

Quest® One ActiveRoles® Management Shell *for Active Directory*

Version 1.6

Administrator Guide

**© 2012 Quest Software, Inc.
ALL RIGHTS RESERVED.**

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST'S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters
LEGAL Dept
5 Polaris Way
Aliso Viejo, CA 92656
email: legal@quest.com

Refer to our Web site (www.quest.com) for regional and international office information.

Trademarks

Quest, Quest Software, the Quest Software logo, Simplicity at Work, and ActiveRoles are trademarks and registered trademarks of Quest Software, Inc. For a complete list of Quest Software's trademarks, see <http://www.quest.com/legal/trademarks.aspx>. Other trademarks are property of their respective owners.

Third Party Contributions

This product contains some third party components (listed below). Copies of their licenses may be found at <http://www.quest.com/legal/third-party-licenses.aspx>. Source code for components marked with an asterisk (*) is available at <http://rc.quest.com>.

COMPONENT	LICENSE OR ACKNOWLEDGEMENT
.NET logging library 1.0	BSD 4.4

CONTENTS

INTENDED AUDIENCE	7
CONVENTIONS	7
ABOUT QUEST SOFTWARE, INC.	8
CONTACTING QUEST SOFTWARE	8
CONTACTING QUEST SUPPORT	8
INTRODUCTION	9
USING ACTIVEROLES MANAGEMENT SHELL	10
INSTALLING AND OPENING ACTIVEROLES MANAGEMENT SHELL	10
GETTING HELP	12
CMDLET NAMING CONVENTIONS	13
PARAMETERS	13
SYNTAX	16
PIPELINING	17
ALIASES	18
WHAT'S NEW IN VERSION 1.6	20
WHAT'S NEW IN VERSION 1.4	21
TROUBLESHOOTING	25
CMDLET REFERENCE - ACTIVE DIRECTORY	27
CONNECT-QADSERVICE	28
DISCONNECT-QADSERVICE	31
GET-QADUSER	32
SET-QADUSER	52
NEW-QADUSER	60
DISABLE-QADUSER	65
ENABLE-QADUSER	67
UNLOCK-QADUSER	69
DEPROVISION-QADUSER	71
GET-QADGROUP	74
SET-QADGROUP	87
NEW-QADGROUP	93
GET-QADGROUPMEMBER	98
ADD-QADGROUPMEMBER	106
REMOVE-QADGROUPMEMBER	109
GET-QADMEMBEROF	111
ADD-QADMEMBEROF	118
REMOVE-QADMEMBEROF	121
GET-QADCOMPUTER	123
SET-QADCOMPUTER	136

NEW-QADCOMPUTER	141
DISABLE-QADCOMPUTER	145
ENABLE-QADCOMPUTER	147
RESET-QADCOMPUTER	149
GET-QADOBJECT	151
GET-QADMANAGEDOBJECT	162
SET-QADOBJECT	170
NEW-QADOBJECT	174
MOVE-QADOBJECT	177
RENAME-QADOBJECT	179
REMOVE-QADOBJECT	181
RESTORE-QADDELETEDOBJECT	184
NEW-QADPASSWORDSETTINGSOBJECT	187
GET-QADPASSWORDSETTINGSOBJECT	191
GET-QADPASSWORDSETTINGSOBJECTAPPLIESTo	203
ADD-QADPASSWORDSETTINGSOBJECTAPPLIESTo	210
REMOVE-QADPASSWORDSETTINGSOBJECTAPPLIESTo	212
GET-QADROOTDSE	214
GET-QADPERMISSION	216
ADD-QADPERMISSION	221
REMOVE-QADPERMISSION	226
GET-QADOBJECTSECURITY	228
SET-QADOBJECTSECURITY	230
ADD-QADPROXYADDRESS	233
SET-QADPROXYADDRESS	236
REMOVE-QADPROXYADDRESS	238
CLEAR-QADPROXYADDRESS	240
ENABLE-QADEMAILADDRESSPOLICY	241
DISABLE-QADEMAILADDRESSPOLICY	243
CMDLET REFERENCE - QUEST ONE ACTIVEROLES	245
PUBLISH-QARSGROUP	246
UNPUBLISH-QARSGROUP	249
GET-QARSACCESSTEMPLATE	251
GET-QARSACCESSTEMPLATELINK	259
SET-QARSACCESSTEMPLATELINK	269
NEW-QARSACCESSTEMPLATELINK	273
REMOVE-QARSACCESSTEMPLATELINK	277
GET-QARSONOPERATION	280
GET-QARSLASTOPERATION	289
GET-QARSWORKFLOWDEFINITION	291

