# SDM Software Group Policy Automation Engine 4.0 ----- Windows® PowerShell User Guide

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

*With PowerShell*


**SDM Software, Inc.**
www.sdmsoftware.com

Updated 4/9/2016

# Table of Contents


Overview ..... 4

Installation ..... 4

How Do I Get Started with GPAE? ..... 5

- Limitations of GPAE and the local GPO ..... 5
- GPAE and Caching of Settings ..... 6
- Accessing a Domain-based GPO ..... 6
- GPAE Structure ..... 6

GPAE and Enumerations ..... 8

Accessing and Modifying Policy Items with GPAE ..... 8

- GPAE and Operating System Versions ..... 8
- Managing Administrative Template Policies ..... 8
  - Managing Administrative Template Policies Containing Parts ..... 10
- Leveraging Custom ADM Files ..... 14
- GPAE Setting Paths and Escape Characters ..... 15
- Managing Security Policies ..... 16
  - Account Policies ..... 16
  - Audit Policy ..... 18
  - User Rights Assignment ..... 19
  - Security Options ..... 20
  - Event Logs ..... 21
  - Restricted Groups ..... 22
  - System Services ..... 23
  - Registry & File System Security ..... 26
  - Software Restriction Policy ..... 29
- Windows Firewall with Advanced Security ..... 33
- Scripts Policy ..... 33
- Deployed Printers ..... 35
- Software Installation Policy ..... 35
- Folder Redirection Policy ..... 39
- IE Maintenance Policy ..... 41
- Group Policy Preferences ..... 43

GPP Drive Mapping ..... 44
Define Power Management Schemes ..... 50
GPP Printer Mappings ..... 51
GPP Internet Settings ..... 52
Dell Software's Quest Authentication Services ..... 53
Access Control ..... 53
Sudo Settings ..... 54
Summary ..... 55

## Overview

Samples are included here for the use of SDM Software's Group Policy Automation Engine (GPAE) 4.0x. GPAE provides an automation interface into Group Policy Object (GPO) settings, and supports reading and writing the following policy areas:

- Administrative Templates
- Software Installation
- Folder Redirection
- Security Settings
- IE Maintenance
- Scripts
- Deployed Printers
- Group Policy Preferences

GPAE 4.0 supports reading and writing settings from Dell Software's Quest Authentication Services (QAS). GPAE supports a subset of the settings offered by QAS. These areas of QAS are supported in GPAE 4.0 and above:

- Access Settings – users.allow and users.deny Configuration
- Sudo settings

The primary interface shipped with GPAE is PowerShell. GPAE ships a single PowerShell cmdlet called **Get-SDMGPObject**. From that cmdlet all operations related to reading and writing GPO settings are originated.

***Note: If you want to perform operations against the GPO, rather than its settings (for example, creating, deleting, linking, or setting permissions on GPOs) you can use our free PowerShell cmdlets for GPMC, which ship as part of the GPAE setup, or can be found at [www.sdmsoftware.com/freeware](www.sdmsoftware.com/freeware).***

## Installation

The provided MSI file will:

- Install the GPAE PowerShell files and related components
- Provide a PowerShell "module" called SDM-GroupPolicy which includes the **get-sdmgpobject** cmdlet.
- Create a Program Group on your Start Menu, under **SDM Software, Group Policy Automation Engine**, with 4 icons.
    - The icon labeled "**Group Policy Automation Engine 4.0 Object Reference**" is the documentation for the GPAE object model, and provides examples for each policy area.
    - The icon labeled "**Launch GPAE 4.0 for Powershell"** is a shortcut that launches PowerShell with the GPAE module loaded, allowing you to start using GPAE right away.

## How Do I Get Started with GPAE?

Even if you are not a PowerShell guru, you can leverage GPAE to quickly manage your environment's Group Policy. The first thing to know is how to "connect" to a GPO, which is required to read or write GP settings.

> GPAE can connect to either local or domain-based GPOs. Local GPOs can be on the machine where the GPAE is installed or on remote workstations or servers. As long as you have administrative access to a remote system, you can connect to and manage its local GPO. This includes multiple local GPOs on Vista and Server 2008 boxes. For now, let's look at how you can connect to local and domain-based GPOs.

In PowerShell, connecting to a GPO involves assigning the GPO to a variable. For example, if connecting to the local GPO on the machine where GPAE is installed, type the following in PowerShell:
*$gpo = get-sdmgpobject*

Either of these commands will create a reference to the local GPO on the local system. If you type out the value of the $gpo variable, you will see output similar to Figure 1 below.

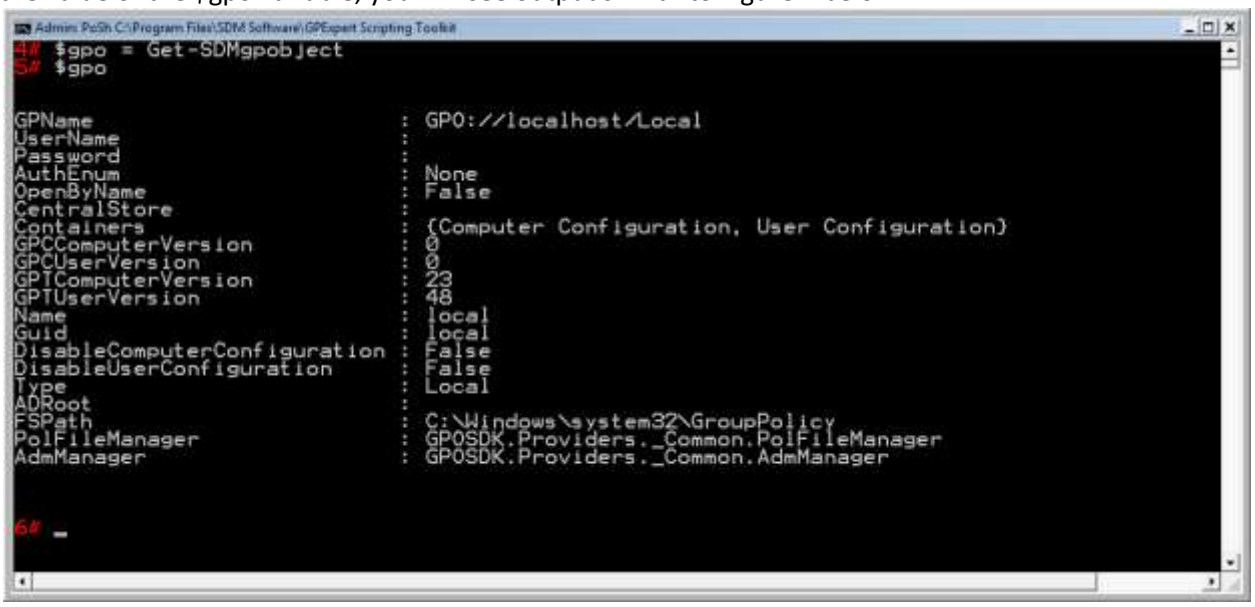



**Figure 1: Viewing the output of a GPO reference**

To get a reference to a local GPO on a different system, use the following syntax:
*$gpo = get-sdmgpobject –gponame "gpo://sdm1/Local"*

where **sdm1** is the name of the remote system.

## Limitations of GPAE and the local GPO

1. The local GPO does not natively support certain policy areas, such as Software Installation and Folder Redirection. As a result, GPAE does not support reading or writing from/to these areas on the local GPO.
2. GPAE **does not support reading or writing most local security policy**. This is primarily a limit on the way security policy in the local GPO is stored/not stored. When you edit the local GPO natively, any settings made under Computer Configuration\Windows Settings\Security Settings are made directly against the live system, rather than stored in settings files (as is the case on a domain-based GPO). However, with the 3.0 release of GPAE, we did add support for reading and writing settings under

Computer Configuration\Windows Settings\Security Settings\Local Policies\Security Options, within the local GPO.

## GPAE and Caching of Settings

When a reference to a GPO is created and stored in a variable it no longer represents the live GPO. The 'reference' is a copy. You are now free to update the reference without affecting the live GPO. Once the local reference is updated to meet the administrator's intent, it then needs to be re-committed (or saved) back up to the live GPO. In the samples throughout this document you will see how the 'Save()' method is called to commit your changes. In addition to the 'Save()' method you will see a 'Refresh()' method which will update the 'reference' copy with changes to the live GPO.

## Accessing a Domain-based GPO

For most tasks related to GPAE, you will want to read and write settings from domain-based GPOs. The syntax is similar to the local GPO examples, with one key difference—you need to provide the "connection string" to the domain-based GPO you wish to access. If you have done any AD scripting, you will find the format similar. The following example shows how you would connect to the Default Domain Policy GPO in the domain called cpandl.com

```
$gpo = get-sdmgpobject  –gponame "gpo://cpandl.com/Default Domain Policy" -openbyName
```

Note that in this example above, I use the –gponame parameter as I did in the previous remote local GPO example. But this time instead of providing a machine name, I give it an Active Directory DNS domain name. After the domain name, I provide the friendly name of the GPO I want to connect to. Finally, I use the **openbyName** parameter to indicate I want to connect to the GPO by name rather than its GUID, which is the default, as shown here:

```
$gpo = get-sdmgpobject  –gponame "gpo://cpandl.com/{31B2F340-016D-11D2-945F-00C04FB984F9}"
```

You can also connect to GPOs in other domains or with other credentials, other than the ones in which the GPAE is installed. For example, if I want to access a GPO in a child domain called west.cpandl.com, I can use the following syntax:

```
$gpo = get-sdmgpobject  -gponame "gpo://west.cpandl.com/{31B2F340-016D-11D2-945F-00C04F
-username west\administrator -password Passw0rd1
```

Once you've connected to a GPO, you can read and write settings from the GPO. Some background on GPAE's structure, or object model, is helpful for understanding examples of using each supported policy area.

## GPAE Structure

Within GPAE, a GPO is broken into **Containers** and **Settings**. If you look at a GPO in the GP Editor, you will see a hierarchy of folders that represent the GPO namespace. For example, Computer Configuration\Administrative Templates\System\Group Policy is a "folder path" within the GPO namespace. In GPAE, that path represents a set of containers. Each level in the folder hierarchy is its own container. Within a container path, there are a number of settings that can be read and written to. These are the policy items. For example, the "Group Policy Slow Link Detection" policy item shown in Figure 2 below is a **Setting**, as far as GPAE is concerned.

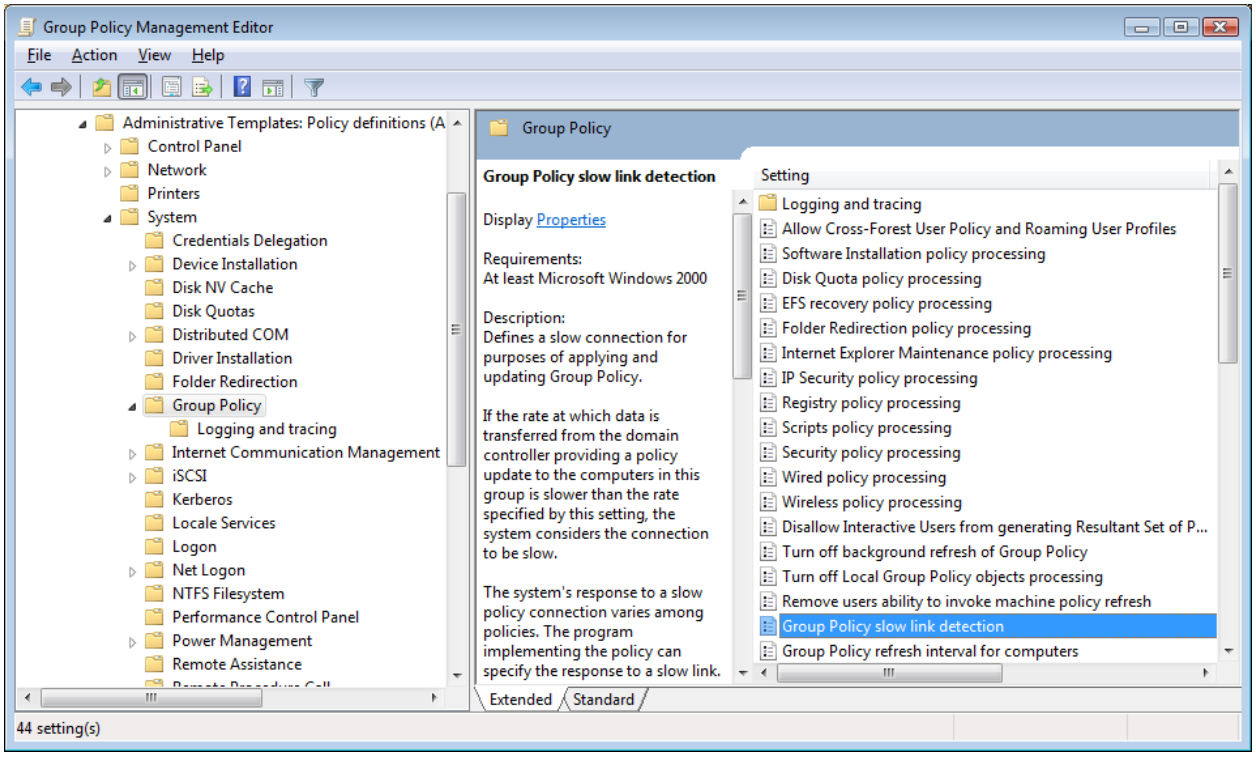



**Figure 2: Viewing GPO Settings**

This model of containers and settings applies whether you are reading or writing within Administrative Templates, Software Installation or Security policy. The main differences arise within each area when it gets down to how you access and modify those settings, as you will see in our examples below. The key part is that in order to read and write settings you need to first tell GPAE which container and setting you want to modify, then you perform the read or write.

## GPAE and Enumerations

Enumerations are an important part of GPAE. Enumeration is a programming construct that allows us to define values with words. For example, in GPAE an Administrative Template policy might be enabled by setting its state property to "Enabled," but "Enabled" must have a value associated with it when saved to the GPO. For that reason, we use the convention *[GPOSDK.AdmTempSettingState]"Enabled"* to indicate that an Administrative Template setting is enabled. This convention is specific to PowerShell—but the enumeration itself comes from GPAE. In PowerShell, when you want to refer to an enumeration, you use the convention of:
[type name]"<member name>"

Therefore, in this example above, [GPOSDK.AdmTempSettingState] is the type name that comes from GPAE and "Enabled" is the member of that type that we want to leverage. GPAE contains many different enumerations for each type of policy area we support. You can see these enumerations and their members within this user guide and the accompanying "GPAE Object Reference" help file that installs with the product.

## Accessing and Modifying Policy Items with GPAE

In this section, we present samples for each of the supported policy areas within GPAE. These samples are presented in PowerShell and provide examples of reading and writing settings within each policy area.

### GPAE and Operating System Versions

Because each version of Windows supports certain policy capabilities, you will generally only be able to access those capabilities from that platform. For example, the Pushed Printers feature is only available in GPAE when it's run from a Vista or Server 2008 box, just as in the native tools. Similarly, if you are accessing Administrative Templates policy from Vista or Server 2008, the GPAE will use the ADMX template files and not ADM files (and vice-versa on XP and Server 2003). And of course, within Administrative Templates, you will only be able to access policy items that are supported on the platform that the GPAE is running from.
**One major exception to the rules above: GP Preferences settings are accessible from down-level (i.e. XP and Server 2003) installations of GPAE.**

## Managing Administrative Template Policies

The most common area used within the GPAE is Administrative Template policy. GPAE works within the GPO namespace and leverages ADM or ADMX templates to provide an extensible model for managing registry policy. Because Admin Template policy comes in various forms – from a simple enabled or disabled policy to more complex PART-based policies that have additional options – you will use slightly different methods to access each type. For example, to read the state of a particular policy, use this general approach:

```
$gpo = get-sdmgpobject -gponame "gpo://cpandl.com/GPAE Demo" –openbyname
$setting = $gpo.GetObject("Computer Configuration/Administrative Templates/System/Logon/Always wait for the network at computer startup and logon");
$setting.Get("State")
```

What this short 3-line script does is:
Line 1 - Get a connection to a domain-base GPO called GPAE Demo
Line 2 - Create a reference to an Administrative Template policy setting ("Always wait for the network…") and assign it to the $setting variable.
Line 3 - Read the current state of that policy setting. "State" is a property on the setting object, with three possible values:

-1 is Not Configured
1 is Enabled
0 is Disabled

This is the case for any Administrative Template Policy. If you want to read a setting, use the Get() (or GetEx() in the case of multi-valued settings) method. If you want to write a setting, use Put() (or PutEx() in the case of multi-valued settings). If you want to see other properties on a given setting, type:
*$setting.GetPropertyNames()*

Then call Get() on those property names, as in Figure 3 below:

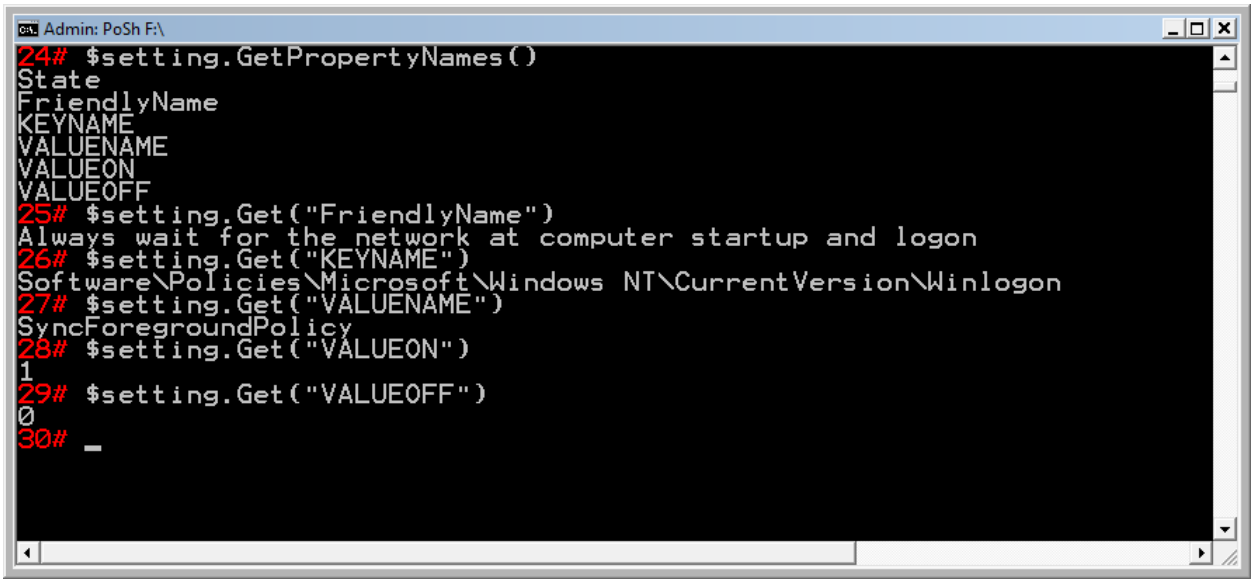



**Figure 3: Retrieving Setting Property Names**

As seen in the figure above, GPAE not only keeps the state of the policy item, but also the underlying metadata from the ADM or ADMX file, showing which registry key and value is impacted by this policy.

