PROXY Administration with PowerShell
for PROXY Pro v10 and later

This guide provides information about using the Microsoft Windows PowerShell scripting platform to automate administrative tasks to be performed on the PROXY Pro Gateway Server and PROYX Pro Host products.

This document last updated on Thursday, December 13, 2018 for Proxy v10.1. Information is applicable to PROXY v10.0 and later releases.

Table of Contents

Introduction

Administrators often request ways to automate many of the routine tasks they perform in managing a Proxy Gateway Server -- e.g. moving Hosts between groups, adding or removing security permissions for users, etc.

These tasks are usually performed by hand using the Web Console or Proxy Gateway Administrator MMC snap-in (GWA). The GWA, in turn, performs its actions by making calls to the Proxy SDK runtime controls. These controls are installed by default on any system on which any other Proxy software component is installed, e.g. Gateway, Host, or Master.

Prior to Proxy v8.0, these controls were not accessible via scripting due to license restrictions. In Proxy v8.0 and later, both the Proxy Gateway control (PrxGW) and the Host Administration control (PrxHA) are now accessible via scripting. As a result, it is now possible to automate any formerly manual administrative process by assembling the right sequence of commands into a PowerShell script.

And by adding event handlers to your scripts, it is now possible to augment the built-in behavior of the Gateway Server with a customized, fully-autonomous Gateway management process.

The Proxy PowerShell Scripting package contains a set of PowerShell modules comprising a set of powerful yet easy-to-use routines you can call from your own PowerShell scripts, or directly from the PowerShell console. These routines simplify the process of making Proxy SDK calls to perform low level actions like moving Hosts between groups or modifying Gateway security settings.

Once you have loaded the Proxy PowerShell modules into a PowerShell command console, you're ready to start calling the high level routines from your PowerShell scripts, or directly from the command line. You also will have access to the complete low-level object model presented by the Proxy SDK.

This document gives a quick introduction to the prerequisites, the process of loading the Proxy PowerShell modules, and some common usage patterns. It also provides an overview of the modules provided in this package (that can be used as-is) and the sample scripts also provided (which you modify, and use as the basis for building more complex scripts).

Quick Start

If you’re impatient to get started, or are comfortable with PowerShell and want to dive right in, follow these steps *exactly* to get going. The explanation & rationale for these steps is explained in the sections below – if you want to know why these are the right steps, or run into trouble and need to diagnose what went wrong, please read the later sections *very carefully* to get the details.

1. Copy the contents of distribution ZIP file, “Modules” directory, to a directory named “%USERPROFILE%\Documents\WindowsPowerShell\Modules”. Create that directory if it doesn’t already exist.

2. Copy the contents of the distribution ZIP file, "Scripts" directory, to your desktop, keeping them in a folder named "Scripts".
3. Create a shortcut on your desktop to launch PowerShell (x86) with the correct configuration. On Windows x64 editions, this command line is

```
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -executionpolicy remotesigned -mta
```

   On Windows x86 editions, this command line is:

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -executionpolicy remotesigned -mta
```

   Set the "Startup Directory" of the shortcut to be "%USERPROFILE%\Desktop\Scripts".
4. Launch the new shortcut to open the PowerShell environment. If it current directory is *not* your desktop "Scripts" directory, change to that directory.
5. Enter the command ".\ReportGatewayInfo.ps1 -gateway <dns-name-of-gateway>", filling in the DNS name of your Gateway Server.
   a. This script assumes the Gateway Server accepts connections on TCP protocol port 2303; to specify a different protocol and port, specify the "-protocol" option with the protocol and port number separated by a vertical bar, e.g. "-protocol "SSL|8443"".
   b. This script uses the logged-in user identity by default; to specify alternate credentials for authentication, specify the "-credentials" option with the username and password separated by a vertical bar, e.g. "-credentials "domain\administrator|P@ssword"".

Common Error Scenarios

Error Message: STA Apartment mode

```
WARNING: You are running in STA Apartment mode.
This occurs in the PowerShell ISE, or by default in PowerShell v3 and later.
The Gateway Client's COM object will not function properly when running in
the STA. Specifically, if the network connection drops or fails, that will
not be reflected in the Gateway object state.
```

To resolve this issue, run the PowerShell environment in the COM MTA model. This can be specified on the PowerShell command line with the "-mta" switch.

Error Message: 80040154 Class not registered

```
New-Object : Retrieving the COM class factory for component with CLSID
{F632313B-2910-11E2-A0AC-005056C00008} failed
due to the following error: 80040154 Class not registered (Exception from
HRESULT: 0x80040154 (REGDB_E_CLASSNOTREG)).
```

This error indicates either that the PROXY Pro SDK Runtime controls are not installed on this machine, *or* that you're running the 64-bit version of PowerShell. The PROXY SDK controls are 32-bit, in-process COM objects, and must be used from the x86 version of PowerShell. Be certain to run PowerShell from the "C:\Windows\SysWOW64\WindowsPowerShell\v1.0\" directory.

Error Message: Import-Module cannot load

