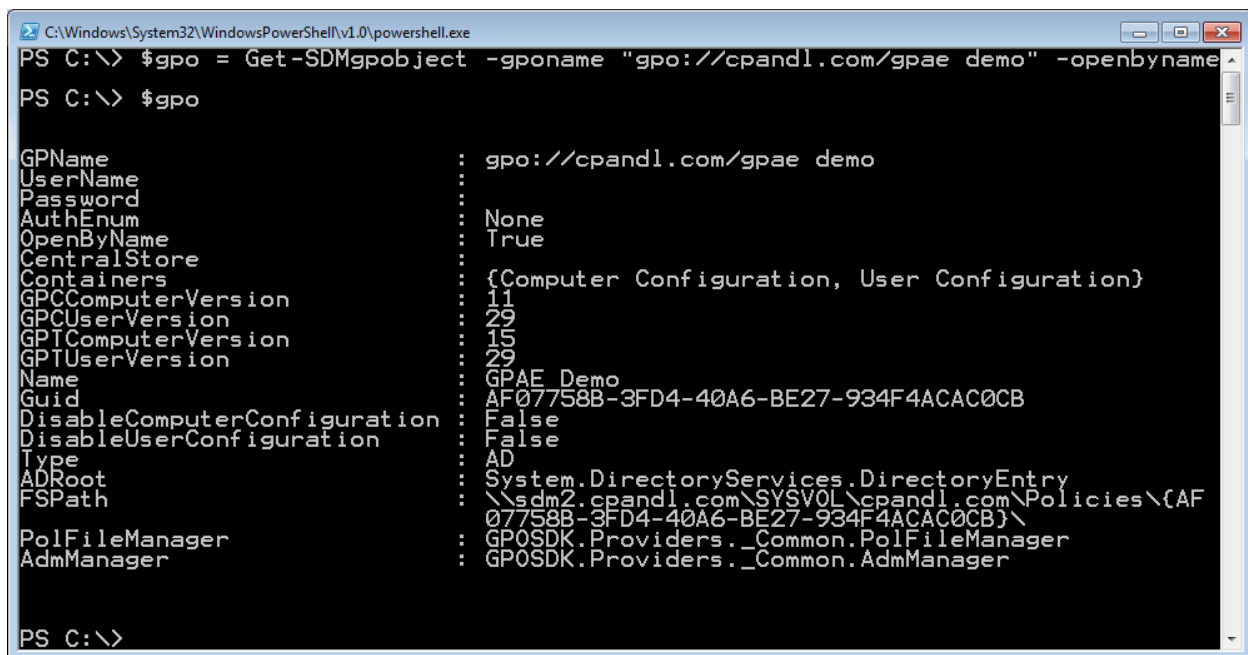
Note that after issuing the command `$gpo = get-sdmgobject`, we asked PowerShell to tell us what `$gpo` holds. As you can see, it contains some general information about the GPO as well as some information specific to the GPAE.

Now what if we want to access the local GPO on a remote machine? Simple; we just alter the command above as follows:

```
$gpo = get-sdmgobject -gpoName "gpo://xp3/Local"
```

This command does something slightly new. It uses the `gponame` parameter to tell the `get-sdmgobject` cmdlet which GPO to talk to. In this case, it uses a convention very similar to Active Directory scripting, but instead of saying "LDAP://..." it uses "GPO://...". In this example, we are connecting to the local GPO on a remote XP workstation called `XP3`.

Well that's all well and good, but what is really interesting is when we connect to AD-based GPOs, as we do in Figure 5.



```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
PS C:\> $gpo = Get-SDMgobject -gponame "gpo://cpandl.com/gpae demo" -openbyname
PS C:\> $gpo

GPName           : gpo://cpandl.com/gpae demo
UserName         :
Password         :
AuthEnum         : None
OpenByName       : True
CentralStore     :
Containers       : {Computer Configuration, User Configuration}
GPCComputerVersion : 11
GPCUserVersion   : 29
GPTComputerVersion : 15
GPTUserVersion   : 29
Name             : GPAE Demo
Guid             : AF07758B-3FD4-40A6-BE27-934F4ACAC0CB
DisableComputerConfiguration : False
DisableUserConfiguration : False
Type             : AD
ADRoot           : System.DirectoryServices.DirectoryEntry
FSPath           : \\sdm2.cpandl.com\SYSTEM\cpandl.com\Policies\{AF07758B-3FD4-40A6-BE27-934F4ACAC0CB}
PolFileManager   : GP0SDK.Providers._Common.PolFileManager
AdmManager       : GP0SDK.Providers._Common.AdmManager

PS C:\>
```