Now that we've read a simple Administrative Template Setting, let's look at how you can modify settings within this policy area. In our example above, it takes only a slight modification of script:

*$gpo = get-sdmgpobject -gponame "gpo://cpandl.com/GPAE Demo" –openbyname*
*$setting = $gpo.GetObject("Computer Configuration/Administrative Templates/System/Logon/Always wait for the network at computer startup and logon");*
*$setting.Put("State", [GPOSDK.AdmTempSettingState]"Enabled")*
*$setting.Save()*

In this example, you see that the first two lines of the script are the same as above. However, after we get a reference to the setting, we need to use the Put() method to modify the State property, to either enable or disable the setting. In our example here, Put takes two parameters. First, the name of the property we are modifying ("State") and then the state we are putting it into. This takes a special form. The [GPOSDK.AdmTempSettingState] text is called an enumeration, and is a special designator that tells GPAE that "Enabled" means something more than just a string. Finally, we need to save the setting change we made—this commits the change to the GPO. Until you call the Save() method on the setting, it will not be persisted into the GPO.

Note, that if you want to set the setting to either Disabled or Not Configured, you would change the 3rd line above to one of these:

*$setting.Put("State", [GPOSDK.AdmTempSettingState]"Disabled")*
*$setting.Put("State", [GPOSDK.AdmTempSettingState]"Undefined")*

## Managing Administrative Template Policies Containing Parts

The previous examples assumed management of Admin Template settings that have simple "Enabled," "Disabled," "Not Configured" settings. However, many settings contain these basic states as well as "parts" that specify additional options. Figure 4 shows an example of a Windows Update policy that uses these parts:

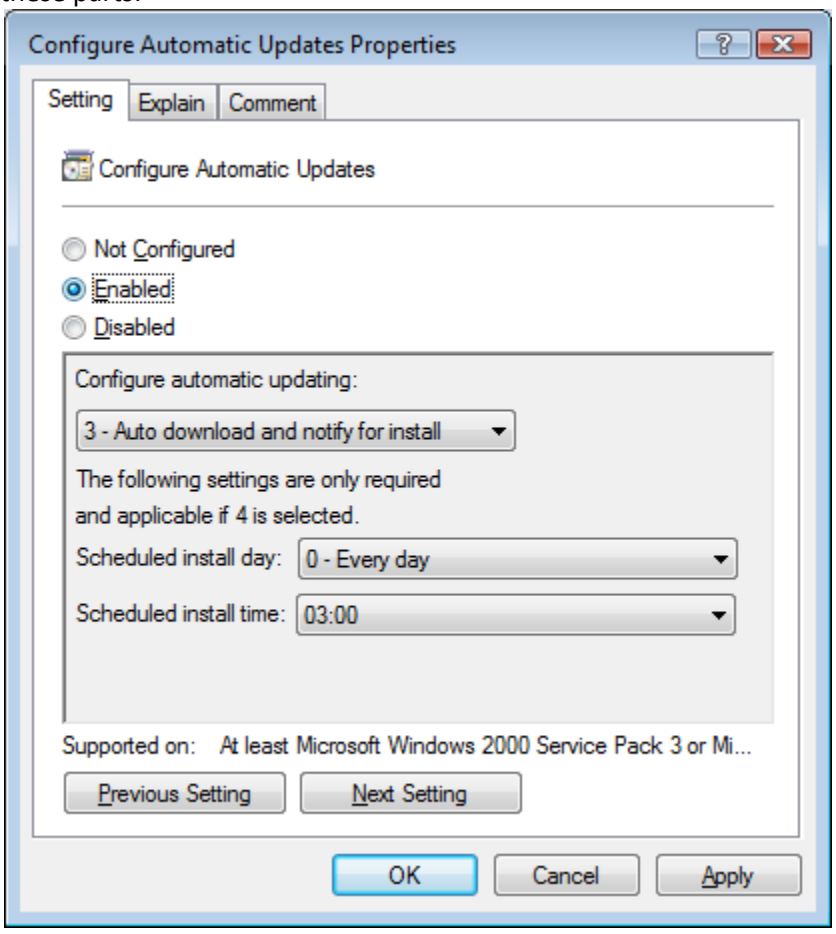



**Figure 4: Viewing a Policy with Parts**

GPAE is capable of reading and writing settings from/to these parts. The following code shows how to write settings to one of these parts:

```
$gpo = get-sdmgpobject  -gponame “gpo://cpandl.com/GPAE Demo” –openbyname
$setting = $gpo.GetObject("Computer Configuration/Administrative Templates/Windows Components/Windows Update/Configure Automatic Updates");
$setting.Put(“State”, [GPOSDK.AdmTempSettingState]"Enabled")
$setting.Put("Configure automatic updating:",3)
$setting.Put("Scheduled install day: ",1)
$setting.Put("Scheduled install time:",3)
$setting.Save()
```

Notice that after we enabled the policy “Configure Automatic Updates,” we updated the various parts, shown in Figure 4 above, by calling Put() against each of the properties representing the parts. How did we know what the names of the properties were, and what the possible values are? We previously called *$setting.GetPropertyNames()* after the first two lines, which returned the information shown in Figure 5 below.

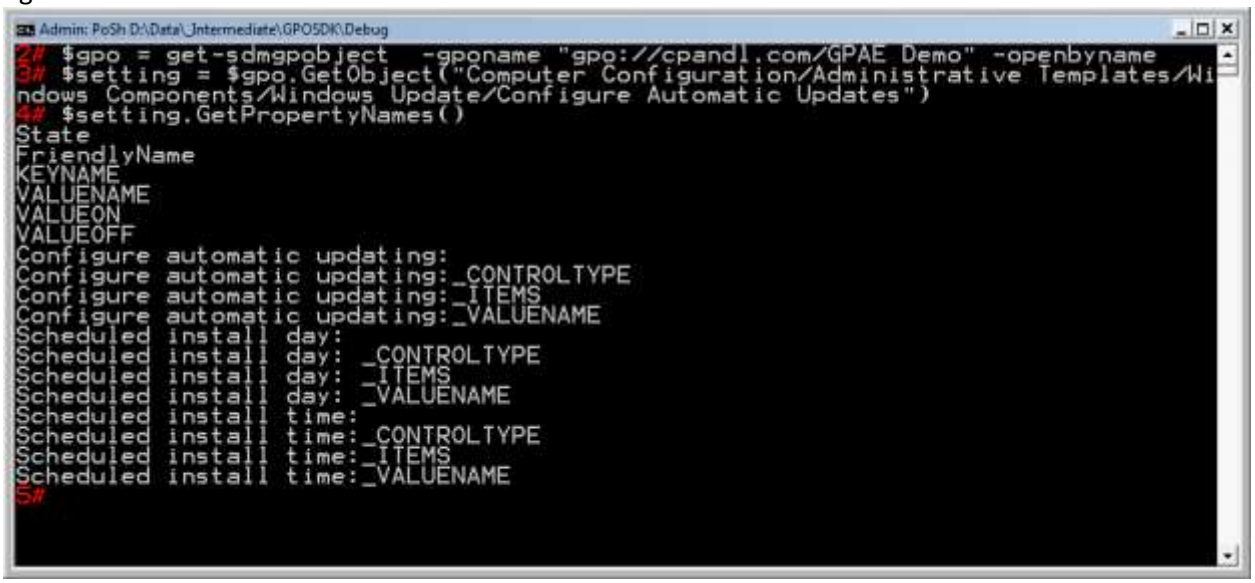



**Figure 5: Viewing property names on a setting**

As you can see in the Figure above, there are a number of base properties, such as the State property, which is read-write; some read-only properties like KEYNAME and VALUENAME associated with that base State property; and the properties associated with the Parts. In this case, this policy has 3 parts: “Configure automatic updating:”, “Scheduled install day:”, and “Scheduled install time:”. Note that each part also has 3 read-only properties associated with it that end in _CONTROLTYPE, _ITEMS and _VALUENAME. Of these, the most interesting for scripting of parts is the _ITEMS property. Once you know the name of the part property you wish to read or write, the _ITEMS collection will give you all of the possible values for that property, as shown in Figure 6 below:

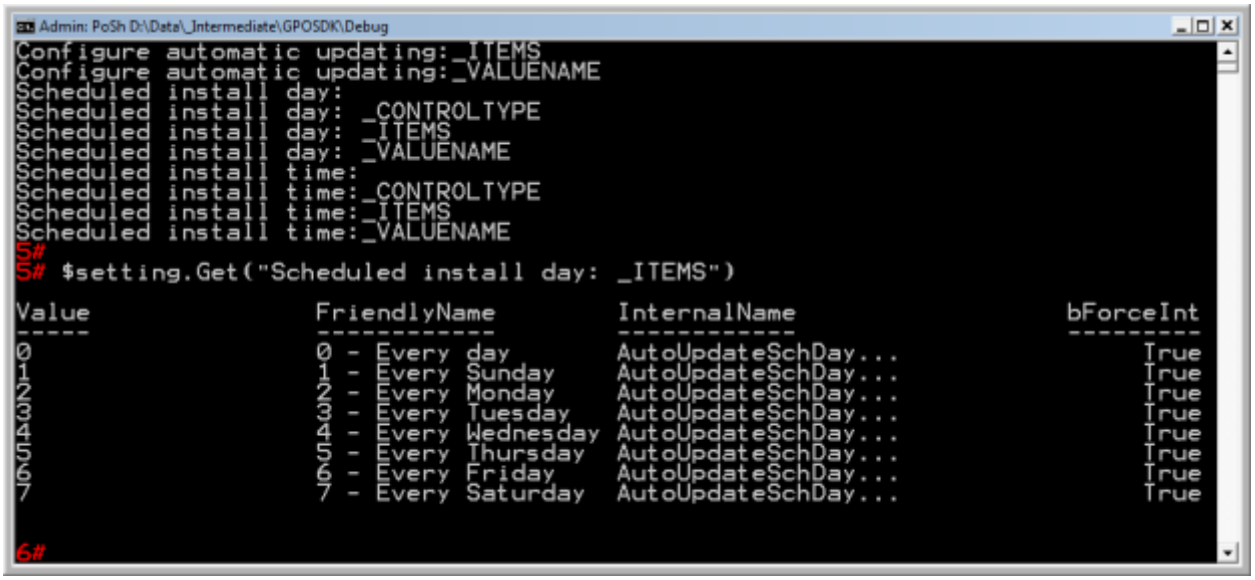



**Figure 6: Viewing Part property possible values**

You can then use either the Value or FriendlyName property for that part property to set its value. **Note that when you are putting or getting the value of a property, the property name must follow exactly with what you see in the GetPropertyNames() call—it is case and space sensitive. For example, in Figure 5 above, "Scheduled install day: " contains a space after the colon. That is part of the property name and must be included in any read or write of the value.**

### *Managing Listbox Settings within AdministrativeTemplates*

Listboxes are a special kind of part within Administrative Template policy and require some special handling. A listbox part, as it appears in the GP Editor, allows you to enter lists of values for a given setting. In some cases, the listbox supports lists of key-value pairs, as in the commonly used IE "site-to-zone-assignment" policy shown in the figure below:

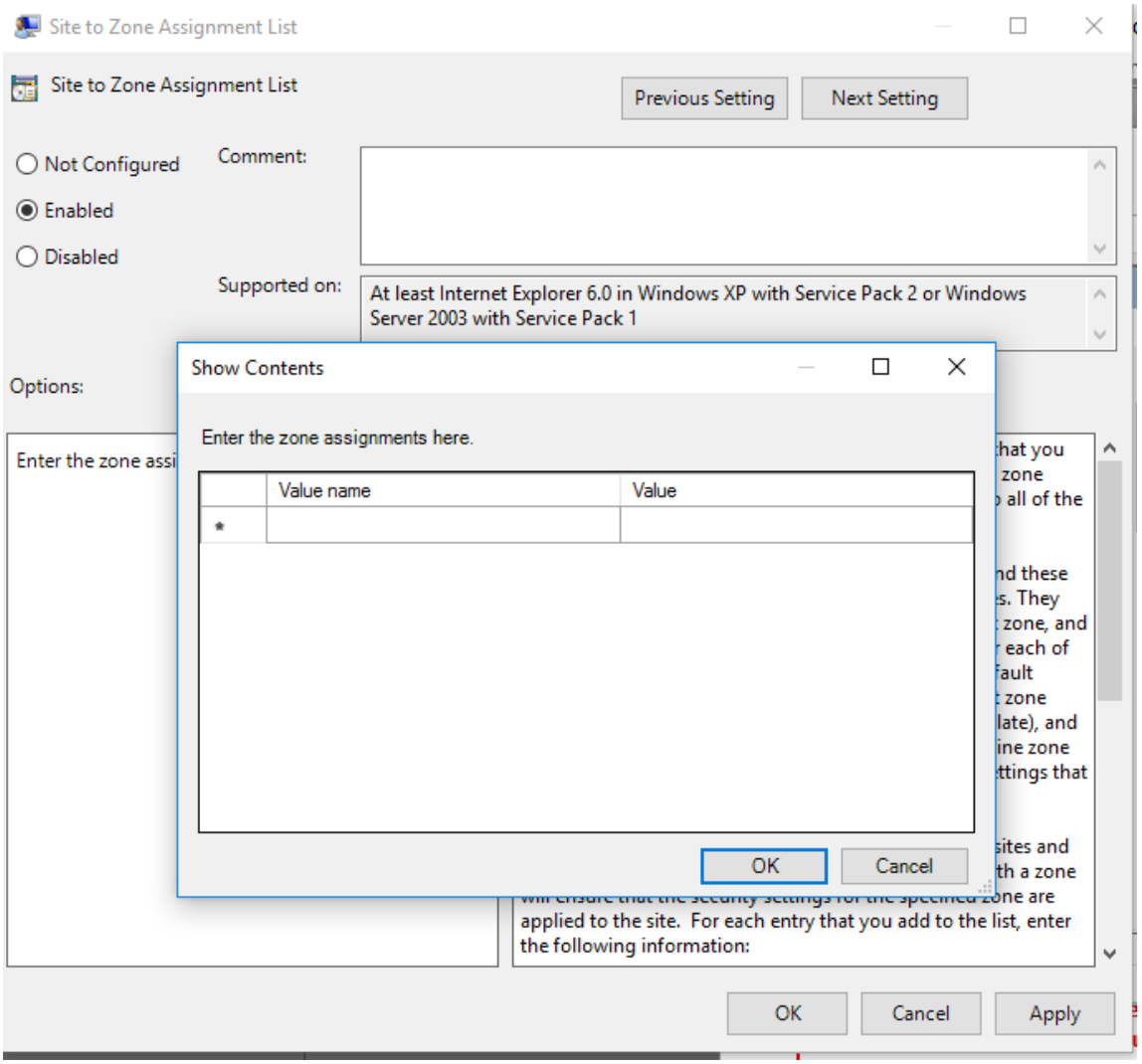



*Figure 6a : Viewing a key-value pair listbox*

The following script sample shows how you can manage listboxes using GPAE:

```
$gpo = get-sdmgpobject -GpoName "gpo://cpandl.com/sitetozoneassignment" -OpenByName
$setting = $gpo.GetObject("Computer Configuration/Administrative Templates/Windows
Components/Internet Explorer/Internet Control Panel/Security Page/Site to Zone
Assignment List")
#enable and add new site to zone assignments
$setting.Put("State",[GPOSDK.AdmTempSettingState]"Enabled")
#sites and their values are stored in the following two properties, as arrays
#add new sites
$sites = @("www.microsoft.com","www.google.com","www.cisco.com")
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE","Enter the zone assignments
here._KEYS",$sites)
#add values corresponding to each site
$values = @("1","2","2")
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE","Enter the zone assignments
here.",$values)
$setting.Save()
#now add a new site to the existing list

$setting.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND","Enter the zone assignments
here._KEYS","www.apple.com")
```

```
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND","Enter the zone assignments
here.","3")
$setting.Save()

#now remove a site

$setting.PutEx([GPOSDK.PropOp]"PROPERTY_DELETE","Enter the zone assignments
here._KEYS","www.microsoft.com")
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_DELETE","Enter the zone assignments
here.","1")
$setting.Save()

#get current values
$setting.Get("Enter the zone assignments here._KEYS")
$setting.Get("Enter the zone assignments here.")
```

## Leveraging Custom ADM Files

It is possible to use Custom ADM files with GPAE. However, you need to take extra steps to make GPAE aware of the custom settings, and there are some limitations using these settings. When a custom ADM is added to a GPO, it is copied up to the SYSVOL portion of that GPO on the PDC emulator Domain Controller (DC) and then replicated to all other DCs. It's copied up to a folder within the GPO called "ADM," as shown in Figure 7 below.