```
import-module : File
C:\Users\Administrator\Documents\WindowsPowerShell\Modules\proxy-
hostadmin\proxy-hostadmin.psm1 cannot
be loaded because running scripts is disabled on this system. For more
information, see about_Execution_Policies at
http://go.microsoft.com/fwlink/?LinkID=135170.
```

This error indicates that the execution policy prevents the script from running. Follow the provided link for more information. Use the "set-executionpolicy" cmdlet as discussed in the "Prerequisites" section of this document to enable script execution.

Prerequisites

Using any of these scripts requires:

- Microsoft Windows PowerShell v3.0, v4.0, v5.0, or v5.1 x86 edition. Note well that on a Windows x64 operating system, you must use the x86 version of the PowerShell scripting environment, because the Proxy components are COM components that are 32-bit only.
- Proxy SDK Runtime v10.0 or later installed. Note that these runtime components are installed along with the Host, Master, Gateway Server, and Deployment Tool product components, so if any of these items is installed on the computer, the Proxy SDK Runtime is there too.

PowerShell v3.0 is available from Microsoft in the Windows Management Framework 3.0. This is available for Windows 7 and Server 2008 R2 at http://www.microsoft.com/en-us/download/details.aspx?id=34595.

PowerShell v4.0 is available from Microsoft in the Windows Management Framework 4.0. This is included in Windows 8.1 and Server 2012 R2, and is available for Windows 7, Server 2008 R2, and Server 2012 at http://www.microsoft.com/en-us/download/details.aspx?id=40855.

PowerShell v5.0 (v5.1) is available from Microsoft in the Windows Management Framework 5.0 (5.1). This is included in Windows 10 and Server 2016, and is available for Windows 7, Server 2008 R2, Windows 8.1, Server 2012, and Server 2012 R2 at https://www.microsoft.com/en-us/download/details.aspx?id=50395 (or https://www.microsoft.com/en-us/download/details.aspx?id=54616 for v5.1).

To confirm what version of PowerShell you have:

1. Open the "Windows PowerShell (x86)" command prompt.
2. Issue the command "`$Host.Version`".
3. The output is the version of the PowerShell environment. The important number is the "Major" version number, which corresponds to the PowerShell major version (2 through 5).

To make the Proxy administration modules available to PowerShell, they either need to be copied to a directory on the PowerShell path, or placed in a location that is added to the path. To find out what the current PowerShell path is, issue the command "`$ENV:PSModulePath`".

This typically includes a per-user directory like "c:\users*username*\Documents\WindowsPowerShell\Modules"; placing the modules in this location is recommended. Alternately, you can add any directory to this path with a command like "$ENV:PSModulePath = "c:\proxy\powershell;" + $ENV:PSModulePath".

Finally, the default execution policy is to not allow scripts to run in general. Please see "`get-help about_execution_policies`" and "`get-help about_signing`" for more details about execution policy and code signing.

For initial testing and development, we recommend changing the execution policy to RemoteSigned. This allows local scripts to run, but provides protection against (potentially) malicious scripts from the internet. The command to set this for just the current PowerShell command prompt window is "`Set-ExecutionPolicy –ExecutionPolicy RemoteSigned –Scope Process -Force`".

Finally, be certain to read the next section, "PowerShell Threading Models", for important information about starting PowerShell v3 or later in the MTA threading model.

PowerShell Threading Models

Because the Proxy SDK components are COM objects, their operation is dependent on the COM threading model of the process that they're used in. Proxy supports both the single threaded apartment model (STA) as well as the multi-threaded apartment model (MTA). However, PowerShell implements the STA in a non-standard way, and use of the MTA is **strongly recommended**.

In PowerShell v2.0, the PowerShell command prompt defaults to the MTA model, and the SDK controls work reliably here. The Integrated Scripting Environment (ISE) uses the STA model, but does not consistently/promptly process Windows messages. This leads to Proxy components failing to recognize and process events like disconnect, and therefore this mode is less reliable than the MTA model.

In PowerShell v3.0 and later, both the PowerShell command prompt and the ISE default to the STA model for consistency. Because PowerShell doesn't consistently/promptly process Windows message in this mode, like PowerShell v2.0, this is less reliable than the MTA model.

PowerShell v3.0 or later command prompt should be started in MTA mode using the command line switch "-mta". You can set this by creating a shortcut with a command line like:

```
path\powershell.exe -executionpolicy remotesigned -mta
```

where '*path*' is the correct path to the PowerShell executable, e.g. "%SystemRoot%\system32\WindowsPowerShell\v1.0" on Windows x86 or "%SystemRoot%\syswow64\WindowsPowerShell\v1.0" on Windows x64.

References

PowerShell is a Microsoft technology in the Windows platform, and Proxy Networks cannot answer questions about generic PowerShell scripting. For information about PowerShell, please see the following Microsoft references:

Windows PowerShell

https://docs.microsoft.com/en-us/powershell/
https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view=powershell-5.1
https://docs.microsoft.com/en-us/powershell/developer/windows-powershell

PowerShell's Security Guiding Principles (Blog post by the PowerShell Team)

http://blogs.msdn.com/b/powershell/archive/2008/09/30/powershell-s-security-guiding-principles.aspx

Getting Started

To see the Proxy PowerShell Scripting at work, execute the following sequence of commands after following the instructions in "Prerequesites".

```
PS> Import-Module Proxy-Gateway
PS> Connect-Gateway 'gatewayspecifier'
PS> $gws = Get-GatewayObject
PS> $gws.getHosts() | where { $_.machine -ne 'virtual' }
PS> Disconnect-Gateway
PS> $gws = $null
```

These steps do the following, in order:

1. Import the "Proxy-Gateway\Proxy-Gateway.psm1" module contents into this PowerShell command prompt.
2. Connect to the specified Gateway Server. The arguments for "`Connect-Gateway`" are:
    a. "`-station` *`stationname`*" to specify the Gateway to connect to. This can be a machine name, DNS name, or IP address. This is the default argument.
    b. "`-protocol` *`protocolname`*" to specify the protocol, either "TCP", "UDP", "SSL", "WSS" or "WS".
    c. "`-credentials` *`credentialsstring`*" to specify explicit credentials (instead of using the logged-in username). This string must be quoted, and is in the form "*domain\username|password*". If this argument is omitted, connection will use current User's credentials from your Windows logon session.
3. Retrieve the connected Gateway object into the local variable `$gws`.
4. Call the `getHosts()` function on the Gateway object, and filter the results to eliminate items where the "`machine`" property is the value "virtual".
5. Disconnect the connected Gateway object. This invalidates, but does not clear, the local variable `$gws`.
6. Clear the local variable `$gws`.

Note very well that the design of the Proxy PowerShell Scripting module is such that there can be only one connection to a Proxy Gateway Server per instance of PowerShell. In order to connect to a different Gateway Server, the current connection must be disconnected first (using the `Disconnect-Gateway` cmdlet).

This is also true of the Host Administration connection, available in the Proxy-HostAdmin module.

Proxy PowerShell Script Modules

The Proxy PowerShell Scripting package includes a number of modules to provide access to Proxy features. These modules should be placed in one of the directories from the $ENV:PSModulePath path, or a new directory should be added to that path, in order to locate the modules.

The modules included in this release of the package are as follows. Each module is in a directory with the module name, and the file is the module name with extension ".PSM1".

Module	**Description**
Proxy-Gateway	Main functionality for accessing Gateway Server. Creates and manages Gateway connection, subscribes to Gateway events, and includes example event handlers.
Proxy-HostAdmin	Administration of Host Settings via Gateway. Includes commands to connect to the Host Administration object via a Gateway server.
Proxy-Constants	Defines security and other constants. Should be imported to have definitions of the specific security rights in Proxy-GatewaySec and other security modules. Has IDs of the top-level groups.
Proxy-GatewaySec	Security-related cmdlets. Importing Gateway Accounts and assigning their category and related groups and hosts.
Proxy-Utils	Defines the "Wait-EventWithMessageLoop" function, which is required to wait for an event when running in the COM STA. This module is automatically included by other modules that need it.
Proxy-Gateway-Utils	Helper functions. This module is automatically included by other modules that need it.
Proxy-Gateway-ADs	Helper functions for accessing Active Directory Services. This module is automatically included by other modules that need it.
Proxy-ConnectionEvents	Defines constants used within the PowerShell scripts. This module is automatically included by other modules that need it.

Proxy Sample PowerShell Scripts

In addition to the modules that provide subroutine library functionality, and which can be used as-is, this package includes several sample scripts designed to be modified to meet your needs.

The scripts included in this release of the package are as follows. Each script is in the "Scripts" directory, with extension ".PS1".

Script	Description
Add-GatewayToHostConfig Remove-GatewayFromHostConfig	Scripts to add/remove a Gateway configuration to/from a Host that's already connected-to for administration. Used in ConfigureAllHostsOnGateway script.
Add-NewPersonalUser	Script for importing a new user to the Gateway security model. This script should be changed to accommodate specific organizational policies and processes.
ConfigureAllHostsOnGateway	Script to enumerate all Hosts managed by a Gateway, connect to each one for Host administration, and the update the Gateway configurations at that Host. Uses Add-GatewayToHostConfig and Remove-GatewayFromHostConfig scripts.
Delete-MatchingGatewayGroups	Script to find Gateway groups that fit a pattern, and delete them. This script was developed by a customer that had originally used a Host Grouping Rule with the default "OU=" prefix, then decided to change that, so new groups with the new naming convention were created. The script was used to delete the no-longer-needed "OU=*" groups.
Get-HostSettings	Script to read many of the Host settings and return name/value pairs of information as a hash table or PSObject.
Get-HostSettingsToXML	Script to read many of the Host settings and return an XML document containing them.
Invoke-HostTask	Script to invoke a PowerShell script block against a connected PrxHA control instance for the specified Host.
ReportGatewayInfo	Script to count different objects in the Gateway Server and report the counters. This is a very simple script and is a good way to test that the PowerShell environment is working.
taskWatchGatewayEvents	Script demonstrates how to run in the background and monitor for specific Gateway event notifications.

Test, Get and Disconnect existing Gateway connection

Import the module from script, not knowing if it's already loaded