Figure 5: Connecting to an AD-based GPO

Note that this command takes a slightly different form than the previous one. In the previous command, we connected to a machine name (XP3) and the local GPO. In this command, we are connecting to the “GPAE Demo” GPO on the **cpandl.com** domain. We are also using the *–openbyname* parameter because the default for the GPAE is to expect a GPO GUID rather than its friendly name. What you also see in Figure 5 is that when we print out the contents of `$gpo`, it contains some information that looks very similar to what our free GPMC cmdlet **get-sdmgpo** returned. However, the similarities stop there. Once we have this reference to an AD-based GPO, we are ready to **start making changes** to the GPO!

Modifying Administrative Template Policy

The first thing we need to do, once we have the GPO reference, is to get a reference to the setting we wish to change. Let’s look at modifying a simple Administrative Template policy that has a state of Enabled, Disabled or Not Configured (its default state). The following command uses the reference to the GPO we created in Figure 5 to connect to an Administrative Template setting that controls whether Windows computers use Offline Files or not.

```
$setting = $gpo.GetObject("Computer Configuration/Administrative
Templates/Network/Offline Files/Allow or Disallow use of the Offline Files
feature")
```

Note that this command is relatively straightforward. We are calling the *GetObject* method on the `$gpo` object we created earlier. The parameter inside that call is the full path, as we would see it in Group Policy Editor, to the policy we want to modify. Note that the GPAE also lets you refer to Administrative Template paths using their language-independent “tags” as are found in the ADM or ADMX files that underlie these policy settings.

After we have assigned `$setting` to the policy we wish to modify; now we need to modify the policy. We do that using the following two commands:

```
$setting.Put("State", [GPOSDK.AdmTempSettingState]"Enabled")  
  
$setting.Save()
```

What the first of these two commands does is call the *Put* method on the `$setting` object. That method takes two parameters—the name of the property on the policy we wish to modify—in this case the “State” property (state corresponds to the Enabled, Disabled or Not Configured state of the policy item) and the value we want to assign to that property. In this case, we see a strange looking declaration of `[GPOSDK.AdmTempSettingState]`, which is specific to the GPAE, and indicates that the string “Enabled” has a special value that is specific to Administrative Template policies. The second command, that calls `Save()`, actually commits the enabled policy change to the GPO. Until that `Save()` command runs, the GPO is not yet modified.

This example we just looked at is specific to modifying Administrative Template policy. The GPAE supports Administrative Template policies that are more complex than just simple Enabled, Disabled or Not Configured states. For example, the following listing modifies the Windows Update policies in a given GPO, which use dropdown boxes in Group Policy Editor to allow the administrator to choose options for when and how often Windows Update runs.

```
$gpo = Get-SDMgpobject -gpoName "gpo://cpandl.com/Default Domain Policy" -  
openByName $true;  
  
$stng = $gpo.GetObject("Computer Configuration/Administrative  
Templates/Windows Components/Windows Update/Configure Automatic Updates");  
  
$stng.Put("State", [GPOSDK.AdmTempSettingState]"Enabled"); #this step enables  
the setting—this is universal for all Admin Templates  
  
$stng.Put("Configure automatic updating:",4); #This chooses the first option  
below.  
  
$stng.Put("Scheduled install day: ",5); #and the second option  
  
$stng.Put("Scheduled install time:",20); #and the 3rd option  
  
$stng.Save();
```

Listing 1: Modifying Administrative Template policy settings that have multiple “Parts”

In this listing, the first three commands look very familiar from our previous example. However, the next three are more interesting. They are essentially configuring the additional options that are available for this policy, beyond just enabling it. While they are specific to this particular policy, the GPAE does allow you to interrogate a policy setting to determine exactly what are the policies that are available and what their possible values are (in the case of policies that have pre-defined values).