GET-QARSWORKFLOWINSTANCE	299
GET-QARSAPPROVALTASK	304
APPROVE-QARSAPPROVALTASK	311
REJECT-QARSAPPROVALTASK	313
CMDLET REFERENCE - X.509 CERTIFICATE MANAGEMENT	315
GET-QADLOCALCERTIFICATESTORE	316
NEW-QADLOCALCERTIFICATESTORE	318
REMOVE-QADLOCALCERTIFICATESTORE	319
GET-QADCERTIFICATE	321
WHERE-QADCERTIFICATE	327
ADD-QADCERTIFICATE	334
IMPORT-QADCERTIFICATE	337
SHOW-QADCERTIFICATE	339
EDIT-QADCERTIFICATE	340
EXPORT-QADCERTIFICATE	341
REMOVE-QADCERTIFICATE	344
REMOVE-QADPRIVATEKEY	350
GET-QADCERTIFICATEREVOCATIONLIST	355
ADD-QADCERTIFICATEREVOCATIONLIST	356
IMPORT-QADCERTIFICATEREVOCATIONLIST	357
EXPORT-QADCERTIFICATEREVOCATIONLIST	359
REMOVE-QADCERTIFICATEREVOCATIONLIST	361
GET-QADPKIOBJECT	363
PUBLISH-QADCERTIFICATE	365
UNPUBLISH-QADCERTIFICATE	367
PUBLISH-QADCERTIFICATEREVOCATIONLIST	369
UNPUBLISH-QADCERTIFICATEREVOCATIONLIST	371
CMDLET REFERENCE - UTILITY	373
SET-QADINACTIVEACCOUNTSPOLICY	374
GET-QADINACTIVEACCOUNTSPOLICY	376
SET-QADPROGRESSPOLICY	377
GET-QADPROGRESSPOLICY	378
CONVERT-QADATTRIBUTEVALUE	379
GET-QADPSSNAPINSETTINGS	381
SET-QADPSSNAPINSETTINGS	384
ENABLE-QADDIAGNOSTICLOG	388
DISABLE-QADDIAGNOSTICLOG	389
GET-QADDIAGNOSTICLOGSTATUS	390
USING CERTIFICATE MANAGEMENT CMDLETS	391
UNDERSTANDING DIGITAL CERTIFICATES	391




CERTIFICATE STORES AND CONTAINERS	400
CERTIFICATE MANAGEMENT CMDLETS OVERVIEW	402
CERTIFICATE STORE, CERTIFICATE, AND CRL OBJECTS	403
WORKING WITH CERTIFICATE STORES	406
ADDING CERTIFICATES TO A CERTIFICATE STORE.	410
EXPORTING CERTIFICATES FROM A CERTIFICATE STORE	417
WORKING WITH CERTIFICATE REVOCATION LISTS (CRLs)	421
MANAGING ACTIVE DIRECTORY PKI-RELATED CONTAINERS	423

Intended Audience

This document has been prepared to assist you in becoming familiar with the Quest® One ActiveRoles® Management Shell *for Active Directory*. The Administrator Guide contains the information required to install and use the Quest One ActiveRoles Management Shell *for Active Directory*. It is intended for network administrators, consultants, analysts, and any other IT professionals using the product.

Conventions

In order to help you get the most out of this guide, we have used specific formatting conventions. These conventions apply to procedures, icons, keystrokes and cross-references.

ELEMENT	CONVENTION
Select	This word refers to actions such as choosing or highlighting various interface elements, such as files and radio buttons.
Bolded text	Interface elements that appear in Quest Software products, such as menus and commands.
<i>Italic text</i>	Used for comments.
<i>Bold Italic text</i>	Used for emphasis.
Blue text	Indicates a cross-reference. When viewed in Adobe® Reader®, this format can be used as a hyperlink.
	Used to highlight additional information pertinent to the process being described.
	Used to provide Best Practice information. A best practice details the recommended course of action for the best result.
	Used to highlight processes that should be performed with care.
+	A plus sign between two keystrokes means that you must press them at the same time.
	A pipe sign between elements means that you must select the elements in that particular sequence.

About Quest Software, Inc.

Established in 1987, Quest Software (Nasdaq: QSFT) provides simple and innovative IT management solutions that enable more than 100,000 global customers to save time and money across physical and virtual environments. Quest products solve complex IT challenges ranging from [database management](#), [data protection](#), [identity and access management](#), [monitoring](#), [user workspace management](#) to [Windows management](#). For more information, visit www.quest.com.

Contacting Quest Software

Email	info@quest.com
Mail	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	www.quest.com

Refer to our Web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a Quest product and have a valid maintenance contract. Quest Support provides unlimited 24x7 access to our Support Portal at www.quest.com/support.

From our Support Portal, you can do the following:

- Retrieve thousands of solutions from our Knowledge Base
- Download the latest releases and service packs
- Create, update and review Support cases

View the **Global Support Guide** for a detailed explanation of support programs, online services, contact information, policies and procedures. The guide is available at: www.quest.com/support.