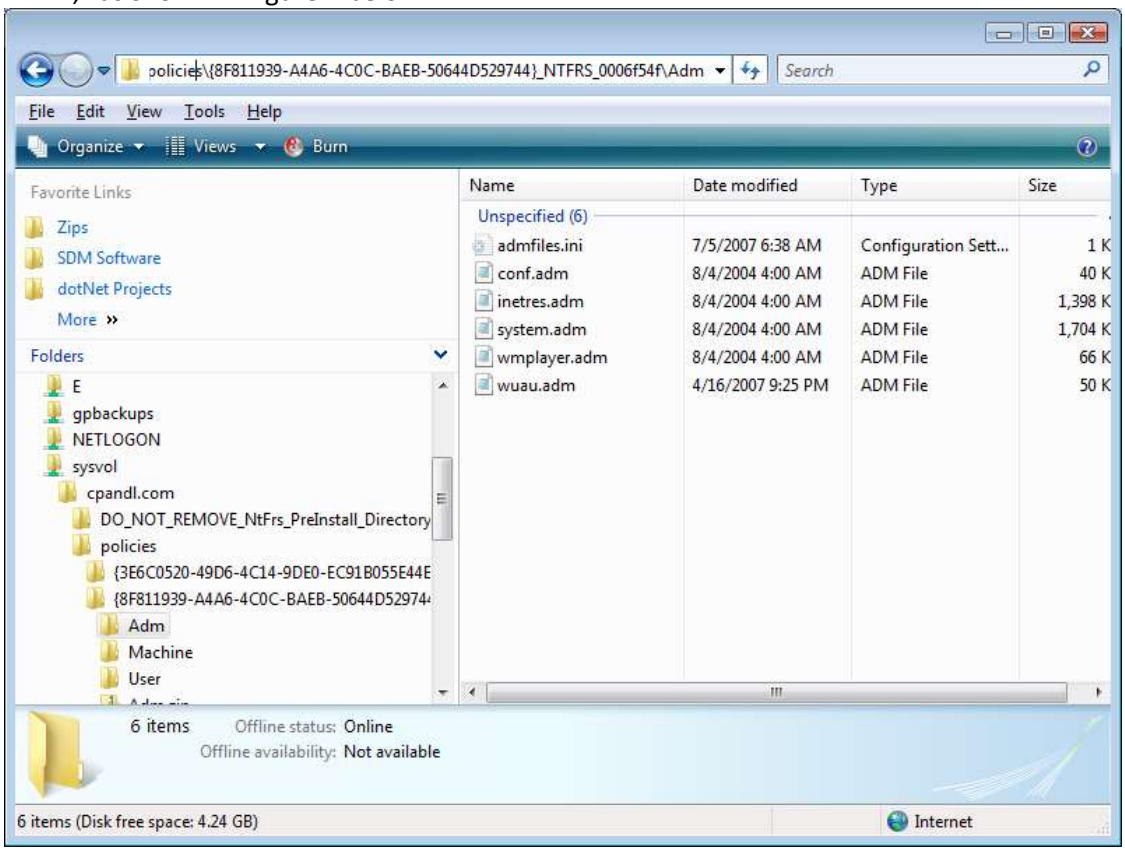



**Figure 7: Viewing the ADM folder in the SYSVOL portion of a GPO**

Within this ADM folder, you will also find a file called **admfiles.ini** that contains an entry for each of the 5 standard ADMs that are part of any GPO created from Windows XP or 2003. If you want GPAE to recognize your custom ADM, add a line to this admfiles.ini file that includes your custom ADM. For example, if I have a custom ADM that I've added to my GPO called mysettings.adm, the admfiles.ini file would need to look like Figure 8:

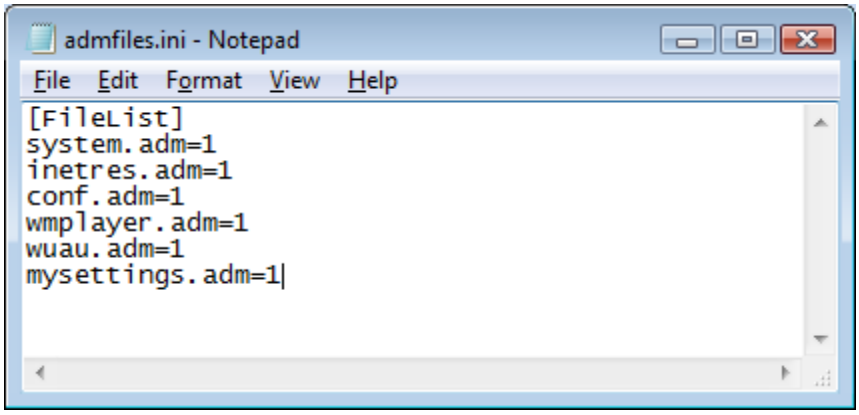



**Figure 8: Viewing a modified ADMfiles.ini**

Note that we added **mysettings.adm=1** to the last line of the file. The next time GPAE is launched and connects to this GPO, it will notice that there is this new custom ADM and incorporate it into the Administrative Templates namespace.
If you have custom ADMX/ADML files, you don't have to do anything special for GPAE to recognize them. You can copy them to either your Central Store or the local C:\Windows\PolicyDefinitions folder in order for them to be recognized.

**NOTE: Not all custom ADMs or ADMXs will be properly recognized by GPAE. The ADM syntax can be fairly "loose" and there are some syntactical forms of an ADM that GPAE cannot currently parse correctly. You will notice this happening when you receive a "Container not found" error when trying to read or write a particular part of the Administrative Templates namespace. Removing the errant custom template will usually fix the problem. Please report any of these to support@sdmsoftware.com and provide the custom ADM or ADMX file that you are using so that we may incorporate these fixes into future versions of GPAE.**

## GPAE Setting Paths and Escape Characters

The method for referencing containers and settings in GPAE requires that you use forward-slashes to distinguish the hierarchical nature of the GPO namespace. For example:

*$setting = $gpo.GetObject("Computer Configuration/Administrative Templates/Windows Components/Windows Update/Configure Automatic Updates");*

This code references a setting under the Windows Update section of Windows Components using forward-slashes to differentiate container levels. This works well until you come across a setting that contains a legitimate forward-slash that does not delineate a container. For example:

*$setting = $gpo.GetObject("User Configuration/Administrative Templates/Windows Components/Microsoft Management Console/Restricted/Permitted snap-ins/Disk Management");*

In this example, the forward-slash in "Restricted/Permitted snap-ins" does not constitute a break in the container hierarchy—it is just used to separate two words in the path. In that case, we need to be able to "escape" the legitimate use of '/' and not confuse GPAE. Version 4.0 of GPAE supports an escape

character (`), to the left of the 1 key, for this purpose. If you are using the C# library of GPAE, use one of these characters, as follows:

```
$setting = $gpo.GetObject("User Configuration/Administrative Templates/Windows Components/Microsoft Management Console/Restricted`/Permitted snap-ins/Disk Management");
```

But within PowerShell, since the ` character is also the PowerShell escape character, you will need to use it twice, as follows:

```
$setting = $gpo.GetObject("User Configuration/Administrative Templates/Windows Components/Microsoft Management Console/Restricted``/Permitted snap-ins/Disk Management");
```

This guarantees that the GPAE escape character is itself escaped, and will allow access to the setting.

## Managing Security Policies

Security policy provides a number of different sub-policy areas. GPAE supports some, but not all, of the policies under the Computer Configuration\Windows Settings\Security Settings namespace. Areas not currently supported include IP Sec Policies and Public Key Policies as well as the new for Vista and Server 2008 Wired and Wireless Policies. However, GPAE does support the following areas within security policy:

- Account Policies, including Password, Lockout and Kerberos Policies
- Local Policies including Audit policy, User Rights Assignment and Security Options
- Event Log policies
- Restricted Groups policies
- System Services
- Registry & File System
- Software Restriction Policies (only Path & Hash Rules supported)

Let's look at using each of these areas now. The general approach is the same as in other policy areas: connecting to a GPO, getting a reference to a container and setting and then reading and writing that setting.

### Account Policies

You can define account policy on any domain-based GPO. The following code shows how to get the current minimum password length from the Default Domain Policy GPO.

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/Default Domain Policy" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Account Policies/Password Policy/Minimum password length");
$setting.Get("Value")
```

We created a reference to the setting within the GPO namespace that is the "Minimum password length" policy item, and then used the Get() method on that setting to get its value (i.e. the minimum password length).

Similar to getting a reference to an individual setting, we can reference a container and get a list of the settings available underneath it. For example, if I want to see what policy items are available under the Password Policy "container" in the example above, I can do as Figure 9 shows:

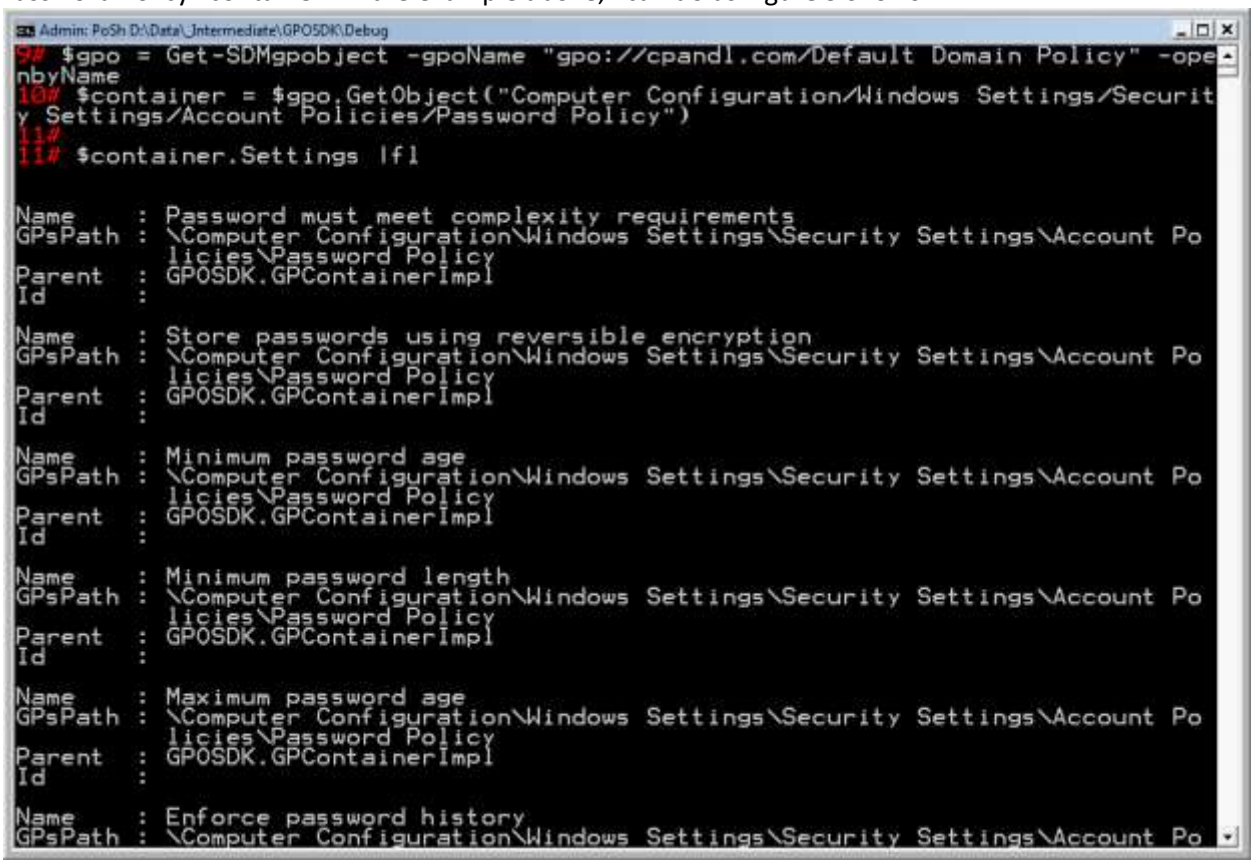



**Figure 9**: **Viewing the contents of a policy container**

We set the $container variable equal to the path just above the "Minimum Password Length" policy item. This equates to what GPAE refers to as a container, and is similar to a folder within a file system. A container object in GPAE contains a property called Settings, which is a collection of all of the settings within that container. When I issued the ***$container.Settings |fl*** command above, it output a list of the policy settings available under Password Policy, including my Minimum password length policy.

That explains how to retrieve account policy settings. Let's show now how to set or modify them. For simple policy items like minimum password length, the policy item has only two properties. Similar to Administrative Templates policy, first you need to tell Group Policy that you want to enable the setting, then you need to give it a value, as shown below:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Account Policies/Password Policy/Minimum password length");
$setting.Put("Defined",$true)
$setting.Put("Value",14)
$setting.Save()
```

In this example above, we Put() the "Defined" property to $true to enable the policy and then set its "Value" property at 14, the minimum password length we want for this GPO. For those security settings that only have an on or off setting, we need only use this statement to enable the policy:
*$setting.Put("Defined",$true)*

This is the equivalent of the *$setting.Put("State", [GPOSDK.AdmTempSettingState]"Enabled")* statement enabling an Administrative Template policy, but in this case, "Defined" is specific to security policy settings.

For Account Policy settings that have states of enabled or disabled, the following sample describes how to set those policies.

```
$gpo = get-sdmgpobject -gponame "gpo://cpandl.com/Account Policy Test" -openbyname
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Account Policies/Password Policy");
$setting = $container.GetObject("Password must meet complexity requirements")
$setting.Put("Defined",$true)
$setting.Put("Value",0)
$setting.Save()
```

In this example, the "Password must meet complexity requirements" policy must be defined, and then you can tell the GPO whether you want to enable or disable it. So, we set the Defined property to $true and then set the Value property to 0 to disable it (or 1 to enable). **Note that if you are changing the policy from enabled to disabled, you need to make sure your script code explicitly sets the Defined property to true each time you set the value.**

### Audit Policy

Audit policy lets you set what you wish to audit on a target system. For example, if we want to enable success and failure auditing of directory service access within a GPO, the following code snippet will do that:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Local Policies/Audit Policy/Audit directory service access");
$setting.Put("Defined",$true)
$setting.Put("Value",[GPOSDK.AuditPolicyValue]"SuccessAndFailure")
$setting.Save()
```

In this example, we set the "Defined" property of the "Audit directory service access" setting to true, then use a special "Enum" for this policy called GPOSDK.AuditPolicyValue to set the success and failure flags to "SuccessAndFailure.' Note that this Enum can have the following possible values:

| Option | GPAE Enumeration Value |
|---|---|
| Failure Auditing only | [GPOSDK.AuditPolicyValue]"Failure" |
| No Auditing | [GPOSDK.AuditPolicyValue]"None" |
| Success Auditing only | [GPOSDK.AuditPolicyValue]"Success" |
| Success and Failure Auditing | [GPOSDK.AuditPolicyValue]"SuccessAndFailure" |

So if you want to change a currently audited policy item to not be audited anymore, you can set it from its current value to “None.” When that GPO is processed, auditing will be disabled for that particular item.

### User Rights Assignment

User Rights Assignment policy lets you define users and groups for whom you wish to grant user rights to performs tasks against systems that receive the policy. From GPAE, you need to first enable the policy and then add the security principals (users or groups) to the policy. Because the list of security principals can contain more than one, this is one of those policies where you’ll want to use the GetEx() and PutEx() methods to retrieve and set values. For example, the following PowerShell script retrieves the current values from the “Allow Logon Locally” user right, and then adds the “GPO Admins” group to the user right:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/Default Domain Controllers Policy" -openbyName;
$settings = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Local Policies/User Rights Assignment/Allow log on locally");
# retrieve the current settings
$settings.GetEx("Value")
# now add a group to the current settings
$currentValues = [System.Collections.ArrayList]$settings.GetEx("Value")
$currentValues.Add("CPANDL/GPO Admins")
$settings.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE", "Value", $currentValues)
$settings.Save()
```

$settings.GetEx(“Value”) returns the current contents of the “Value” property on the setting, which is the “Allow Log on Locally” user right. Next we want to add a new group to the list, so, we set the variable $currentValues equal to the $settings.GetEx(“Value”) statement, but we precede it with the [System.Collections.ArrayList] type name. That tells PowerShell to treat this current list of values as an arraylist. We add a new element to the array in the next statement, by adding “CPANDL\GPO Admins” to the list. Next, we call the PutEX method, which takes three parameters. The first parameter is a GPAE specific ENUM called GPOSDK.PropOp. This enum is meant to be used with the PutEx method and is designed to update, clear, append or delete elements from a multi-valued property.