Example: Automating Windows Firewall Exception lists

Another example of the GPAE's strengths are its ability to take Group Policy editing tasks that are complex and cumbersome using the Group Policy Editor, and make them simple. An example of this is the setting of Windows Firewall exceptions. The following listing shows how you can use the GPAE to script multiple firewall exceptions at once in a script that can be easily modified as needed to add or remove additional exceptions. Contrast that to the process in place within the Group Policy Editor and you quickly see how the GPAE can save a lot of time and reduce the error of Group Policy management.

```
$gpo = get-sdmgpoobject -gponame "gpo://cpandl.com/Default Domain Policy" -
openbyname $true;

$st = $gpo.GetObject("Computer Configuration/Administrative
Templates/Network/Network Connections/Windows Firewall/Standard
Profile/Windows Firewall: Define port exceptions");

$st.Put("State", [GPOSDK.AdmTempSettingState]"Enabled");

$st.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND", "Define port
exceptions:", "80:TCP:*.Enabled:HTTP");

$st.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND", "Define port
exceptions:", "23:TCP:*.Enabled:Telnet");

$st.Save();
```

Listing 2: Adding Windows Firewall exceptions using the GPAE

In this example, we are adding two new port exceptions to the Default Domain Policy. As you can see, similar to Listing 1, we are doing more than just enabling the policy. Note that this particular type of policy setting uses the *PutEx* method instead of a simple *Put*. The GPAE's documentation describes the differences, but basically, because we are modifying a more complex setting type, we need to use this more advanced method.

Modifying Security Policy

Of course, the GPAE can modify more than just Administrative Template policy. Let's say we want to modify security policy to enable System Event auditing within a given GPO. No problem! Listing 3 shows exactly how that would work.

```

$gpo = get-sdmgobject -gponame "gpo://cpandl.com/Default Domain Policy" -
openbyname $true;

$secSetting = $gpo.GetObject("Computer Configuration/Windows
Settings/Security Settings/Local Policies/Audit Policy")

$settingName = "Audit system events"

$setValue = [GPOSDK.AuditPolicyValue]"Success"

$setting = $secSetting.Settings.ItemByName($settingName)

$setting.Put("Defined",1)

$setting.Put("Value",$setValue)

$setting.Save()

```

Listing 3: Modify security audit policy using the GPAE.

In the example in Listing 3 above, we are doing something slightly different from our previous examples with Administrative Templates. Because security settings don't use the simple Enabled, Disabled and Not Configured states, we need some different scripting to accommodate this. The first two lines are familiar—we are getting a reference to a GPO and then getting a reference to a policy path—in this case, we are referencing the Audit Policy container within the Group Policy Editor namespace. The next three lines set up the policy we want to modify—in this case the “Audit System Events” policy. This policy item is stored in `$settingName`. The next line, which assigns `$setValue`, stores what type of auditing we want to enable—in this case “Success” auditing using another special GPAE enumeration called `[GPOSDK.AuditPolicyValue]`. Next, we assign `$setting` to the actual policy item by using the *ItemByName* method. This is necessary because we can't just get a reference directly into the Audit System Events setting, due to the way its defined in the GPO. Once we've got a reference to the setting we want, the `$setting.Put("Defined",1)` setting is equivalent to enabling the policy, the next command sets the type of auditing we want to “Success” and finally, we save the settings.

Modifying Drive Mapping Policy

Another powerful example of using GPAE involves managing drive mappings using **Group Policy Preferences** and GPAE.

In the example in Listing 4 below, we use GPAE to create a Group Policy Preferences **Drive Mapping** policy to map a “P:” drive for all users of the “Marketing Users” AD security group, using the Item-Level Targeting feature in GPP, which is supported in GPAE.

```

$gpo = Get-SDMgobject -gpoName "gpo://cpandl.com/GPAE Demo" -openByName

$container = $gpo.GetObject("User Configuration/Preferences/Windows
settings/Drive Maps")

```

```

$map = $container.Settings.AddNew("P Drive")
$map.Put("Action",[GPOSDK.EAction]"Replace")
$map.Put("Drive Letter","P")
$map.Put("Location","\\sdml\public")
$map.put("Reconnect", $true);
$map.put("Remove this item when it is no longer applied", $true);
$map.Put("Order",1);
$map.Put("Label as", "SDM Public Drive");
$map.Save()

# Now use Item-Level Targeting to have the drive mapping apply only to a
particular user group
$iilt = $gpo.CreateILTargetingList()
$itm =
$iilt.CreateIILTargeting([GPOSDK.Providers.ILTargetingType]"FilterGroup")
$itm.Put("Group","Marketing Users")
$itm.Put("UserInGroup", $true)
$iilt.Add($itm)

# Now save filters to current map setting
$map.put("Item-level targeting", $iilt)

$map.Save()

```

The power of GPAE is that you can use this method to set, change or remove GPP settings for any of the different policy areas supported AND, you can leverage item-level targeting to change how the policy settings applied.