Introduction

Quest® One ActiveRoles® Management Shell *for Active Directory* is an Active Directory-specific automation and scripting shell that provides a command-line management interface for administering directory data either via Quest One ActiveRoles or by directly accessing Active Directory domain controllers. ActiveRoles Management Shell is built on Microsoft Windows PowerShell technology.

This document is designed to introduce new users to ActiveRoles Management Shell. The document provides information on the basic concepts and features of ActiveRoles Management Shell, and includes reference topics about the commands (cmdlets) that can be run in ActiveRoles Management Shell. The document examines:

- Installing and using ActiveRoles Management Shell
- ActiveRoles Management Shell command-line tools

ActiveRoles Management Shell is implemented as a Windows PowerShell snap-in, providing an extension to the Windows PowerShell environment. To get acquainted with the basic features of Windows PowerShell, refer to the Windows PowerShell Getting Started Guide, which you can access at <http://msdn.microsoft.com/en-us/library/aa973757.aspx>. For more detailed information on Windows PowerShell, see the Windows PowerShell Primer document, which is included with the Windows PowerShell installation.

As the commands provided by ActiveRoles Management Shell conform to the Windows PowerShell standards, and are fully compatible with the default command-line tools that come with Windows PowerShell, the information found in the above documents of Microsoft's is fully applicable to ActiveRoles Management Shell.

Using ActiveRoles Management Shell

Quest One ActiveRoles Management Shell *for Active Directory*, built on Microsoft Windows PowerShell technology, provides a command-line interface that enables automation of directory data-related administrative tasks. With ActiveRoles Management Shell, administrators can manage directory objects such as users and groups. Thus, they can create new users and groups, modify user properties, and add or remove members from groups.

The management operations are performed either via the Quest One ActiveRoles proxy service or by directly accessing directory data on domain controllers. In both cases, ActiveRoles Management Shell provides a flexible scripting platform that can reduce the complexity of current Microsoft Visual Basic scripts. Tasks that previously required many lines in Visual Basic scripts can now be done by using as little as one line of code in ActiveRoles Management Shell.

By accessing the directory services through the Quest One ActiveRoles proxy service, ActiveRoles Management Shell makes it possible to take full advantage of the security, workflow integration and reporting benefits of Quest One ActiveRoles. In this way, the directory data modifications made by ActiveRoles Management Shell are supplemented and restricted by the data validation, provisioning and deprovisioning rules enforced by Quest One ActiveRoles.

The ActiveRoles Management Shell command-line tools (cmdlets), like all the Windows PowerShell cmdlets, are designed to deal with objects—structured information that is more than just a string of characters appearing on the screen. The cmdlets do not use text as the basis for interaction with the system, but use an object model that is based on the Microsoft .NET platform. In contrast to traditional, text-based commands, the cmdlets do not require the use of text-processing tools to extract specific information. Rather, you can access portions of the data directly by using standard Windows PowerShell object manipulation commands.

Installing and Opening ActiveRoles Management Shell

Installation Requirements

Before you install ActiveRoles Management Shell, ensure that your system has the following software installed:

- Microsoft Windows XP Service Pack 3, Microsoft Windows Server 2003 Service Pack 2, Windows Vista Service Pack 2, Windows Server 2008 Service Pack 2, or a later version of the Microsoft Windows operating system
- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Windows PowerShell 2.0

If you are planning to manage Terminal Services user properties by using cmdlets on a Windows XP or Windows Vista based computer, then you have to install additional administration tools on that computer:

- On a computer running a 32-bit edition of Windows XP, install Windows Server 2003 Service Pack 2 Administration Tools Pack for x86 editions, available for download at <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=86b71a4f-4122-44af-be79-3f101e533d95>

- On a computer running a 64-bit edition of Windows XP, install Windows Server 2003 Service Pack 2 Administration Tools Pack for x64 editions, available for download at <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=514bd06e-f3bc-4054-8429-c49f51e2190b>
- On a computer running a 32-bit edition of Windows Vista, install Microsoft Remote Server Administration Tools for Windows Vista, available for download at <http://www.microsoft.com/downloads/details.aspx?FamilyID=9ff6e897-23ce-4a36-b7fc-d52065de9960&DisplayLang=en>
- On a computer running a 64-bit edition of Windows Vista, install Microsoft Remote Server Administration Tools for Windows Vista for x64-based Systems, available for download at <http://www.microsoft.com/downloads/details.aspx?FamilyID=d647a60b-63fd-4ac5-9243-bd3c497d2bc5&DisplayLang=en>

In addition, on a Windows Vista based computer, you need to configure the administration tools to enable ADSI Terminal Services extensions. For instructions, see <http://blogs.technet.com/askds/archive/2008/03/31/rsat-and-aduc-getting-the-terminal-services-tabs-to-appear-in-ad-users-and-computers.aspx>