Possible values for the **GPOSDK.PropOp** enumeration:

| Option | GPAE Enumeration Value |
|---|---|
| Clear property (remove all values) | [GPOSDK.PropOp]"PROPERTY_CLEAR" |
| Update an existing property value | [GPOSDK.PropOp]"PROPERTY_UPDATE" |
| Append a new value to the property | [GPOSDK.PropOp]"PROPERTY_APPEND" |
| Delete an existing value from the property | [GPOSDK.PropOp]"PROPERTY_DELETE" |

In this case, we use the "PROPERTY_UPDATE" member of the enum, which clears all current values in the property and writes into it the contents of the 3rd parameter—in this case, $currentValues. Finally we call Save().
Note that if you just want to write a single security principal into a User Rights Assignment policy, you don't have to go through this process of retrieving the current values, appending the new one and writing it back to the Value property. You can simply do this:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/Default Domain Controllers Policy" –
openbyName;
$settings = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Local
Policies/User Rights Assignment/Allow log on locally");
$settings.Put("Defined",$true)
$settings.Put("Value","CPANDL\GPO ADMINS")
$settings.Save()
```

**Note: GPAE now explicitly supports local user accounts. Prior to GPAE version 2.2, if you wanted to add local user accounts to a restricted group or user right assignment, GPAE would try to resolve the account name to a domain user. If there was a domain user with the same name as the local user, the user added to the restricted group or user rights assignment was the domain one. As of version 2.2, you can force GPAE to not resolve the user account to AD by placing an asterisk (*) in front of the user name. In our example above, this would look like:**

**$settings.Put("Value", "*joelocal")**

### Security Options

The Security Options section of security policy allows you to extend it by editing a file on the system where you edit policy. That file is called sceregvl.inf and is located in the c:\windows\inf folder. The following KB article explains how you can use this file to customize settings: http://support.microsoft.com/kb/214752. GPAE supports dynamically discovering existing and customized Security Options policy, so if you have added settings to your sceregvl.inf, GPAE will find them. Note that this file is customized on a per machine basis, so the customized version of the file must be registered on the system where GPAE is installed for the product to see the custom settings.
In the simplest case, most of the security options require setting the "Defined" property and the "Value" property to true to enable that policy, as shown here:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Local
Policies/Security Options/Shutdown: Clear virtual memory pagefile");
$setting.Put("Defined", $true)
$setting.Put("Value", $true)
$setting.Save()
```

You might ask why you have to set both Defined and Value to true. This is because a Security Options policy can be Defined, and then it can be either enabled or disabled. Setting the "Value" property to true is equivalent to checking the "Enabled" radio button in the GP Editor UI, whereas setting it to false is equivalent to disabling it.

In the case of a Security Options policy that requires you to type in a value, the "Value" property is populated with the value you choose. For example, to rename the Administrator account:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/SecuritySettings/Local
Policies/Security Options/Accounts: Rename administrator account");
$setting.Put("Defined", $true)
$setting.Put("Value","NoAdmin")
$setting.Save()
```

In this example, we set the "Value" property to the name we want to change the administrator account to.

### Event Logs

The Event Logs policy lets you configure event log size and retention behavior. As with other security policies, you can Get() or Put() the "Defined" and "Value" properties to read and write settings. In the example below, we set the maximum size and retention behavior for the Application event log.

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Event
Log/Maximum application log size");
$setting.Put("Defined", $true)
# put maximum log size in Kilobytes
$setting.Put("Value",20096)
$setting.Save()
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Event
Log/Retention method for application log");
$setting.Put("Defined", $true)
$setting.Put("Value",[GPOSDK.EventLogRetentionType]"OVERWRITE_AS_NEEDED")
$setting.Save()
```

In the example above, we are first setting the "Maximum application log size" setting (in kilobytes) by setting the "Defined" property to true and then setting the "Value" property to the size of the log we want (in this case, 20096 kilobytes). After we save that setting, we get a reference to the "Retention method for application log" setting, enable it and set its "Value" property to "Overwrite as needed" using the GPOSDK.EventLogRetentionType Enum. Note that this Enum has three possible values:

- `OVERWRITE_AS_NEEDED`
- `OVERWRITE_BY_DAYS`
- `DO_NOT_OVERWRITE`

If you set the OVERWRITE_BY_DAYS option on the retention method setting, you will also need to set the "Retain application log" setting to the number of days you wish to retain.

### Restricted Groups

Restricted groups policy lets you control local group membership on workstations and servers. This policy has both a "Members" and "Member Of" component, and GPAE supports setting both. In the following example, we use the "Members" component to control the membership of the local Administrators group:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Restricted Groups");
$setting = $container.Settings.AddNew("Administrators")
$members = [System.Collections.ArrayList]$setting.GetEx("Members")
$members.Add("CPANDL\GPO Admins")
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE", "Members", $members)
$setting.Save()
```

**Note: GPAE now explicitly supports local user accounts. Prior to GPAE version 2.2, if you wanted to add local user accounts to a restricted group or user right assignment, GPAE would try to resolve the account name to a domain user. If there was a domain user with the same name as the local user, the user added to the restricted group or user rights assignment was the domain one. As of version 2.2, you can force GPAE to not resolve the user account to AD by placing an asterisk (*) in front of the user name. In our example above, this would look like:**

```
$settings.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE","Members"," *joelocal")
```

You'll note a key difference with the script above. In this example, we are assigning the $container variable to the Restricted Groups container, rather than assigning $setting right away. That's because, with Restricted Groups, settings are represented by the groups that you have defined in the policy. If no groups have been defined, you need to first get a reference to the container, and then add the group you want to manage using the $container.Settings.AddNew() method. In this case we want to control members of the local Administrators group so we add that group to the policy and assign it to $setting. Next, we create an arraylist of current member names that we want to belong in the Administrators group, and assign that to $members. In the next line, we add the new group that we want to add to the policy to the arraylist $members. Next, we use the PutEx() method on the setting since we have a multi-valued property, called "Members," that we need to edit. We update the property using the GPOSDK.PropOp Enum's PROPERTY_UPDATE value so that any existing members are removed when we update with our new list. Another method, if we want to add a new group, stored in the $members variable, to any existing groups within the policy, is to use the PROPERTY_APPEND enumeration value, which does not clear the group members list before adding the new members.

The process for managing the "Member of" property is identical. However, in that case, the group you add with AddNew() method is the name of the group that you want to belong to other groups. For example, if you want to add the "CPANDL\Help Desk Admins" group to a computer's local "Remote Desktop Users" group, you would do the following:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Restricted Groups");
$setting = $container.Settings.AddNew("CPANDL\Help Desk Admins")
```

```
$setting.Put("Memberof","Remote Desktop Users")
$setting.Save()
```

Note that in the example above, because we are only putting one group name into the Memberof property, we can use Put rather than PutEx. However, note that this assumes there are no other groups currently defined on this property. If there were, we would have to use PutEx() and the GPOSDK.PropOp Enum.

How about retrieving current restricted groups policy? It's mostly the same process as the above example, except that we use GetEx instead of PutEx. The following code shows the way:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Restricted
Groups");
$groups = $container.Settings
foreach ($groupName in $groups)
{
        $groupName.Name
        "Members: " + $groupName.GetEx("Members")
        "Member of: " + $groupName.GetEx("Memberof")
}
```

In this simple example, we assign $groups to the $container.Settings property. This property equates to all the groups under management within this Restricted Groups policy. Then we "foreach" through each group under management, and dump its Members and Membersof properties to the console. Note that we're simply dumping these properties to a single line of output but you could format the return more neatly, depending on your needs.

### System Services

The System Services security policy lets you control service startup mode as well as service permissions (e.g. who can stop and start services). As a result, there is a component associated with defining the service startup mode and another component that relates to service security (i.e. the Access Control Entries on the service). Relating to GPAE, the list of services that you can control is wide open. The key is to provide the service's name as it appears in the registry under HKLM\System\CurrentControlSet\Services. Each service defined on a system has a registry key under that path, and it's the name of that key that you will need to use to define your system services policy. For example, to control the startup mode of the Windows Update service, type the following:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/System
Services");
$setting = $container.Settings.AddNew("wuauserv")
$setting.Put("Defined", $true)
$setting.Put("StartupType", [GPOSDK.ServiceStartupType]"Disabled")
$setting.Save()
```

Note that in this example, “wuauserv” is the service registry name for the “Windows Update” service. That service is set to disabled in my example here. The GPAE enumeration GPOSDK.ServiceStartupType can have the following possible values:

| Option | GPAE Enumeration Value |
|---|---|
| Service startup set to automatic | [GPOSDK.ServiceStartupType]"Automatic" |
| Service startup set to manual | [GPOSDK.ServiceStartupType]"Manual" |
| Service startup set to disabled | [GPOSDK.ServiceStartupType]"Disabled" |
| Service startup set to disabled | [GPOSDK.ServiceStartupType]"Default" |

By default, when you get a reference to the System Services container, as shown above, it will not have any services defined within it. Call the AddNew() method on the $container.Settings property in order to add your new service definition. This is in contrast to what you see in the GP Editor UI, where all of the services found on the machine on which you are editing the GPO are listed. However, using the GPAE method, you can control any service that you wish, regardless of whether it’s found on the machine or not. You just need to know its registry name rather than its friendly name.

If you want to modify the security ACL on a service to control who can manage it, that takes a bit more work. The good news is that the process for modifying service ACLs is the same for registry and file system security, too, so the knowledge you gain here will carry into those policy areas. The following code sample grants the “GPO Admins” group the ability to stop, start and pause the Windows Update service:

```
1.  $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2.  $container = $gpo.GetObject("Computer Configuration/Windows Settings/Security
    Settings/System Services");
3.  $setting = $container.Settings.AddNew("wuauserv")
4.  $setting.Put("Defined", $true)
5.  $setting.Put("StartupType", [GPOSDK.ServiceStartupType]"Automatic")
6.  $service_security = new-object System.Security.AccessControl.GPOSDK.ServiceSecurity
7.  $flags = [System.Security.AccessControl.GPOSDK.ServiceRights]"Stop" -bor
    [System.Security.AccessControl.GPOSDK.ServiceRights]"Start" -bor
    [System.Security.AccessControl.GPOSDK.ServiceRights]"PauseOrContinue" -bor
    [System.Security.AccessControl.GPOSDK.ServiceRights]"Read";
8.  $rule = new-object System.Security.AccessControl.GPOSDK.ServiceAccessRule("CPANDL\GPO
    Admins",$flag,[System.Security.AccessControl.AccessControlType]"Allow");
9.  $service_security.AddAccessRule($rule)
10. $setting.Put("Security",$service_security.GetSecurityDescriptorSddlForm([System.Security.Acce
    ssControl.AccessControlSections]"All"));
11. $setting.Save()
```

This script does a lot, so let’s go through it by line number:

Lines 1-5 just repeat what we did in the previous script sample, except this time we’re setting the service startup to Automatic.

Line 6 creates a new GPAE-specific object called ServiceSecurity and assigns it to $service_security. This object will hold our permissions that we define later.

Line 7 sets the $flags variable to the actual permissions we want on the service. Note that each permission is defined within an Enumeration called ServicesRights. This enumeration has the following members:

| |
|---|
| Read |
| Write |
| Execute |
| FullControl |
| Delete |
| ReadPermission |
| ChangePermission |
| TakeOwnership |
| QueryConfiguration |
| ChangeConfiguration |
| QueryStatus |
| EnumerateDependents |
| Start |
| Stop |
| PauseOrContinue |
| Interrogate |
| SendUserDefinedControl |

**Note: The GPAE GPOSDK.ServiceRights enumeration is derived from the access rights defined for the Windows Service Control Manager, defined here: http://msdn.microsoft.com/en-us/library/ms685981(VS.85).aspx.**

In Line 7 we use the PowerShell Boolean OR operator (-bor) to combine the individual permissions that we want into a permission set. In the case of granting Stop, Start and Pause rights, we need to grant those four permissions as well as the Read permission in order to fully grant the ability to stop, start and pause a service.

Line 8 uses the $flags variable we just created to create a new object of type ServiceAccessRule, which holds the details of our permission, including the group we want to grant the access ("CPANDL\GPO Admins"), the actual permission set stored in $flags, and the type of permission access – either Allow or Deny—using the AccessControlType enumeration.
Note that the System.Security.AccessControl.AccessControlType is a .Net Framework enumeration, defined at http://msdn.microsoft.com/en-us/library/w4ds5h86(VS.80).aspx – it is not a GPAE enumeration.

Line 9 adds our new rule to the $service_security object (note that we could have had multiple rules that we were defining for multiple permissions). In each case, a new rule must be assigned to $service_security using the AddAccessRule() method.

Line 10 calls Put() on the service setting ($setting) to modify its "Security" property with our $service_security object that contains our permission(s).

Note that we are calling the GetSecurityDescriptorSddlForm() method on the $service_security object because we need to store the permission in SDDL format. This method is actually provided by the .Net Framework, rather than GPAE. We pass the [System.Security.AccessControl.AccessControlSections]"All" enum into this method because we want to use all of the access control entries found in this permission.

To retrieve service properties, simply call Get() on the setting's properties, like this:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/System Services");
$setting = $container.Settings.ItemByName("wuauserv")
$setting.Get("StartupType")
$setting.Get("Security")
```

What is returned for the StartupType will be an integer value equating to the following:
2 – Automatic
3 – Manual
4 – Disabled
4 – Default

The Get() call to the Security property returns an SDDL string corresponding to the permission set granted to the service.

### Registry & File System Security

The steps needed to read and write registry and file system security policy are similar to managing System Services above. Registry and file system security policy let you control permissions on registry keys, file folders and files. Let's start by setting a policy on a registry key:

```
1.  $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2.  $container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Registry");
3.  $setting = $container.Settings.AddNew("MACHINE\Software\SDM Software")
4.  $setting.Put("Defined", $true)
5.  $setting.Put("PermissionControl", [GPOSDK.RegistryPermissionControl]"Replace")
6.  $registry_security = new-object System.Security.AccessControl.RegistrySecurity
7.  $rule = new-object System.Security.AccessControl.RegistryAccessRule(
    "CPANDL\GPO Admins",[System.Security.AccessControl.RegistryRights]"FullControl",
    [System.Security.AccessControl.AccessControlType]"Allow");
8.  $registry_security.AddAccessRule($rule)
9.  $rule = new-object
    System.Security.AccessControl.RegistryAccessRule("BUILTIN\Administrators",[System.Security.
    AccessControl.RegistryRights]"ReadKey",[System.Security.AccessControl.AccessControlType]"Al
    low");
10. $registry_security.AddAccessRule($rule)
11. $setting.Put("Security",$registry_security.GetSecurityDescriptorSddlForm([System.Security.Acc
    essControl.AccessControlSections]"All"));
12. $setting.Save()
```

In Lines 1 and 2, we connect to our GPO and get a container reference to the registry security policy area.
Line 3 is where we add the registry key that we want to control. Note that if we are managing HKLM, we use “MACHINE” and if the key is under HKCU, then “USER” is the keyword to begin with.
Line 4 sets the policy to defined.
In Line 5, we use GPAE’s RegistryPermissionControl enum to control how we want the permissions to propagate. This enum is used by both file and registry security policy, so the options are the same. The choices for this enum are :

| Option | GPAE Enumeration Value |
|---|---|
| Permissions set to “Configure this file or folder then propagate inheritable permissions to subfolders and files” | [GPOSDK.RegistryPermissionControl]”Propagate” |
| Permissions set to “Configure this file or folder then replace existing permissions on all subfolders and files with inheritable permissions” | [GPOSDK.RegistryPermissionControl]”Replace” |
| Permissions set to “Do Not Allow Permissions on this file or folder to be replaced” | [GPOSDK.RegistryPermissionControl]”Delete” |
| Same as propagate above | [GPOSDK.RegistryPermissionControl]”Default” |

These correspond to the UI choices within policy as shown in Figure 10 below.

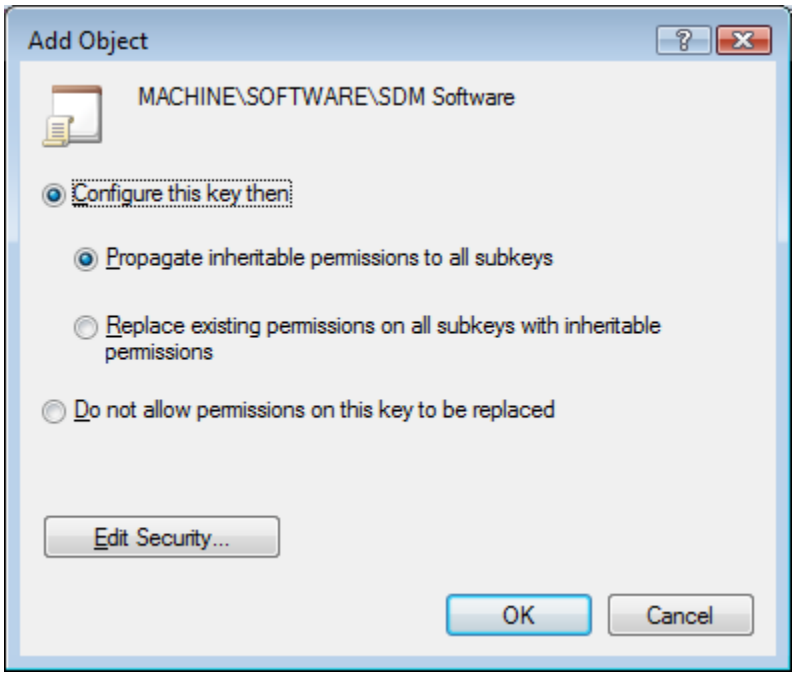



**Figure 10: The propagation options for registry security**

Line 6 creates a new object using the .Net type of System.Security.AccessControl.RegistrySecurity. We'll use this type to store the permissions we want to grant to the registry key.

Line 7 does the work of creating the access rule, using the .Net System.Security.AccessControl.RegistryAccessRule type. This is where we build the permission to grant by providing the group name ("CPANDL\GPO Admins") the rights, using the .Net enum called System.Security.AccessControl.RegistryRights and the access control type (e.g. Allow or Disallow).

Line 8 adds that rule to the previously created $registry_security object.
Lines 9 & 10 create a second rule using a different group and set of rights and add it to the $registry_security object as well.
Line 11 sets the Security property on the setting object with the rule we just created.
Line 12 saves it to the GPO.

File System Security policy is identical in approach to registry security policy. The difference is that we are setting security on files or folders instead of registry keys, so that the rights used and the propagation desired are slightly different. We won't go through the whole process here except to provide a sample for setting folder security on the C:\Program Files\SDM Software folder below:

```
$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/File System")
$dir_name = "%SYSTEMDRIVE%\Program Files\SDM Software";
$setting = $container.Settings.AddNew($dir_name)
$setting.Put("Defined", $true)
$setting.Put("PermissionControl", [GPOSDK.FolderPermissionControl]"Replace")
$folder_security = new-object System.Security.AccessControl.DirectorySecurity
$rule = new-object System.Security.AccessControl.FileSystemAccessRule("Authenticated
Users",[System.Security.AccessControl.FileSystemRights]"FullControl",[System.Security.AccessControl.Inh
eritanceFlags]"None",[System.Security.AccessControl.PropagationFlags]"InheritOnly",[System.Security.A
ccessControl.AccessControlType]"Allow");
$folder_security.AddAccessRule($rule)
$rule = new-object System.Security.AccessControl.FileSystemAccessRule("Authenticated
Users",[System.Security.AccessControl.FileSystemRights]"DeleteSubdirectoriesAndFiles",[System.Security.
AccessControl.AccessControlType]"Deny");
$folder_security.AddAccessRule($rule)
$setting.Put("Security",$folder_security.GetSecurityDescriptorSddlForm([System.Security.AccessControl.
AccessControlSections]"All"));
$setting.Save()
```

The most important note above is the creation of the first $rule object, which uses some more options to create a permission that grants Full Control to the Authenticated Users group, but also sets inheritance flags, propagation rules and access control types. Note that the creation of these rules is strictly a function of the underlying .Net types. That means the FileSystemAccessRule type has a number of constructors that support creating the permission with a variety of characteristics. This is true across system services, registry and file security policy. More information on this type can be found here: http://msdn.microsoft.com/en-us/library/system.security.accesscontrol.filesystemaccessrule.aspx.

### Software Restriction Policy

Software restriction policy is used to control application execution. It can be set under either Computer Configuration or User Configuration. GPAE supports reading and writing Software restriction policy (SRP). SRP has general settings that control its overall behavior and then it has support for 4 types of rules that control application execution based on:

- Path Rules
- Hash Rules
- Certificate Rules
- Network Zone Rules

**NOTE: GPAE currently only supports the first two rule types—path and hash rules.**

Here's how to create a rule that sets the default execution level to Disallowed (i.e. Whitelist mode) and then allows execution of all code under C:\program files:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security
   Settings/Software Restriction Policies/Security Levels/Disallowed");
3. $setting.Put("Default",$true)
4. $setting.Save()
5. $container = $gpo.GetObject("Computer Configuration/Windows Settings/Security
   Settings/Software Restriction Policies/Additional Rules Path");
6. $setting = $container.Settings.AddNew("%ProgramFiles%")
7. $setting.Name = %ProgramFiles%"
8. $setting.Put("SecurityLevel",[GPOSDK.SecurityLevel]"Unrestricted")
9. $setting.Save()
```

Line 1 gets our reference to the GPO.
Line 2 gets a reference to the setting under the SRP policy Security Levels container called "Disallowed."
Lines 3 and 4 set the "Default property on this setting to true and save it to the GPO.
Line 5 then gets a reference to the container object for Path-based rules. By default, this container will not contain any settings, so we need to add the setting using the AddNew() method on the Settings property of the container.
Line 6 does this, and we pass in the path that we want to control—in this case we are using the environment variable %programfiles% to refer to the program files folder but we could have just as easily used a literal path (e.g. "c:\program files") or a registry path rule (described here: http://technet.microsoft.com/en-us/library/cc736814.aspx).
Line 7 sets the Name property of the $setting variable since we assigned the new setting. Typically this is set to the same name as the value that was just added (in Line 6).
Line 8 sets the SecurityLevel property to Unrestricted (the choices are "Unrestricted", "Disallowed" or "BasicUser")
Line 9 saves the rule to the GPO.

Hash Rules are similar to Path rules, with the exception that the rule item you are adding points to a file, and the hash is calculated on that file and stored in the policy. The following script describes how you can create a Hash Rule to disallow the use of Regedit.exe (even if it were renamed):

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security
   Settings/Software Restriction Policies/Security Levels/Unrestricted");
3. $setting.Put("Default",$true)
4. $setting.Save()
5. $container = $gpo.GetObject("Computer Configuration/Windows Settings/Security
   Settings/Software Restriction Policies/Additional Rules Hash");
6. $setting = $container.Settings.AddNew("c:\windows\regedit.exe")
7. $setting.Name = "c:\windows\regedit.exe"
8. $setting.Put("SecurityLevel",[GPOSDK.SecurityLevel]"Disallowed")
9. $setting.Save()
```

The main difference in this script is that the container path in Line 5 changes from "Additional Rules Path" to "Additional Rules Hash."

**NOTE: Application Control Policies ("AppLocker") are not currently support in GPAE.**

### *SRP Rule Properties*

GPAE supports the following properties on a given SRP rule (e.g. using the sample script above, we can issue a $setting.GetPropertyNames() to see these three properties):

**RestrictionType (read-only)**

A read-only property that returns the type of rule this represents. Possible values for this property include:

- Hash
- Certificate
- InternetZone
- Path

**SecurityLevel (read-write)**

Set the level for a given rule, as shown in Line 8 above. Possible values for this property are represented within the [GPOSDK.SecurityLevel] enumeration, which includes the following members:

- BasicUser
- Disallowed
- Unrestricted

Thus, a member of this enumeration is referenced in Powershell using the syntax: *[GPOSDK.SecurityLevel]"BasicUser"*

**Description (read-write)**

A freeform text property to describe the purpose of a particular rule.

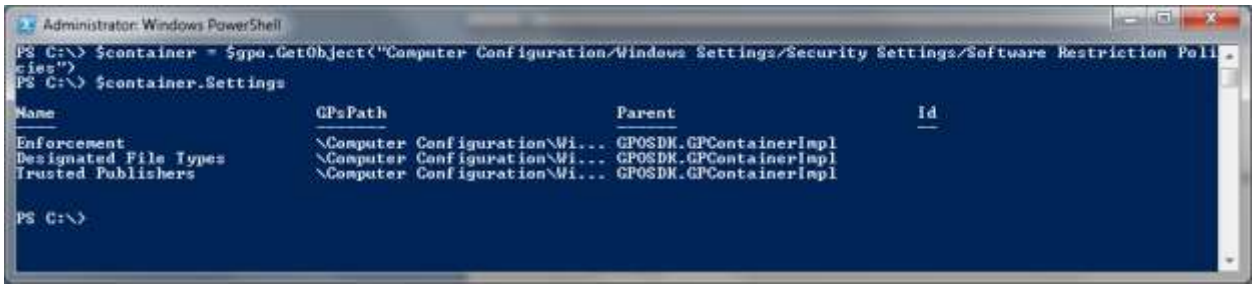
### *General SRP Properties*

There are also a number of properties that you can control outside of rules. For example, we saw in Line 2 of the script sample above that you can set the default Security Level. The figure below shows that you can also control Enforcement, Designated File Types and Trusted Publishers.

In order to access these general properties, you get a reference to the Software Restriction Policies container, like this:

*$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Software Restriction Policies");*

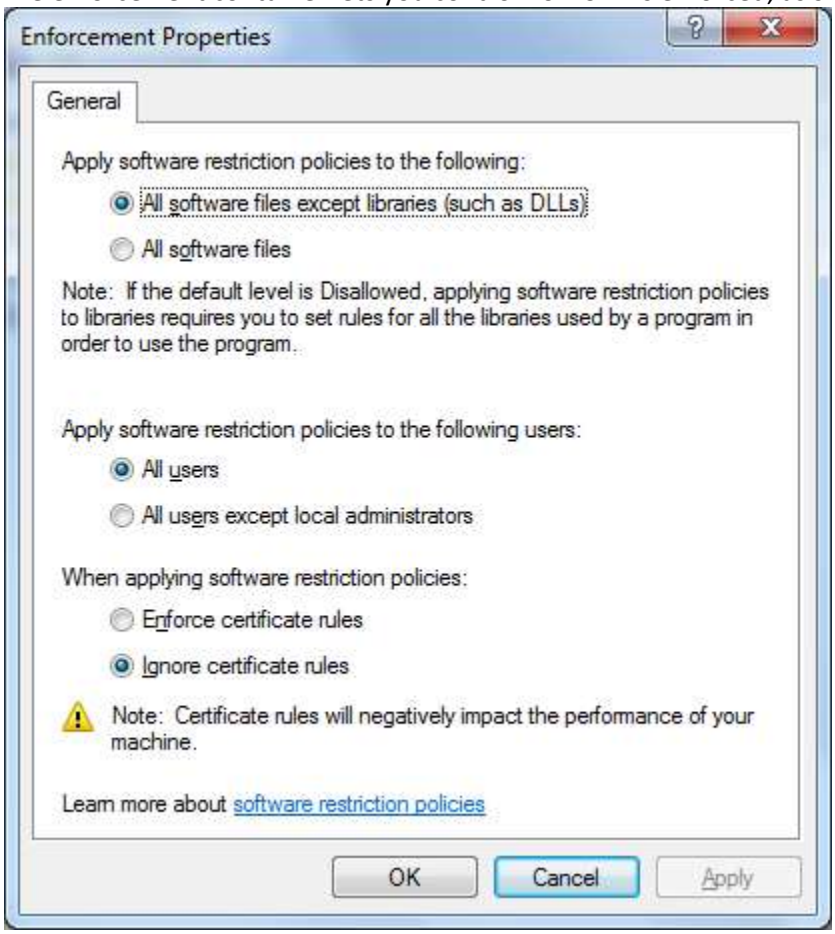
If I view the Settings property on this container, they contain 3 setting objects, as shown here:




Let's go through each and describe the options:

**Enforcement**

The enforcement container lets you control how SRP is enforced, as shown in the following screen:



Of these options, GPAE supports controlling the first two enforcement types, but not the "certificate rules" enforcement type. The following GPAE enumerations correspond to the options in the screen above:

**"Apply software restriction policies to the following"**

| Option | GPAE Enumeration |
|---|---|
| All software files except libraries | [GPOSDK.ApplyToSoft]"ExceptLibraries" |
| All software files | [GPOSDK.ApplyToSoft]"AllSoftware" |

**"Apply software restriction policies to the following users"**

| Option | GPAE Enumeration |
|---|---|
| All Users | [GPOSDK.ApplyToUser]"AllUsers" |
| All software files | [GPOSDK.ApplyToUser]"ExceptLocalAdmin" |

Since the Enforcement property is treated as a setting, it has properties of its own, which correspond to the two options above, that we support in GPAE. These properties are:
**ApplyToSoft**
**ApplyToUser**

Here's an example of how we can set the ApplyToSoft property on the Enforcement setting:

```
$container = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Software Restriction Policies");
$setting = $container.GetObject("Enforcement")
$setting.Put("ApplyToSoft",[GPOSDK.ApplyToSoft]"AllSoftware")
$setting.Save()
```

**Designated File Types**
Designated file types let you control what file extensions are controlled by SRP rules. By default, Windows includes a number of these, but you can add custom file types to the list. The Designated File Types setting in GPAE contains only one property—**FileTypes**—that stores the list of file extensions in a string array. Use the [GPOSDK.PropOp] enum to edit members of the FileTypes array. For example, if we want to add the .vbs and .ps1 script extensions to the list of designated file types, we can use the following script sample:

```
$setting = $gpo.GetObject("Computer Configuration/Windows Settings/Security Settings/Software Restriction Policies/Designated File Types");
$newtypes = "VBS","PS1"
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND","FileTypes",$newtypes)
$setting.Save()
```

Similarly, if we want to remove those two file extensions, we would use the following modification to the script code above:

```
$setting.PutEx([GPOSDK.PropOp]"PROPERTY_DELETE","FileTypes",$newtypes)
$setting.Save()
```

## Windows Firewall with Advanced Security

GPAE currently doesn't support Windows Firewall with Advanced Security policy. However, Microsoft does provide PowerShell cmdlets to manage WF policy. This is described here: https://technet.microsoft.com/en-us/library/hh831755.aspx.

## Scripts Policy

GPAE supports reading and writing into the Scripts policy area. This includes per-computer Startup and Shutdown scripts as well as per-user Logon and Logoff scripts. The following script shows how to write a logon script into a per-user GPO setting:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $setting = $gpo.GetObject("User Configuration/Windows Settings/Scripts/Logon")
3. $script = new-object "GPOSDK.ScriptEntry"
4. $script.Name = "\\dc1\netlogon\logon.vbs"
5. $script.Parameters = "-force"
6. $setting.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND", "Items", $script)
7. $setting.Save()
```

Line 1 connects to our GPO.
Line 2 connects to the container within the GP namespace for Logon scripts (note that to create a Startup Script entry, you would refer to "Computer Configuration/Windows Settings/Scripts/Startup" instead).
Line 3 creates a new object specific to GPAE called a ScriptEntry object. This object contains two properties, called "Name" and "Parameters." The name points to the name of the script while the Parameters property is an array that contains one or more parameters that you wish to pass to the script. These parameters are entered within the GPO scripts policy as shown in Figure 11 below:

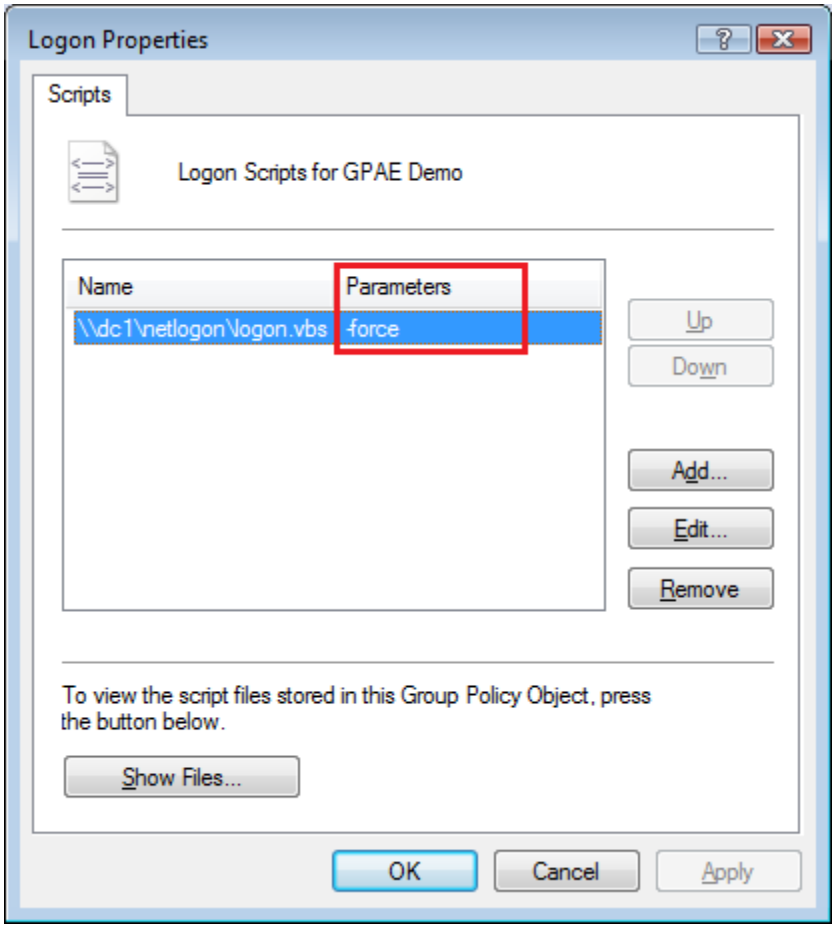



**Figure 11: Using parameters in Scripts Policy**

Lines 4 & 5 set the Name and Parameters properties for the new script. Note that Name property should point to the path where the actual script file is stored. By default, there are two choices. You can either store the script file in an external network location, such as that shown in the example above, or you can store the script within the actual GPO's SYSVOL storage area. In either case, GPAE does not copy your script file to the correct location—this must be done either manually or by a separate script operation. GPAE just creates the reference to the script's location in the GPO. However, to ease the process of copying a script file into the SYSVOL storage of the GPO, the $setting object created above contains a read-only property called "DefaultPath," which, when called, returns the full path to the location in the GPO where the script file should be copied, as shown in Figure 12 below:


**Figure 12: Using the DefaultPath property to find the location in the GPO to copy the script files**

Line 6 uses the PutEx() method on the $setting object to populate the script reference into the GPO. In this case, $setting contains a multi-valued property called “Items” that we are populating with the $script object. Note that we are using the [GPOSDK.PropOp]"PROPERTY_APPEND" enum value to update the Items property rather than wiping out what may already be there. Also note that the Items property can hold multiple script references so this operation is the safest to ensure that we are not removing any existing script references.
Line 7 saves the script reference to the GPO.