Writing Settings with GPAE – A Summary

So you see, while each policy area is handled slightly differently, the general approach for using GPAE is the same:

1. Get a reference to the GPO.
2. Get a reference to the policy setting.
3. Enable (or disable) the policy setting.
4. Modify any additional properties required by the setting.
5. Save the setting.

This model holds true regardless of whether we are setting Windows Firewall exceptions or deploying a new Software Installation package using the GPAE.

Reading Settings

Another example of using the GPAE in a slightly different way is in leveraging its ability to read settings within a GPO. For example, suppose you wanted to verify a particular setting across multiple GPOs within your environment? This is where the GPAE and PowerShell come in handy. The following Listing 4 shows an example of a script that does just this:

```
$gpos = import-csv gpos.txt;

foreach ($mygpo in $gpos)
{
    $path = "gpo://cpandl.com/" + $mygpo.Name

    $gpo = Get-SDMgpoobject -gpoName $path -openbyname $true;

    $container = $gpo.getObject("Computer Configuration/Administrative
Templates/System/Logon");

    $settingName = "Always wait for the network at computer startup and logon";

    $setting = $container.Settings.ItemByName($settingName);

    if ($setting.Get("State") -eq -1)
    {
        $mygpo.Name + " does not have setting configured";
    }
    else
    {
        $mygpo.Name + " has setting set to state of: " + $setting.Get("State");
    }
}
```

Listing 4: Iterating through a list of GPOs to verify a policy setting.

In the script in Listing 4, we are using the PowerShell built-in cmdlet called **import-csv** to read a list of GPO names from a CSV file called gpos.txt. This file is a simple single-column listing of GPOs with a header called “Name”. We iterate through this file, and for each GPO found, we enter into the foreach loop and go looking for our setting. Nothing in this script should look new except for the use of the *Get* method instead of the *Put* method. In this case, we are using *Get* to get the value of the “Always wait for

the network..." setting under the path "Computer Configuration/Administrative Templates/System/Logon" within each GPO. For each GPO, if the property of the setting called "State" is set to Not Configured (-1), then we print that out to the console. If the state is any other value that indicates the policy has been set (i.e., to enabled or disabled) then we print out that state.

You can already begin to see the power of being able to do these kinds of queries against multiple GPOs and multiple settings. You can use this capability to first search for a policy value, and then based on the results, modify that policy value. This can provide a powerful 'search and replace' function that heretofore was impossible in Group Policy management.

Beyond that, you could even do things like reading the enabled settings out of one GPO, and then enabling those same settings in a different GPO, effectively being able to extract some settings from GPO A and apply them to GPO B! The key is that the combination of the free **GPMC cmdlets** and the **GPAE** give you a complete solution for managing Group Policy.

Call to Action

Let's review what SDM Software's Group Policy PowerShell solutions provide. With the free GPMC cmdlets, you can create and delete GPOs, link and unlink GPOs, permission GPOs and Scopes of Management (SOMs) and get information on GPOs. You can also backup, restore and import GPOs. All of these capabilities operate on the whole GPO. If you need to read and modify the settings **within** a GPO, that where the **GPAE** comes in.

With the **GPAE**, you can read and write settings from most policy areas, including Administrative Templates, Security, Group Policy Preferences, Software Installation, Folder Redirection, Scripts, IE Maintenance and more!

To Evaluate

To evaluate our free and commercial solutions for Group Policy automation, visit the following locations

- For the free **GPMC Cmdlets**, visit www.sdmsoftware.com/freeware and register to download the GPMC snap-in.
- For an evaluation copy or a demo of the **GPAE**, visit www.sdmsoftware.com/group_policy_scripting and register for a fully functional 15-day evaluation copy of the GPAE.