```
if (-not (Get-Module Proxy-Gateway))
{
	Import-Module Proxy-Gateway
}
```

Test if Proxy-Gateway module is connected to a Gateway Server

```
Test-Connection
```

This cmdlet returns $true if a connection exists and is connected, or $false otherwise.

Retrieve the COM object that is the root of the Proxy object model

```
$gws = Get-GatewayObject
```

This cmdlet returns the COM object “ProxySDKv10.ProxyGWClass”, which is the root object of the Proxy PrxGW Gateway Administration object model.

Disconnect the current connection to the Gateway Server

```
Disconnect-Gateway
```

Proxy Gateway events: examples and templates

Proxy Gateway Server is using .NET wrapped COM events to notify about changes in different collections, like Group Membership, Unmanaged Hosts, Workstations etc. (ProxySDK documentation has a list of all Gateway events in 'ProxyGW Events' section.)

To subscribe for events in the PowerShell script, you need to enable notifications for the specific event group (because they’re off by default), and also to register an action for the event. For example, the following script lines associate the script block $action with changes to the Unmanaged Hosts collection:

```
# Enable Group membership notifications on the Gateway Server
$gws.enableChangeNotifications([ProxySDKv10.GatewayCollection]::coll
      ectUnmanagedHosts, -1, "", -1)
# Enable group membership event's handling in script
$newEvent = Register-ObjectEvent $gws onUnmanagedHostDataChanged -
      Action $action
```

There are several helpers to register actions for the most common events along with the Action's examples and templates. Event registration helpers:

- Register-GroupMembershipChange - Group membership change notifications. (Fired when Host is moving from group to group except "Unmanaged" and "All Hosts" which are not real groups). Template Action: $template_event_group_membership_action
- Register-UnmanagedHostChange - Unmanaged Host collection change notifications. (Fired when new host is added to "Unmanaged" Hosts). Template Action: $template_event_unmanaged_action. Example Action: $example_event_unmanaged_action.
- Register-WorkstationsDataChange - Property of a workstation-type Host has changed notification. (Can be used to find when new Host is added to "Managed" Hosts). Template Action: $template_event_workstation_data_action. Example Action: $copy_new_workstation_to_OUs_action

Examples:

```
Register-UnmanagedHostChange $example_event_unmanaged_action
Register-WorkstationsDataChange $copy_new_workstation_to_OUs_action
```

Security management

The Proxy object model provides full access to the security configuration in the Gateway (restricted by access rights – only authorized users can read or change the security settings). Scripting security policies and changes can avoid errors that may be made when security is edited "by hand".

Side Note on COM Collections and Objects in PowerShell

Many PROXY SDK functions return either a single COM object, or collection of COM objects as an enumeration. In the following explanations, the COM interface names are used for describing object types. These names are not commonly used in PowerShell, but the type of (COM) object determines what properties and methods are available. COM interface names being with "I" by convention; in PowerShell, these can simply be assigned to a PowerShell variable and used to call its methods or pass it to other methods which take that type of object as a parameter.

The PROXY SDK "collection" methods always return a COM collection object (enumeration). However, if that object is returned from a PowerShell function and then assigned to a variable, the enumeration is turned into a PowerShell object, which may change how the results appear. Specifically, if the collection is empty, the result is $null; if there was one element in the collection, the result is that single element (not in array form); if the collection had two or more elements, the result is an array of element. Note well however that the COM collection follows Visual Basic 6 conventions and the .Item method indexes from 1 to Count(); the PowerShell array follows C# conventions and is indexed from 0 to Length-1. Note that the PowerShell syntax @(…) can be used to wrap the results in an array, ensuring that single objects are treated consistently as arrays.

Accounts

In v10 and later, it is necessary to have an imported Gateway Account, in order to have access to the PROXY system. The imported account may be either an individual user account, or a group account. Before v10, PROXY only supported Windows accounts. As of v10 and later, we also support AzureAD accounts, with other identity providers planned for the future.

When an account is imported, it is categorized as either a Personal, Master or Administrative account, with increasing rights and privileges automatically assigned to each category. Briefly, a Personal account category has access to only a single Host, a Master account category has access to all Hosts in one or more Host groups, and an Administrative account category has full access to all Hosts and groups.

If the imported account is an individual user account, that account is marked as the given category, and that specific account is then automatically granted specific rights on the Gateway Server, and on whichever Hosts and Host groups have been assigned. If the imported account is a

group account (e.g. "Remote Desktop Users"), then that group account is marked as the given category, and that group account is automatically granted the specific rights on the Gateway, Hosts and Host groups. In that case, any member of that group account may log in to the PROXY system and operate as the assigned category of user.

In summary, when you import an account, you should decide whether to import individual accounts or group accounts, and you should decide whether to import the account as a Personal, Master or Administrative category.

Account Management Methods

Accounts are operated through the "$gws.settings" property of the Gateway object which has following methods:

- getAccounts() # Returns collection of IProxyGWAccount objects. These are existing Gateway Accounts.
- importAccount(IProxyUserAccount* Account, [int] UserCategory, [string] GroupIDs, [string] WorkstationID); # Imports an account and assigns its category, groups and hosts. Returns IProxyGWAccount object.
- importAccountByName([string] AccountName, [string] AccountID, [string] AccountAuthority, [int] UserCategory, [string] GroupIDs, [string] WorkstationID, [int] AccountType, [string] EmailAddress, [string] DisplayName); # Imports an account and assigns its category, groups and hosts. Returns IProxyGWAccount object.

With these methods you can enumerate existing accounts, represented by IProxyGWAccount, or to import new accounts. If you import account of existing User/Group it will simply apply new security settings to it.
Two interfaces are important for account management:

- IProxyGWAccount represents a successfully imported Gateway Account.
- IProxyUserAccount is a separate interface that simply describes a user or group account that exists in a given repository, e.g. Active Directory.

To import an individual account as a Personal category user, you need at minimum either an IProxyUserAccount or you need to know the account name, the account ID (in Windows Authentication this is a SID), and the workstation ID of the PROXY Host to be assigned to that account; for Personal category users, the GroupIDs is left blank. To import an individual or group as a Master category user, you again need to know the account name and ID, and you need the ID of one or more PROXY Host groups. The Host group IDs are then constructed as a comma-separated list, e.g. "<group1>,<group2>…<etc.>"; for Master category users, the WorkstationID is left blank. To import an Administrative category account, leave both WorkstationID and GroupIDs blank. You can also import an account that has already been imported, and thereby change its category or reset its security permissions to a known state.
To obtain a IProxyUserAccount, you can either create one from scratch and fill in the properties "by hand", or you can use the one of the following APIs exposed by IProxySecurity_v9, obtained from any security object exposed by the Gateway, e.g. "$gws.settings.gatewaySecurity".

NOTE: There is a known issue with these functions in PROXY v10.1 FCS version. Please use the latest v10.0 hotfix, or v10.1 hotfix#1 or later to avoid this issue.

IProxySecurity_v9:

IProxySecurity_v9 was introduced in PROXY v9 to allow the Web Console to remotely edit security on installed PROXY Hosts (version 9 and 10). In PROXY v10, the same interface was used in our new Gateway security model, which supports not just the Windows security model, but also AzureAD (and other directories in future releases).

Along with the other methods, it allows the lookup User Accounts in authorities accessible from Proxy Identity Manager (PIM). These accounts are used to import new Users and/or Groups:

- GetAuthorities(); # Returns a list of authorities as an enumeration of strings.
- SearchAuthorityForUsers([string] Authority, [string] Prefix, [int] Limit, [ref] MoreData); # Returns collection of IProxyUserAccount objects.
- SearchAuthorityForGroups([string] Authority, [string] Prefix, [int] Limit, [ref] MoreData); # Returns collection of IProxyUserAccount objects.

Please note:

1. "Authority" is the name of the identity provider being searched, i.e. either the local Gateway machine name or the Active Directory domain, or the AzureAD name. You will need to use **the exact identity provider's name**, as it known to the PIM. In order to get the exact name you can use method "GetAuthorities()", which will return a list of known names.
2. "Prefix" is a possibly empty search string used to filter the set of results.
3. "Limit" simply limits the number of results.
4. "MoreData" indicates whether there were more results possible than were returned due to the limit.

As mentioned above, when an account is imported, it is automatically granted some specific rights on the Gateway and on one or more Hosts or Host groups. If those default rights are not suitable for your application, you may directly manipulate the security settings on the Gateway or the Host or Host Groups via the IProxySecurity_v9 interface.

In general, changes you make via IProxySecurity_v9 supersede the default access provided when the account was imported, but the account category always places a limit on the rights which may be assigned. For instance, a Master category account may never be granted full rights to the Gateway settings. In this way, the account category acts as a "security blanket".

Usage

To use the security management modules, you need to "Import-Module" both "Proxy-Constants" and "Proxy-GatewaySec". Module "Proxy-GatewaySec" provides a simple wrapper around functions you will need to import and remove Gateway Accounts.

- *Get-GatewayAuthorities* # Returns a list of "authorities" which can be used for importing User Accounts to the Gateway Server. Authorities from this list can be used to call Find-UserAccounts.
- *Find-UsersInAuthority* # Finds a list of User Accounts in given authority matching some prefix.

- *Find-GroupsInAuthority* # Finds a list of Groups in given authority matching some prefix.
- *Add-GatewayAccount* # Imports User Account to the Gateway assigning it a "category" and linking it to the security of Groups and Hosts.
- *Remove-GatewayAccountByName* # Removes Gateway Account.

In order to find more about parameters of these function use Get-Help with the function name. Also, Add-NewPersonalUser.ps1 script demonstrates usage of “Proxy-GatewaySec” module for importing users from the Identity Provider.

Some common tasks

List all Gateway-based groups