This process is identical whether you are creating per-user logon or logoff scripts or per-computer startup or shutdown scripts. The key difference is in Line 2: the reference that you create to the appropriate container within the GPO. And it’s important to remember that GPAE does not copy your script file into the GPO—you must do this manually or reference the script file to someplace external to the GPO.

## Deployed Printers

The Deployed Printers policy is a Windows® Vista/Windows 7/Windows Server 2008 policy feature and as such should not be confused with the printer mapping feature available in GP Preferences. However, GPAE supports defining either per-computer or per-user Deployed Printers policy. The following script shows an example of how to define a per-user Deployed Printer:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $container = $gpo.GetObject("User Configuration/Windows Settings/Deployed Printers")
3. $printer = $container.Settings.AddNew("HP Laser Jet 3055 PCL5")
4. $printer.Put("ServerName","\\SDM2")
5. $printer.Save()
```

Line 1 connects to the GPO.
Line 2 connects to the per-user Deployed Printers container
Line 3 adds the share name of our new printer. Note that this is the name that the printer is shared out under on our print server. That is usually the name of the print driver to install but you can change it when you share the printer.
Line 4 sets the ServerName property on the $printer object to the UNC path of our print server—in this case it’s \\SDM2.
Line 5 saves the printer definition to the GPO.

If we want to retrieve printer definitions, it’s a simple matter of calling the Get() method on each printer object within the Deployed Printers container. The printer name is contained within the “Name” property and, as shown above, the print server in use is contained within the “ServerName” property.

## Software Installation Policy

GPAE supports creating Software Installation policies for either per-computer or per-user deployments. Software Installation policy requires packages to be deployed to come in the Windows Installer (.MSI) packaging format; so, this is supported in GPAE. The following script gives an example of how to create a per-computer deployment of Adobe Acrobat Reader:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
```

2. *$container = $gpo.GetObject("Computer Configuration/Software Settings/Software Installation");*
3. *$package = $container.Settings.AddNew("Adobe Acrobat Reader 8")*
4. *$package.Put("uncPath", "\\dc101\packages\Acroread8\Acroread.msi")*
5. *$package.Put("type", [GPOSDK.InstallPackageType]"Assigned")*
6. *$package.Save()*
7. *$currentFlags = $package.Get("flags")*
8. *$package.Put("flags",($currentflags –bor [GPOSDK.InstallPackageFlags]"PR_UNINSTALL_ON_REMOVE"));*
9. *$package.Save()*

Line 1 connects to our GPO.
Line 2 gets a reference to the per-computer Software Installation container.
Line 3 creates the new package by calling the AddNew() method on the $container.Settings property. In this case, the name we enter is the name that appears in the GPO and on the client, for the package to be installed (e.g. “Adobe Acrobat Reader 8”).
Line 4 modifies the uncPath property on the $package object to tell it from where to get our MSI file. Note that in this case, we are entering a UNC path—the path entered here must make sense to the clients that will be installing the package, hence a UNC path (and ideally a Distributed File System (DFS) path should be used to point to the package location).
Line 5 uses the GPOSDK.InstallPackageType enum to tell the script what kind of deployment we want to create (the choices are “Assigned,” “Published,” and “Removed”). Note that only Assigned and Removed are available for per-computer packages while all three options are available for per-computer packages.
Line 6 saves the package deployment. After the Save () method call occurs, you will notice a brief Windows installer dialog appear on the screen where the script is running. This is normal and represents a call to the Windows installer to create the Application Advertisement Script that is associated with GP-based packages. Once the Save is performed the package is now available in the GPO.

We may want to alter the default package flags that are available once the package is deployed. The package flags control the behavior options you see when you are viewing the package in GP Editor. For example, by default the package is created with the behavior “Uninstall this application when it falls out of the scope of management” unchecked, meaning that the application is left in place on the client if the GPO no longer applies. But if we want to change that behavior, we have to modify the package flags on the package after we create it. Note that this process must happen after the package has been initially saved, rather than during the initial package creation process. This is the reason that we save the package in Line 6, then procede to modify the flags.
Line 7 starts the process of modifying the current flags by using the Get() method on the $package object to get the current value of the “flags” property.
Line 8 uses Put() to set the flags property to the new value, which is the current flags value boolean Or’d (-bor) with the flag we want to add. In this case, we call the GPOSDK.InstallPackageFlags enum with the appropriate flag we wish to add.
Line 9 saves the package again to commit our flag changes.

The following table lists all of the package flag options available and what they mean.

| Option | GPAE Enumeration |
|---|---|

| | |
|---|---|
| The unmanaged version of this application must be uninstalled before assigning. | [GPOSDK.InstallPackageFlags]” PF_UNINSTALL_UNMANAGED” |
| This is a published application | [GPOSDK.InstallPackageFlags]”PF_PUBLISHED” |
| This package was deployed after Beta 3 of Windows 2000. | [GPOSDK.InstallPackageFlags]”PF_BETA3_W2K” |
| This application can be auto-installed (advertised) on demand. | [GPOSDK.InstallPackageFlags]”PF_AUTO_INSTALL” |
| This application is orphaned. A published application can become orphaned if the administrator removes the application from the list of deployable applications. | [GPOSDK.InstallPackageFlags]”PF_ORPHANED” |
| This application should be treated as uninstalled. | [GPOSDK.InstallPackageFlags]”PF_TREAT_UNINSTALLED” |
| This application is available as a pilot only. | [GPOSDK.InstallPackageFlags]”PF_PILOT_ONLY” |
| This is an assigned application. An assigned application cannot be fully removed by the user. If the user attempts to uninstall the application with the **Add or Remove Programs** feature of the Control Panel, the application will be re-advertised to the computer when the removal completes. | [GPOSDK.InstallPackageFlags]”PF_ASSIGNED” |
| This application is orphaned when the policy is removed. If the administrator removes the policy that is related to this application, the administrator will no longer control the deployment of the application, but the installed application will continue to function. | [GPOSDK.InstallPackageFlags]”PF_ORPHANED_ON_REMOVE” |
| This application is uninstalled when the deployment policy is removed. This is the opposite of the previous flag. | [GPOSDK.InstallPackageFlags]”PR_UNINSTALL_ON_REMOVE” |
| A full installation of user-assigned applications will be performed. | [GPOSDK.InstallPackageFlags]”PF_FULL_INSTALL” |
| Older versions of this application must be upgraded to this version. | [GPOSDK.InstallPackageFlags]”PF_UPGRADE” |
| This package supports only a minimal user interface with a progress bar for the installation process. | [GPOSDK.InstallPackageFlags]”PF_MINIMAL_UI” |
| This is a package for 32-bit versions of Windows that should not be executed on Windows XP Professional x64 | [GPOSDK.InstallPackageFlags]”PF_32_BIT_ONLY” |

| | |
|---|---|
| Edition or 64-bit versions of Windows Server 2003. | |
| This package is suitable for any language. | [GPOSDK.InstallPackageFlags]”PF_ANY_LANGUAGE” |
| This package has upgrades. | [GPOSDK.InstallPackageFlags]”PF_HAS_UPGRADES” |
| This package has a full user interface for the installation process. | [GPOSDK.InstallPackageFlags]”PF_FULL_UI” |
| Classes for this application are preserved during a redeploy operation if the application is redeployed in a domain renaming. | [GPOSDK.InstallPackageFlags]”PF_PRESERVE_CLASSES” |
| Unknown Flag | [GPOSDK.InstallPackageFlags]”PF_UNKNOWN” |

**Note that the package flags supported in GPAE are identical to those supported in the native interfaces—more detail can be found at http://msdn.microsoft.com/en-us/library/ms679099(VS.85).aspx.**

We can also retrieve package information and properties on packages, by calling into the software installation container. Package objects contain the following properties shown in Table 1.

| **Package Properties** |
|---|
| name |
| type |
| flags |
| platform |
| uncPath |
| productCode |
| security |
| phone |
| packagesUpgraded |
| locale |
| aasFilePath |
| url |
| version |
| installUiLevel |
| transformModifications |
| publisher |
| deploymentCount |

**Table 1: Software Installation Package Properties**

For example, given that we have defined the $container variable in the previous script to point at the per-computer Software Installation container, the following one-line PowerShell command will iterate through all packages and return their name, install type (e.g. Assigned or Published) and publisher name:

```
foreach ($package in $container.Settings) {$package.Name, $package.Get("type"), $package.Get("publisher")}
```

## Folder Redirection Policy

Folder Redirection policy provides the ability to redirect several well-known user profile folders (e.g. My Documents, Start Menu, Desktop) to alternate locations—usually server shares. Folder Redirection policy in Vista and Server 2008 supports a superset of the folders that are supported on Windows XP and Server 2003, and so GPAE supports writing both types, including the ability to create the downlevel files necessary for Folder Redirection policy created from Windows Vista or Server 2008 to apply to downlevel clients.

Folder Redirection policy supports two modes—basic and advanced. In basic mode, all users that receive the policy are redirected to the same location, while in advanced mode, users can be redirected to different locations depending on their group membership. For example, you might redirect the My Documents folder for users within the "Marketing Users" group to a server that is dedicated to the Marketing department while redirecting members of the Sales Users group to a dedicated Sales department server. Here is an example of GPAE supporting both basic and advanced redirection:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $container = $gpo.GetObject("User Configuration/Windows Settings/Folder Redirection")
3. $setting = $container.Settings.ItemByName("Application Data")
4. $setting.Put("Behavior",[GPOSDK.FolderRedirectionBehavior]"Simple")
5. $setting.Put("Basic_FolderPath", "\\DC101\Appdata\%username%\AppData")
6. $setting.Put("Redirection_BackOnRemoval", $true)
7. $setting.Save()
```

Lines 1 & 2 of the script connect to the GPO and create a reference to the Folder Redirection container. Line 3 creates a reference to the "Application Data" setting for the purposes of modifying its properties. Line 4 tells the policy that we want to use Simple (i.e. Basic) redirection. The GPAE enumeration **[GPOSDK.FolderRedirectionBehavior]** supports the following values:

| Option | GPAE Enumeration |
|---|---|
| Redirection has no option defined (i.e. default folder location applies) | [GPOSDK.FolderRedirectionBehavior]"Undefined" |
| Simple Redirection | [GPOSDK.FolderRedirectionBehavior]"Simple" |
| Advanced Redirection (by security group) | [GPOSDK.FolderRedirectionBehavior]"Advanced" |
| This folder should be redirected the same way as its parent folder | [GPOSDK.FolderRedirectionBehavior]"FollowParent" |
| Redirection has no option defined (i.e. default folder location applies) | [GPOSDK.FolderRedirectionBehavior]"Default" |

Line 5 modifies the "Basic_FolderPath" property to tell the policy where we want to redirect users' AppData folders to. In this example, we are redirecting each user's Application Data folder to a share called Appdata, and instructing the policy to create a username folder under that share for each user, and then to store the Application Data contents in a folder called Appdata.
Line 6 sets the behavior on the policy to redirect (i.e. copy) the contents of Application Data back to the user's profile if the policy delivering this redirection no longer applies.
Line 7 saves the changes to the GPO.

GPAE supports redirection of all folders supported up to and including Vista and Server 2008. However, when running a GPAE script on a Windows XP or Server 2003 box, use only those folders that are supported in those versions. The following table lists which folders are supported in each version.

| Folder | Supported On |
|---|---|
| Application Data | All Versions |
| Desktop | All Versions |
| Documents | All Versions |
| Pictures | All Versions |
| Start Menu | All Versions |
| Music | Vista, Server 2008 and above |
| Videos | Vista, Server 2008 and above |
| Favorites | Vista, Server 2008 and above |
| Contacts | Vista, Server 2008 and above |
| Downloads | Vista, Server 2008 and above |
| Searches | Vista, Server 2008 and above |
| Saved Games | Vista, Server 2008 and above |
| Links | Vista, Server 2008 and above |

**Table 2: Folder Redirection support by version**

Advanced redirection lets you specify a different target location based on user-group membership. The following script shows how to do this:

```
1.  $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2.  $container = $gpo.GetObject("User Configuration/Windows Settings/Folder Redirection")
3.  $folder = $container.Settings.ItemByName("Documents")
4.  $folder.Put("Behavior",[GPOSDK.FolderRedirectionBehavior]"Advanced")
5.  $new_obj1 = new-object "GPOSDK.FolderRedirectionEntry"
6.  $new_obj1.UserSid = "S-1-5-21-817735531-4269160403-1409475253-1108"
7.  $new_obj1.FolderPath = "\\member100\Public\Marketing"
8.  $folder.PutEx([GPOSDK.PropOp]"PROPERTY_UPDATE", "Advanced_GroupsSettings", $new_obj1)
9.  $folder.Save()
10. $new_obj2 = new-object "GPOSDK.FolderRedirectionEntry"
11. $new_obj2.UserSid = "S-1-5-21-817735531-4269160403-1409475253-3132"
12. $new_obj2.FolderPath = "\\member100\Public\Sales"
13. $folder.PutEx([GPOSDK.PropOp]"PROPERTY_APPEND", "Advanced_GroupsSettings", $new_obj2)
14. $folder.Save()
```

Lines 1 & 2 connect to the GPO and then connect to the Folder Redirection container.
Line 3 gets a reference to the Documents setting.
Line 4 sets the Behavior property on the Documents folder to advanced redirection.
Line 5 departs from the basic folder redirection script, as we create a new object of type GPOSDK.FolderRedirectionEntry. This is because we want to create a new advanced redirection object, which is composed of the UserSid and FolderPath properties.
Line 6 sets the UserSid property to the SID of the user group to be redirected.
Line 7 sets the FolderPath property on the FolderRedirectionEntry object to UNC path where we want our group redirected.

Line 8 updates the Advanced_GroupSettings property on the Documents folder with the object we just created.
Line 9 saves the new object to the $folder object, using the PROPERTY_UPDATE enum. This assumes that we wish to overwrite any existing entries that might already be in the Documents redirection.
Lines 10-14 do the same thing for a second user group and folder location with a new object, but in this case we append to the existing list using the PROPERTY_APPEND enum because we know we already have a setting within it.

## IE Maintenance Policy

IE Maintenance policy contains a number of different settings related to controlling IE. Note that Microsoft has deprecated IE Maintenance policy for systems running IE 10 or above. If you have IE 10 on your system, you will no longer see the IE Maintenance Policy node within GP Editor on a given GPO. However, GPAE is still capable of reading and writing IE Maintenance settings on these systems. For example, it's within this policy area that you can set Browser title, home page and proxy settings. You can also set options such as Security Zone settings. GPAE supports most, but not all areas within IE Maintenance. The notable piece that is missing is the ability to read and write the Content Ratings sections.
Let's look at some example scripts that show how to set various IE Maintenance Policies. The following script sets the IE home and search pages:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $container = $gpo.GetObject("User Configuration/Windows Settings/IE Maintenance/URLs")
3. $IESettings = $container.Settings.ItemByName("PROPERTIES")
4. $IESettings.Put("HomePage","http://www.sdmsoftware.com")
5. $IESettings.Put("SearchPage","http://www.live.com")
6. $IESettings.Save()
```

Lines 1 & 2 get our GPO reference and a reference to the container within IE Maintenance policy that stores URLs.
Line 3 creates a reference to the Settings collection under $container, and we get a property called "PROPERTIES." This seems a bit confusing but in this case, we are storing URL properties within the PROPERTIES collection on the setting object. We'll look at some other properties of $container in the next examples.
Line 4 puts www.sdmsoftware.com into the "HomePage" property on the $IESettings setting.
Line 5 puts the "SearchPage" property.
Line 6 saves the settings.

Adding favorites to IE is where we use some alternative properties on the URLs container:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $container = $gpo.GetObject("User Configuration/Windows Settings/IE Maintenance/URLs")
3. $IESettings = $container.Settings.ItemByName("PROPERTIES")
4. $IESettings.Put("PlaceFavoritesOnTop",$true)
5. $IESettings.Save()
6. $fav = $container.Settings.ItemByName("Favorites")
7. $TitlesList = @("My Company Intranet","Time and Expense App","Vacation Request System")
```

8. *$URLsList = @("http://www.sdmsoftware.com/intranet","http://www.sdmsoftware.com/TimeTracking","http://www.sdmsoftware.com/vacation");*
9. *$icons = @()*
10. *$fav.PutEx([GPOSDK.PropOp]"Property_Update","Titles",$TitlesList)*
11. *$fav.PutEx([GPOSDK.PropOp]"Property_Update","URLs",$URLsList)*
12. *$fav.PutEx([GPOSDK.PropOp]"Property_Update","Icons",$icons)*
13. *$fav.Save()*

Lines 1 & 2 get our GPO reference and a reference to the container within IE Maintenance policy that stores URLs.
Line 3 gets a reference to the PROPERTIES property on the URLS container. This is so we can set a property called “PlaceFavoritesOnTop” that corresponds to the GP Editor option called “Place favorites and links at the top of the list in the order specified below.”
Line 4 sets this property to $true.
Line 5 saves it to the policy.
Line 6 creates a reference called $fav to the “Favorites” setting on the $container object, so that we can add our Favorites.
Lines 7, 8 and 9 build arrays that create the lists that we will use to populate our Favorites.
Line 7 creates a list of nametags that define how the favorite will read in IE.
Line 8 creates a corresponding array that populates the URLs for the favorite names.
Line 9 creates a reference to the Icons that belong to a given Favorite. In this case, we aren’t specifying any icons but we still need to create the array and use it to populate our favorites.
Lines 10, 11 and 12 use the PROPERTY_UPDATE enum to update each property of the favorites with our arrays.
Line 13 saves them to the GPO.

The following script specifies proxy settings within IE Maintenance policy:

1. *$gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName*
2. *$container = $gpo.GetObject("User Configuration/Windows Settings/IE Maintenance/Connection");*
3. *$conn = $container.Settings.ItembyName("PROPERTIES")*
4. *$conn.Put("ProxyEnabled",$true)*
5. *$conn.Put("BypassLocal",$true)*
6. *$conn.Put("HTTP_Proxy","192.168.10.1:80")*
7. *$conn.Put("UseSameProxyForAll",$true)*
8. *$conn.Save()*

Lines 1 & 2 connect to our GPO and then connect to the Connection container under IE Maintenance policy.
Line 3 gets a reference to the PROPERTIES setting under that container.
Lines 4, 5, 6, and 7 put the various properties within that PROPERTIES setting. These properties represent the configurable behavior for the proxy settings in IE Maintenance policy. The “HTTP_PROXY” property sets the IP address and port for HTTP traffic, and the “UseSameProxyForAll” property tells IE Maintenance to use that same address:port combination for all other protocols (e.g. FTP).
Line 8 saves the proxy settings to the GPO.

Alternatively, we could have set them each separately. For example, if you type $conn.GetPropertyNames(), you will see all properties available for this setting, as shown in Table 3:

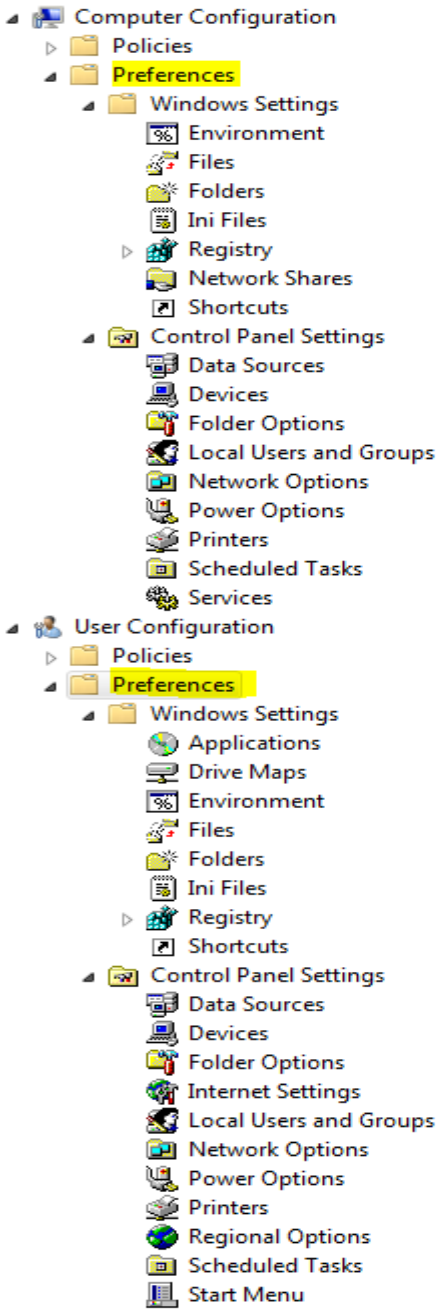
| PropertyName |
|---|
| AutoDetect |
| AutoConfig |
| AutoConfigPeriod |
| AutoConfigURL |
| AutoConfigJSURL |
| ProxyEnabled |
| UseSameProxyForAll |
| HTTP_Proxy |
| HTTPS_Proxy |
| FTP_Proxy |
| GopherProxy |
| SocksProxy |
| ProxyBypassList |
| BypassLocal |
| ImportSettingsFromThisMachine |
| DeleteExistingDialup |
| AutoDetect |

**Table 3: IE Maintenance Connection properties**

## Group Policy Preferences

With Group Policy Preference (GPP) support, you can automate some very powerful GP Settings capabilities. GPP requires an additional installation on client systems that are processing Group Policy, but once this is done; there is a wide variety of capabilities that GPP brings. GPAE supports all of those features (most notably, reading and writing GPP settings from Windows XP and Server 2003 platforms—something not supported by Microsoft in native tools).

GPP contains dozens of policy areas that you can control, as shown in the screenshot below:




GPAE supports all of these GP Preferences policy areas in the current release. In addition, we've added support for IE Settings policies up to and including IE 10/11, starting with the 3.0 version of GPAE. Using GPAE to manage each of these preferences areas is fully documented in the **GPAE Object Reference Help** file, found in the Start Menu Program Group for GPAE. However, we include a couple of examples of more popular GPAE areas here to help get you started.

## GPP Drive Mapping

The approach we show in our examples below are applicable to all GPP areas. Let's start by looking at some of the more popular areas within GPP that you may want to automate, starting with Drive Mappings, found under User Configuration/Preferences/Windows settings/Drive Maps.

The following script shows how you can create drive mappings using GPP:

```
1. $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2. $container = $gpo.GetObject("User Configuration/Preferences/Windows settings/Drive
   Maps");
3. $map = $container.Settings.AddNew("P Drive")
4. $map.Put("Action",[GPOSDK.EAction]"Replace")
5. $map.Put("Drive Letter","P”)
6. $map.Put("Location","\\sdm1\public")
7. $map.Put("Reconnect", $true)
8. $map.Put("Remove this item when it is no longer applied", $true)
9. $map.Put("Order",1)
10. $map.Put("Label as", "SDM Public Drive")
11. $map.Save()
```

Lines 1 & 2 connect to our GPO and then connect to the Drive Maps container under Windows settings.
Line 3 creates the $map object by adding a new setting to the $container.Settings collection. Note that this step, calling AddNew() on the container, is identical across all GPP areas within GPAE. You are required to create the setting with a name before you can operate on it.
Line 4 sets the GPP action type for this setting. GPP supports Create, Delete, Update and Replace. Each causes a different behavior from the client processing the policy. The GPAE enum **[GPOSDK.EAction]** supports the following members:

- Create
- Replace
- Update
- Delete

We set this drive mapping to Replace because later in the script we set the “Remove this item when it is no longer applied" property to true, which requires the action type to be “Replace.”
Line 5 sets the drive letter we want to use.
Line 6 sets the UNC path to the drive mapping.
Line 7 sets the property to Reconnect the drive mapping automatically.
Line 8 sets the option we mentioned earlier, that will remove the mapping from the client if the GPO no longer applies.
Line 9 sets the order of this policy, since we could have multiple drive mappings in a given GPO and you might want to have them ordered in a particular way. Note that the ordering index starts at 1, rather than 0.
Line 10 sets the “Label As” property for the mapping.
Line 11 saves the mapping to the GPO.

### *Item-Level Targeting*

Item-Level Targeting (ILT) lets you control who receives which GPP settings at a per-setting (per-item) level. This feature is specific to GPP, and is supported in GPAE. The most common example is using ILT to apply a particular drive mapping to a particular user group. ILT supports filtering based on a wide variety of criteria, as shown in Figure 13, below.

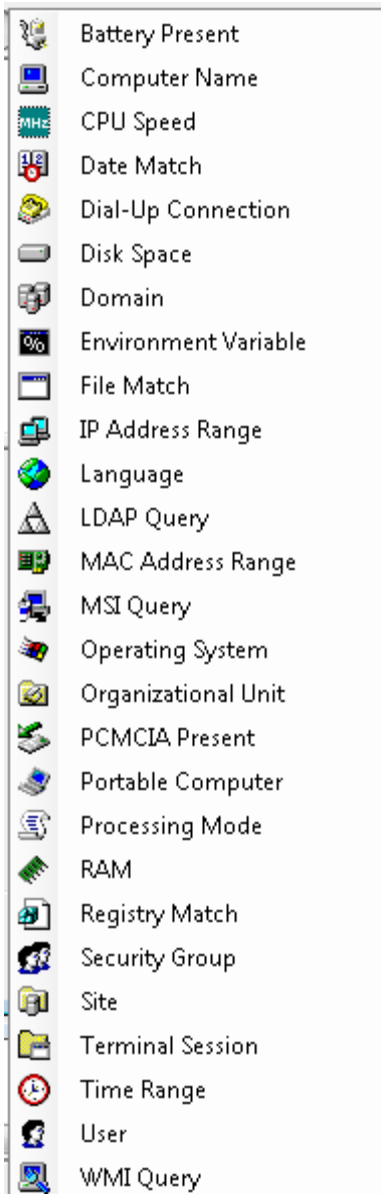



**Figure 3: Viewing the ILT Options**

GPAE supports creating ILT filters for any of the criteria shown above. Let's look at the example of creating a security group-based ILT for the drive mapping we created in the previous script. The following script shows how we would add a couple of security groups and a user account filter to that drive mapping:

```
1. $iilt = $gpo.CreateILTargetingList()
2. $itm = $iilt.CreateIILTargeting([GPOSDK.Providers.ILTargetingType]"FilterGroup")
3. $itm.Put("Group","Marketing Users")
4. $itm.Put("UserInGroup", $true)
5. $objGroup = New-Object System.Security.Principal.NTAccount("cpandl.com","Marketing
   Users");
6. $strSID = $objGroup.Translate([System.Security.Principal.SecurityIdentifier])
7. $itm.Put("SID",$strSID.value)
```

```
8.  $iilt.Add($itm)
9.  $itm = $iilt.CreateILTargeting([GPOSDK.Providers.ILTargetingType]"FilterGroup")
10. $itm.Put("Group","Sales Users")
11. $itm.Put("UserInGroup", $true)
12. $objGroup2 = New-Object System.Security.Principal.NTAccount("cpandl.com","Sales Users")
13. $strSID2 = $objGroup2.Translate([System.Security.Principal.SecurityIdentifier])
14. $itm.Put("SID",$strSID.value)
15. $itm.and = $false
16. $iilt.Add($itm)
17. $itm = $iilt.CreateILTargeting([GPOSDK.Providers.ILTargetingType]"FilterUser")
18. $itm.Put("User","Darren")
19. $objUser = New-Object System.Security.Principal.NTAccount("cpandl.com","Darren")
20. $strSID3 = $objUser.Translate([System.Security.Principal.SecurityIdentifier])
21. $itm.Put("SID",$strSID3.value)
22. $itm.not = $true
23. $iilt.Add($itm)
# Now save filters to current map setting
24. $map.Put("Item-level targeting", $iilt)
25. $map.Save()
```

Line 1 creates an instance of an ILTTargetingList collection object by calling the CreateILTargetingList() method on the GPO. This object holds all of the ILT filters that we are going to create.
Line 2 calls the CreateILTargeting() method on this new created object to create a new instance of an ILT filter. In this call, we tell GPAE which type of ILT filter we want to create. In this case, we are creating a "FilterGroup" type ILT to filter by user group. The GPOSDK.Providers.ILTargetingType enumeration supports all of the different ILT types within GPP, as shown in Table 4 below:

| **Filter Types** |
|---|
| FilterCollection |
| FilterBattery |
| FilterComputer |
| FilterCpu |
| FilterDate |
| FilterDun |
| FilterDisk |
| FilterDomain |
| FilterVariable |
| FilterFile |
| FilterIpRange |
| FilterLanguage |
| FilterLdap |
| FilterMacRange |
| FilterMsi |
| FilterOs |
| FilterOrgUnit |
| FilterPcmcia |
| FilterPortable |

| |
|---|
| FilterProcMode |
| FilterRam |
| FilterRegistry |
| FilterGroup |
| FilterSite |
| FilterTerminal |
| FilterTime |
| FilterUser |
| FilterWmi |
| FilterCollection |

**Table 4: The different ILT Filter types**

Lines 3 & 4 update the properties “Group” and “UserInGroup” on our user group filter.
Lines 5, 6 and 7 get the SID of the user group that is part of this filter. Providing the SID is required by GPAE to build the ILT for groups.
Line 5 gets a special .Net type called NTAccount and uses that to build an object that represents the group. The first parameter is the DNS AD domain name that the group is in, and the second parameter is the group name.
Line 6 translates that object to a SID type of object.
Line 7 assigns the SID value to the SID property on the ILT.
Line 8 adds this newly created filter to our ILT collection.
Lines 9 through 15 do the same thing as in the case of the first group, using a second group name (Sales Users) with one notable exception: Because ILTs have the ability to be combined using Boolean operators, we have to tell the GPO how we want this second group to be dealt with. We call $itm.and = $false to set the “and” property on the group filter to be false, meaning that it should be OR’d with the first group (i.e. the user can be in Group 1 OR Group2 to pass this filter).
Line 16 adds this filter to the ILT collection.
Lines 17-23 use a different filter type—in this case a user account name, to include in our filter. However, in this case, you’ll notice that in Line 23, we are setting the “not” property on the item to true. The result of this combination of groups and users, with logical operators, is the filter shown in Figure 4 below.

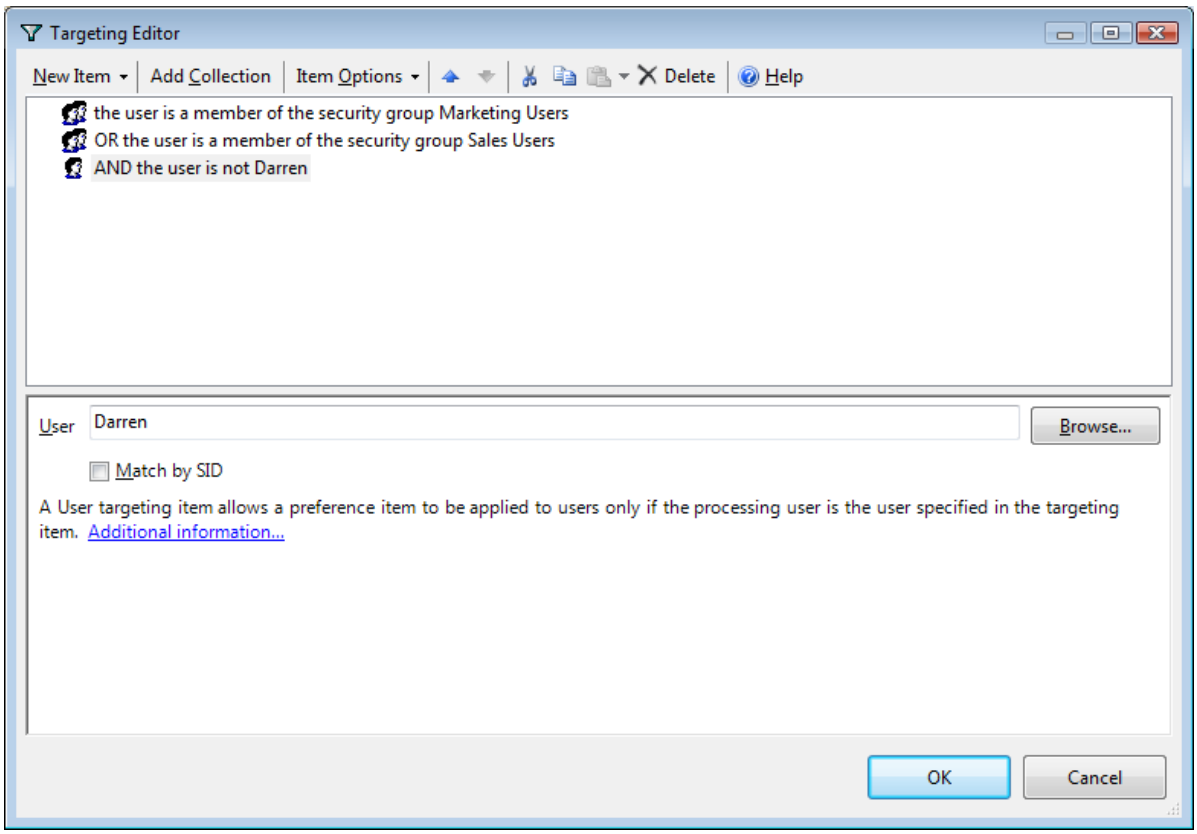



**Figure 4: Viewing a complex ILT filter created with GPAE**

Line 24 puts the "Item-level targeting" property on our $map object that represents the drive mapping we created in the previous script, with the $iilt object that holds all of our filters
Line 25 saves the drive mapping policy to the GPO.

### *Retrieving Item-Level Targeting Settings*

The key thing to know about Item-Level Targeting items and GPAE is that GPAE stores ILTs as a collection object. Specifically, a given GP Preference setting that contains one or more ILTs will be stored as a collection within the **"Item-level targeting"** property.
To retrieve the values of existing Item-Level Targeting items, you would need to iterate into the collection, and pull properties from each Item. Because a given item-level targeting item could be using a different filter type (e.g. group, battery, user, LDAP query, etc.) each filter item could have a different set of properties. Therefore a best practice is to test for the filter as you iterate into it. For example:

```
Foreach ($ilt in $setting.Get("Item-level targeting"))
{
    if ($ilt.GetInterfaceType() –eq [GPOSDK.Providers.ILTargetingType]"FilterCpu")
     { write-output $ilt.Get("SpeedMHz") }
 }
```

In this example, we've created a foreach loop to iterate through all item-level targets on a GP Preferences setting called $setting. We then call GetInterfaceType() on the current ILT in the loop and see if it equals the enumeration for the CPU filter type. If so, then we call the "SpeedMHz" property on

that ILT to get its value. Similarly, if we were searching a filter of type FilterGroup, we would be able to retrieve properties such as ”Group,” ”UserInGroup” and ”SID.”

### Define Power Management Schemes

Let’s look at another example of creating GPP settings within GPAE. The following example shows how you can define power management schemes within the power settings capability of GPP:

```
1.  $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2.  $container = $gpo.GetObject("User Configuration/Preferences/Control Panel Settings/Power
    options/Power scheme (Windows XP)");
3.  $setting = $container.Settings.AddNew("My Custom XP Power Scheme")
4.  $setting.Put("Action", [GPOSDK.EAction]"Update")
5.  $setting.Put("Power scheme name", "My Custom XP Power Scheme")
6.  $setting.Put("Make active", $true)
7.  $setting.Put("Monitor AC", 15)
8.  $setting.Put("Monitor DC", 15)
9.  $setting.Put("Standby AC", 15)
10. $setting.Put("Standby DC", 30)
11. $setting.Put("Hibernate AC", 0)
12. $setting.Put("Hibernate DC", 0)
13. $setting.Put("HDD AC", 0)
14. $setting.Put("HDD DC", 0)
15. $setting.Save()
```

Lines 1 & 2 get a reference to the GPO and then to the container we want to manage.
Line 3, as was indicated in our previous drive mapping example, creates our new GPP item and gives it a name.
Line 4 sets the GPP action we want to use, Update, which ensures that this setting is only updated on a change.
Line 5 gives our custom power scheme a name.
Line 6 tells the client that we want this to be the active power scheme after it’s processed.
Lines 7-14 set the behaviors for this scheme. For example Monitor AC and Monitor DC correspond to when to turn off the monitor when the system is plugged in (AC) or on battery (DC), and so forth.
Line 15 saves the setting to the GPO.

Each GPP area is implemented similarly to the above two examples. To view an area’s unique set of properties, create a new entry as in Line 3 in the above script, then type:
*$setting.GetPropertyNames()*

GPAE can also retrieve settings from existing GPP policies. For example, if we were hitting the above GPO anew, to retrieve what custom power schemes were defined, type:

```
1.  $gpo = get-sdmgpobject -gpoName "gpo://cpandl.com/GPAE Demo" -openbyName
2.  $container = $gpo.GetObject("User Configuration/Preferences/Control Panel Settings/Power
    options/Power scheme (Windows XP)");
3.  $setting = $container.Settings.ItembyName("My Custom XP Power Scheme")
4.  $setting.Get("Monitor AC")
5.  $setting.Get("Monitor DC")
```

This script is basic, but you can get more creative if needed. For example, if a particular GPP area has more than one setting, these will be found in the $container.Settings collection, which you can use to iterate through each setting to find its value, as in:

```
Foreach ($item in $container.Settings){$item.Get("Monitor AC")}
```

### GPP Printer Mappings

In addition to drive mappings, GPAE is fully capable of reading and writing the different kind of GPP Printer Mappings, including Local, TCP-IP and Shared Printers. The following script sample shows how to create a simple TCP-IP Printer mapping, along with ILT for IP Address Range:

```
1.  $gpo = Get-SDMgpobject -gponame "gpo://cpandl.com/GPPDemo" -openbyName
2.  $container = $gpo.GetObject("User Configuration/Preferences/Control Panel
    Settings/Printers/TCP-IP printer")
3.  $printer = $container.Settings.AddNew("My HP 3055")
4.  $printer.put("Action", [GPOSDK.EAction]"Update")
5.  $printer.put("Address","192.168.1.190")
6.  $printer.put("Printer path","\\DriverServer\HPLaser")
7.  $printer.put("Default", $true)
8.  $printer.Save()
    # ILT for IP Address Range
9.  $iilt = $gpo.CreateILTargetingList()
10. $itm = $iilt.CreateIILTargeting([GPOSDK.Providers.ILTargetingType]"FilterIPRange")
11. $itm.Put("Min",[System.Net.IPAddress]"192.168.1.1")
12. $itm.Put("Max",[System.Net.IPAddress]"192.168.1.10")
13. $iilt.Add($itm)
14. $printer.put("Item-level targeting",$iilt)
15. $printer.Save()
```

Line 1 gets a reference to the GPO. Line 2 gets a reference to the TCP-IP Printer container within the GP Preferences container. Line 3 adds the definition for the new printer, as a setting within the container. Lines 4-7 set properties on the TCP-IP printer definition and Line 8 saves the printer definition to the GPO. Line 9 starts the definition of the ILT for an IP address range of 192.168.1.1 – 192.168.1.10. Line 10 creates an object of an ILT type. Then Line 11 creates the ILT for IP address range. Lines 12 & 13 create the Min and Max properties on the IP Address ILT and Line 14 adds the newly created ILT item to the ILT collection. Finally, Line 15 associates the ILT collection to the Item-level Targeting property on the printer we just created, and Line 16 commits the change to the GPO.

### *Renaming GPP Printers*

With the release of GPAE 3.0, we added a special cmdlet for renaming GPP Printers. This cmdlet was added because there was no easy way to rename a GPP Printer object within GPAE in previous releases. The cmdlet is called **Rename-SDMGPP_Printer**. The syntax for this cmdlet is pretty straightforward, and is shown here:

```
Rename-SDMGPP_Printer -gpoName "gpo://cpandl.com/GPPRef Printer Test"
-printerPath "User Configuration/Preferences/Control Panel
Settings/Printers/Shared Printer" -oldName "printerc" -newName
"printer123"
```

This cmdlet takes a number of straightforward parameters. The gpoName parameter is the GPAE path to the GPO containing the GPP printer name you wish to modify. The printerPath parameter is the GPAE path to the container where the printer type resides (Shared Printer, TCP-IP Printer or Local Printer). The oldName parameter contains the existing friendly name of the printer and the newName parameter is the name you wish to change the printer to. Once you issue this command, the printer name will be changed within the GPO.

### GPP Internet Settings

With the release of GPAE 3.0, we added support for newer browser versions within GPP's Internet Settings area. Specifically, we now have support for IE 8 & 9 and IE 10 (which also includes implicit support for IE 11). To access each of these areas, you simply use the following syntax:

1. *$gpo = get-sdmgpobject "gpo://cpandl.com/GPP-IE89" –openbyname*
2. *$container = $GPO.GetObject("User Configuration/Preferences/Control Panel Settings/Internet Settings/Internet Explorer 8 and 9")*
3. *$setting = $container.Settings.ItembyName("Internet Explorer 8")*
4. *$setting.Get("Home")*
5. *# sets Use TLS 1.0 from Enabled to Disabled on the Advanced tab, under Security*
6. *$setting.Put("Use TLS 1.0", $false)*
7. *$setting.Save()*

In this script, we access the IE 8 & 9 settings by first, in Line 1, getting a reference to the GPO containing the settings. In Line 2, we get a reference to the container that holds GPP IE settings, specifically for IE 8 & 9. In Line 3, we call ItemByName on the settings property within the container, to get existing IE 8/9 settings. Note that this property name could vary, depending upon what OS the script is run on. On Windows XP or Windows 7, it will likely say "Internet Explorer 8". On Win8 and greater, it will say "Internet Explorer 8 and 9". In addition, if there are currently no IE 8/9 settings defined on this GPO and you wish to add them, you would use *$setting = $container.Settings.AddNew("Internet Explorer 8")* instead.
In Line 4, we retrieve the value of the setting called "Home", which equates to the Home Page setting in this policy area. In Line 6, we set the Use TLS 1.0 Advanced security page setting to Disabled ($false) and then save that change to the GPO in Line 7. If you want to see all of the available properties under the IE 8/9 Preference area, as they are exposed by GPAE, simply type:

*$setting.GetPropertyNames()*

For IE 10/11, the syntax is very similar. You would reference it as follows:

```
$gpo = get-sdmgpobject "gpo://cpandl.com/GPP-IE10" -openbyname

$container = $GPO.GetObject("User Configuration/Preferences/Control
Panel Settings/Internet Settings/Internet Explorer 10")

$setting = $container.Settings.ItembyName("Internet Explorer 10")
```

Some of the properties on the GPP IE settings are not necessarily intuitive when it comes to setting their values. For example, simple setting properties that are in either an “Enabled” or “Disabled” state will take a value of 1 or 0, respectively, to change them. But some values are more complicated. Take this example for setting the Popup-Blocker filter level:

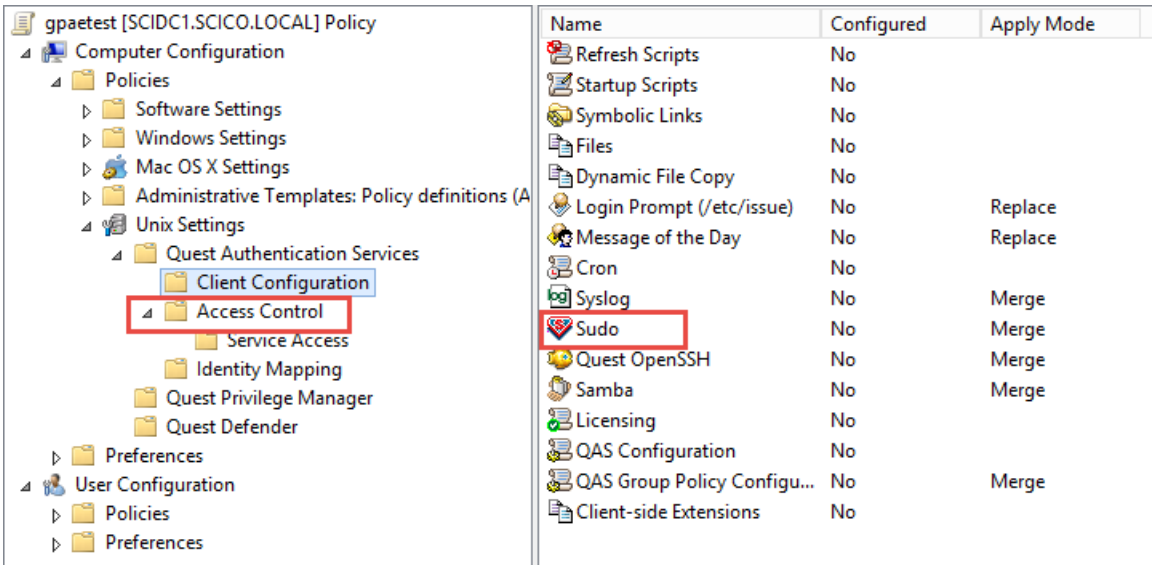
```
$gpo = get-sdmgpobject "gpo://cpandl.com/GPP-IE10" -openbyname
$container = $GPO.GetObject("User Configuration/Preferences/Control Panel
Settings/Internet Settings/Internet Explorer 10")
$setting = $container.Settings.ItembyName("Internet Explorer
10")$setting.Put("Filter level",[GPOSDK.FilterLevels]"Low")
```

In this example, the “Filter Level” property on the IE 10 GPP setting takes a GPAE enumeration, called [GPOSDK.FilterLevel] as its value. This enumeration has the properties of “Low”, “Medium” or “High”. If you have questions on any property’s value and it’s not clear what it should be, you can always contact us at support@sdmsoftware.com and we can help!

## Dell Software’s Quest Authentication Services

Quest Authentication Services from Dell Software began its journey years ago with a company called Vintella. The main purpose of the solution is to provide Group Policy based configuration for Unix clients. SDM Software has introduced support for two specific areas. These areas are:

1) Access Control – allowing users access to logon to target systems
    a. Users.allow
    b. Users.deny
2) Sudo settings – allowing named users to run processes on target systems




### Access Control

Reading and writing to these settings with Group Policy Automation Engine follow the same model that you have already discovered with the other native components of Group Policy. Here is an example of getting and setting value for Access Control setting user.allow.

```
    # Get a reference to a GPO
1.  $gpo = get-sdmgpobject -gpoName "gpo://scico.local/qastest" -openbyName
    # Get a reference to a container
2.  $ACcontainer = $gpo.GetObject("Computer Configuration/Unix Settings/Quest
    Authentication Services/Access Control")
    # Get a reference to a setting (Users.Allow & Users.Deny)
3.  $UAsetting = $gpo.GetObject("Computer Configuration/Unix Settings/Quest
    Authentication Services/Access Control/users.allow")
4.  $UDsetting = $gpo.GetObject("Computer Configuration/Unix Settings/Quest
    Authentication Services/Access Control/users.deny")
    # look at all properties and values
5.  $UAProperties = $UASetting.GetPropertyNames()
6.  foreach ($property in $UAProperties)
7.  {
8.  $propvalue = $UASetting.Get("$Property")
9.  Write-Output "$property = $propvalue"
10. }
    # Set QAS users.allow (users.deny is almost identical)
    # ApplyMode
11. $UASetting.Put("ApplyMode",[GPOSDK.QASApplyModeType]"Replace")
12. $UASetting.Save()
13. $UASetting.Put("ApplyMode",[GPOSDK.QASApplyModeType]"Merge")
14. $UASetting.Save()
    # Manifest Elements
15. $user = new-object "GPOSDK.QASManifestElement"
16. $user.ElementType = [GPOSDK.QASManifestType]"User"
17. $user.Entry="parker@scico.local"
18. $UASetting.Put("ManifestElements",$user)
19. $UASetting.Save()
```

Line 1 gets a reference to a GPO as we have seen throughout the examples in this guide.
Line 2 is a bit unnecessary here, but for completeness this line gets a reference to the container that holds the target settings.
Line 3 is connecting to the *users.allow* setting.
Line 4 is connecting to the *users.deny* setting.
Lines 5 – 10 are a simple foreach loop to go through each property available on the setting and enumerate their properties.
Lines 11 – 14 are examples of changing the 'ApplyMode' property to replace or merge.
Lines 15 – 19 update the setting with the information related to the user who has allow access rights to the target system.

As you can see the above follows the same model as all other GPAE interfaces.

### Sudo Settings

Here is a quick example of setting some Sudo settings.

```
    #Get a reference to the GPO
1.  $gpo = get-sdmgpobject -gpoName "gpo://scico.local/qastest" -openbyName
    # Get a reference to a container and a setting (Sudo)
2.  $Sudocontainer = $gpo.GetObject("Computer Configuration/Unix Settings/Quest
    Authentication Services/Client Configuration/Sudo/")
3.  $Sudosetting = $gpo.GetObject("Computer Configuration/Unix Settings/Quest
    Authentication Services/Client Configuration/Sudo/Configuration");
    # loop through existing properties
4.  $sudoProperties = $sudoSetting.GetPropertyNames()
5.  foreach ($property in $sudoProperties)
6.  {
```

```
7.  $propvalue = $sudoSetting.Get("$Property")
8.  Write-Output "$property = $propvalue"
9.  }
    # New rule
10. $rules = $sudoSetting.Get("Rules")
11. $rule = new-object "GPOSDK.QASSudoerEntry"
12. $rule.UnixCommand = "ALL"
13. $rule.RunAsUser = "root"
14. $rule.PasswordRequired = 1
15. $rule.ApplyToAll = 0
    # Initial user
16. $user = new-object "GPOSDK.QASSudoPrincipal"
17. $user.PrincipalType = [GPOSDK.QASPrincipalType]"user"
18. $user.Principal="Parker"
19. $rule.ApplyToList.Add($user)
20. $rules.Add($rule)
    # Additional Properites/Values
21. $sudoSetting.Put("FileLocation","/opt/quest/sbin/visudo")
22. $sudoSetting.Put("LoadPlugin",0)
23. $sudoSetting.Put("Rules",$rules)
24. $sudoSetting.Save()
```

In the above example much of the content is obvious as you have likely seen it before.
Line 1 – 9 is the same as the above example. Grab a GPO, container and setting and enumerate the available properties.
Lines 10 – 15 are important, this is where the Sudo rule is actually built. It is being built in memory and needs to be committed to the GPO with the .save() method.
Lines 15 – 20 defines the initial user to add to the Sudo rule -and-
Lines 21 – 24 puts it all together and commits the settings to the GPO.

## Summary

This document details the use of SDM Software GPAE to support Group Policy automation using PowerShell. We did not cover the use of .Net within this document but .Net languages are supported by GPAE. Please contact info@sdmsoftware.com for more information about this or view the accompanying GPAE object reference help file. In general, if you are familiar with C#, then the examples provided herein for PowerShell will translate well into C# and other .Net languages. In addition, the object reference that accompanies the GPAE installation provides some C# specific examples that you can use to get started.

For any questions about this product, please contact sales@sdmsoftware.com.