```
$gws.getAllGroups()
```

Add a new sub-group to the top-level group “Managed Hosts”

```
(Get-TopLevelGroup -gateway $gws -groupID $groupIDManagedHosts).addGroup('QA', 'QA Hosts', $groupTypeNormal)
```

List “Active Gateway Data Services” collection (from Gateway Administrator “Active Status” folder)

```
$gws.getActiveClients()
```

List “Active Master Connection Services” collection (from Gateway Administrator “Active Status” folder)

```
$gws.getActiveMasters()
```

List “Active Hosts” collection (from Gateway Administrator “Active Status” folder)

```
$gws.getActiveHosts()
```

List “Active Recordings” collection (from Gateway Administrator “Active Status” folder)

```
$gws.getActiveRecordings()
```

List “Reverse Connections” and “Pending Host Status Updates” collections (from Gateway Administrator “Active Status” folder); note that the Gateway Administrator segregates these items based on the “IsReverseConnection” property)

```
$gws.getActiveHostCtrlConnections()
```

Explore the methods of the “Active Host Control Connection” object

```
@($gws.getActiveHostCtrlConnections())[0] | Get-Member
```

Disconnect a Host (that is connected through the Gateway for remote control & other services)

```
@($gws.getActiveHosts())[0].disconnectHost()
```

Find group named “QA”

```
Find-GroupByName -gateway $gws –name 'QA'
```

List the Hosts that are members of the Group “QA”

```
(Find-GroupByName -gateway $gws –name 'QA').getHosts()
```

Add Host named “WKS-123” to group “QA”

```
(Find-GroupByName -gateway $gws –name
     'QA')).addHost(($gws.getHosts() | ? {$_.name -eq 'wks-123'}))
```

Get list of all Unmanaged workstations

```
$gws.settings.getUnmanagedWorkstations()
```

Manage the unmanaged Host named “WKS-123” (this moves it from “Unmanaged” to “All Hosts”)

```
($gws.settings.getUnmanagedHosts() | ? {$_.name -eq 'wks-
     123'}).isManaged = -1
```

Add the Host named “WKS-123” to the Group named “QA”

```
(Find-GroupByName -gateway $gws –name 'QA'}).addHost(($gws.settings.
     getUnmanagedHosts() | ? {$_.name -eq 'wks-123'}))
```

Running “Service” scripts

Running and managing scripts as Scheduled Jobs

With the release of PowerShell 3.0 it becomes very convenient to schedule periodic maintenance jobs from the PowerShell environment. Jobs can be used to enforce policy rules, perform maintenance tasks and give real time notifications about events from Proxy Gateway Server. PowerShell Scheduled Jobs are managed by Windows Task Scheduler and can use all options available in it. For some IT tasks you may prefer to use Task Scheduler UI after you create a new Scheduled Job from PowerShell.

“You can view and manage the jobs in Task Scheduler, enable and disable them as needed, run them or use them as templates, establish a one-time or recurring schedules for starting the jobs, or set conditions under which the jobs start.” [https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/about/about_scheduled_jobs?view=powershell-5.1]

Also, there is no place here to fully discuss such a big feature as PowerShell’s Job Scheduling, but we will give all necessary information on how to use it, specific to running Proxy PowerShell scripts, and also include some pointers to further reading at the end of this section.

NOTE: You should run PowerShell with elevated privileges in order to register and manage Scheduled Jobs.

Usual way to schedule a new job is to create its “trigger” and “options” objects first. “Trigger” is what tells the job when to run and it is a mandatory parameter. For example:

```
PS> $trigger = New-JobTrigger –Daily –At '4 am' #runs task daily at 4am
```

```
or
PS> $trigger = New-JobTrigger -Once -At (Get-Date) -RepetitionInterval (New-TimeSpan -Minutes 2) -RepetitionDuration ([TimeSpan]::MaxValue) #runs task every two minutes for unlimited period of the time starting from now
```

"Trigger" can be created in any way "*New-JobTrigger*" command allows you, and has no specific restrictions related to Proxy PowerShell scripting.
"Options" object is mandatory for running Proxy PowerShell scripts and have advanced options for a scheduled job. The mandatory part is "*-RunElevated*" switch. For example:

```
PS> $opt = New-ScheduledJobOption –RunElevated # minimum set of options you need to run Proxy PowerShell scripts.
```

When the "Options" and "Trigger" objects created, we can register a new scheduled job:

```
PS> Register-ScheduledJob -Name proxyMtnJob -FilePath "C:\Proxy\Scripts\someLongRunningScript.ps1" -Trigger $trigger -ScheduledJobOption $opt -RunAs32 -Credential "username" # creates new job with given name, trigger and options, running script from the file system
```

When using *Register-ScheduledJob* there is only one parameter which is mandatory for Proxy scripts: - "*-RunAs32*". If this parameter is omitted PowerShell will run script in 64 bit environment which is not compatible with Proxy SDK 32 bit. But if you have Proxy SDK 64 bit installed you should omit this parameter from the command.

After new Scheduled Job is successfully registered, PowerShell has commands to manage it and its instances. For example you can view scheduled job(s):

```
PS> Get-ScheduledJob
```

, or you can run it on-demand in the current shell's background:

```
PS> Start-Job –DefinitionName proxyMtnJob
```

"Jobs that are started by using the Start-Job cmdlet are standard Windows PowerShell background jobs, not instances of the scheduled job. Like all background jobs, these jobs start immediately – they are not subject to job options or affected by job triggers – and their output is not saved in the Output directory of the scheduled job directory."
[https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/about/about_scheduled_jobs?view=powershell-5.1]

Here how you can force the scheduler to run Scheduled Job by adding a new trigger to it which will run job once after 1min delay:

```
PS> $trigNow = New-JobTrigger -Once -At ((Get-Date) +
[timespan]"00:01:00")
PS> Add-JobTrigger -Name proxyMtnJob -Trigger $trigNow
```

NOTE: In order to use Get-Job, Receive-Job and other job related commands to work with Scheduled Job's instances from PowerShell, you either should use one of the Scheduled Job commands first or manually load "PSScheduledJob" module with *Import-Module cmdlet*:

```
PS> Import-Module PSScheduledJob
```

Once Scheduled Job has run, you can run *Get-Job* to fetch a list of available Job instances. Every individual job result will appear on the list and have the same name as the scheduled job that ran it (but different time and ID). Notice that there is a difference between Job and Scheduled Job. Scheduled Job is an entry in Windows Task Scheduler which can be viewed by calling to *Get-ScheduledJob* and it is effectively just a job description, not an actual running job instance. Once the "Trigger" criteria was met a real job instance will run creating new PowerShell background job which is not tied to any interactive PowerShell instance but can be managed from it with *Get-Job* and other related commands. For example you can retrieve all outputs from every run of Scheduled Job from any PowerShell instance with:

```
PS> Get-Job | ? { $_.name –eq "proxyMtnJob" } | % { Receive-Job -id
$_.id -Keep }
```

Output for scheduled jobs is kept on the disk of the computer where the job exists. Usually it's in your user profile directory in *\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs*, with a subfolder for each job name. The resulting output will remain on disk after being "received" with *Receive-Job*, even if you didn't use the "*-Keep*" parameter. You can use *Remove-Job* to delete job's results from the disk or "*-MaxResultCount*" parameter of *Register-ScheduledJob and Set-ScheduledJob* to set how many of saved results to retain on the disk (default is 32). Every new result will delete the oldest one when limit is reached.
So, we covered here simple set of three cmdlets which you need in order to run Proxy Script as a Scheduled Job: - *New-JobTrigger*, *New-ScheduledJobOption* and *Register-ScheduledJob*. Please, take a look at links below and at PowerShell help for related commands before going to the next section where we will discuss what type of Proxy Scripts you can run as Scheduled Jobs.

Links:

- PowerShell 5.1: https://docs.microsoft.com/en-us/powershell/scripting/PowerShell-Scripting?view=powershell-5.1
- Scheduled_Jobs : https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/about/about_scheduled_jobs?view=powershell-5.1

- Scheduled_Jobs_Troubleshooting: https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/about/about_scheduled_jobs_troubleshooting?view=powershell-5.1
- Windows PowerShell Scheduled Job Cmdlets: https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/?view=powershell-5.1
- PSScheduledJob Module ("about_*" sections has a lot of info): https://docs.microsoft.com/en-us/powershell/module/psscheduledjob/about/about_scheduled_jobs?view=powershell-5.1
- MSDN blog about Job Scheduling: http://blogs.msdn.com/b/powershell/archive/2012/03/19/scheduling-background-jobs-in-windows-powershell-3-0.aspx

Managing Scheduled Jobs

- *Add-JobTrigger* – add a new trigger to an existing scheduled job
- *Disable-JobTrigger* – turn off a scheduled job's triggers, but don't delete them
- *Enable-JobTrigger* – enable a previously disabled scheduled job trigger
- *Get-JobTrigger* – gets the triggers of scheduled jobs
- *New-JobTrigger* – create a new job trigger
- *Remove-JobTrigger* – remove a job trigger from the scheduled job
- *Set-JobTrigger* – reconfigure a trigger on a scheduled job
- *Get-ScheduledJobOption* – gets the options for a scheduled job
- *New-ScheduledJobOption* – creates a new option set
- *Set-ScheduledJobOption* – reconfigures a scheduled job's options
- *Disable-ScheduledJob* – disable, but do not delete, a scheduled job
- *Enable-ScheduledJob* – re-enable a previously disabled scheduled job
- *Get-ScheduledJob* – get the scheduled jobs on a computer
- *Register-ScheduledJob* – creates and registers a new scheduled job
- *Set-ScheduledJob* – reconfigure an existing scheduled job
- *Unregister-ScheduledJob* – removes a scheduled job entry

Managing Scheduled Job instances

- *Get-Job* – get background jobs
- *Receive-Job* - gets the results of background jobs
- *Remove*-Job - delete background jobs

Script types

Now, when we know how to run Proxy scripts with the help of PowerShell 3.0 scheduler, let's take a look at different kind of scripts we can run to help with Proxy Gateway Server management and maintenance. There are two types of scripts which we will cover here: short running scripts for periodic maintenance jobs and long running scripts for event notifications.

Short running scripts are just the same scripts as we already covered in other parts of this document and have no special properties. If you have some maintenance work to do, like moving Hosts around or changing their properties according to some policy, just follow the guidelines from previous section and apply it to the same script you will run from the interactive prompt. Simplest form of such script will be: import Proxy modules → connect to Gateway → do the job → disconnect.

Something like this:

```
Import-Module Proxy-Utils
Import-Module Proxy-Gateway
Import-Module Proxy-Gateway-Utils

Connect-Gateway 'localhost'
if(Test-Connection)
{
    $gws = Get-GatewayObject

    # Do the maintenance job here

    Disconnect-Gateway
}
```

Long running scripts are for the cases when you want to receive real-time notifications on something happening in Proxy Gateway Server (GWS). Such script will be scheduled to run at the system's startup and should have two distinct properties; It should subscribe for GWS event notifications and it should handle occasional disconnects.
Subscribing for GWS notifications is covered in "Proxy Gateway events: examples and templates" chapter and you can add your own events handling to your scripts following it's guidelines.
Handling of disconnects is something new and you only need it for long running scripts to survive GWS going down or network problems. "*Scripts\taskWatchGatewayEvents.ps1*" is a template for such a script. It uses an example action *$template_event_group_membership_action* from *Proxy-Gateway* module and cmdlet *Register-GroupMembershipChange* to register this action to be fired on the "Group Membership Change" event from GWS. When creating "trigger" and "options" for long running job, use following code as a guideline:

```
PS> $trigger = New-JobTrigger -AtStartup
PS> $opt = New-ScheduledJobOption -MultipleInstancePolicy StopExisting
-RunElevated
```

Or some other combinations which fit a profile of long-running script. The only mandatory parameter is "*-RunElevated*" in the "options" object.
It is a good idea to run your script from PowerShell interactive environment first, to make sure it's working, before assigning it to Scheduled Job.

NOTICE: Scheduled Script's output is not available until script is finished. This makes "Long running script" output invisible to `Receive-Job` command when it still running. Use some other mechanism to monitor long running script: - log files, email, Windows Event Log, etc. The "*Scripts\taskWatchGatewayEvents.ps1*" **example is using PSLog module**

which can be found at: http://gallery.technet.microsoft.com/scriptcenter/PSLog-Send-messages-to-a-db389927 **or installed with "**`Install-Module PSLog`**".**