Installing Microsoft .NET Framework

For instructions on how to download and install Microsoft .NET Framework 3.5 Service Pack 1, visit Microsoft Download Center at microsoft.com/downloads/en/details.aspx?FamilyID=ab99342f-5d1a-413d-8319-81da479ab0d7

Installing Microsoft Windows PowerShell

ActiveRoles Management Shell requires Windows PowerShell 2.0, a part of Windows Management Framework Core. Visit the Microsoft Support page “Windows Management Framework Core package (Windows PowerShell 2.0 and WinRM 2.0)” at support.microsoft.com/?kbid=968930 to download the update appropriate to your Windows operating system, and then install the update. Computers running the Microsoft Windows Server 2008 R2 or Windows 7 operating system do not require the update.

Installing ActiveRoles Management Shell

To install ActiveRoles Management Shell

1. Run the **Setup.exe** file, included with the ActiveRoles Management Shell distribution package.
2. Follow the instructions on the installation wizard pages.

Opening ActiveRoles Management Shell

You can open ActiveRoles Management Shell by using either of the following procedures. Each procedure loads the ActiveRoles Management Shell snap-in into Windows PowerShell. If you do not load the ActiveRoles Management Shell snap-in before you run a command (cmdlet) provided by that snap-in, you will receive an error.

To open ActiveRoles Management Shell from the Programs menu

- Select **Start | All Programs | Quest Software | ActiveRoles Management Shell for Active Directory**.

To add the ActiveRoles Management Shell snap-in from Windows PowerShell

1. Start Windows PowerShell.
2. At the Windows PowerShell prompt, enter the following command:

```
Add-PSSnapin Quest.ActiveRoles.ADManagement
```

Upon the shell start, the console may display a message stating that a certain file published by Quest Software is not trusted on your system. This security message indicates that the certificate the file is digitally signed with is not trusted on your computer, so the console requires you to enable trust for the certificate issuer before the file can be run. Press either **R** (Run once) or **A** (Always run). To prevent this message from appearing in the future, it is advisable to choose the second option (**A**).

Getting Help

ActiveRoles Management Shell uses the Windows PowerShell help cmdlets to assist you in finding the appropriate information to accomplish your task. The following table provides some examples of how to use the **Get-Help** and **Get-Command** cmdlets to access the help information that is available for each cmdlet in ActiveRoles Management Shell.

COMMAND	DESCRIPTION
Get-Help	When you use Get-Help without any parameters, you are presented with basic instructions on how to use the help system in Windows PowerShell, including Help for ActiveRoles Management Shell.
Get-Help <Cmdlet>	When you use Get-Help with the name of a cmdlet as an argument, you are presented with the help information for that cmdlet. For example, to retrieve the help information for the Connect-QADService cmdlet, use the following command: <ul style="list-style-type: none">• Get-Help Connect-QADService
Get-Command	Get-Command without any parameters lists all the cmdlets that are available to the shell. You can use the Get-Command cmdlet with the Format-List or Format-Table cmdlet to provide a more readable display. For example, use Get-Command Format-List to display the output in a list format.
Get-Command <Cmdlet>	When you use Get-Command with the name of a cmdlet as an argument, you are presented with information about the parameters and other components of that cmdlet. The <Cmdlet> entry allows for wildcard character expansion. For example, to retrieve information about the cmdlets with the names ending in Member , you can use the following command: <ul style="list-style-type: none">• Get-Command *Member
Get-Command -Noun <CmdletNoun>	Get-Command -Noun <CmdletNoun> lists all the cmdlets with the names that include the specified noun. <CmdletNoun> allows for wildcard character expansion. Thus, you can use the following command to list all the cmdlets provided by ActiveRoles Management Shell: <ul style="list-style-type: none">• Get-Command -Noun QA*

Cmdlet Naming Conventions

All cmdlets are presented in verb-noun pairs. The verb-noun pair is separated by a hyphen (-) without spaces, and the cmdlet nouns are always singular. The verb refers to the action that the cmdlet performs. The noun identifies the entity on which the action is performed. For example, in the **Set-QADUser** cmdlet name, the verb is **Set** and the noun is **QADUser**. All ActiveRoles Management Shell cmdlets have the nouns prefixed with **QA**, to distinguish the ActiveRoles Management Shell cmdlets from those provided by Windows PowerShell itself or by other PowerShell snap-ins.

You can use the following command to list all cmdlets found in ActiveRoles Management Shell:

```
get-command Quest.ActiveRoles.ADManagement\*
```

Tab Expansion to Auto-complete Names

ActiveRoles Management Shell provides a way to complete command and parameter names automatically, thus speeding up command entry. You can fill in cmdlet names and parameters by pressing the TAB key.

To use tab expansion on a cmdlet name, type the entire first part of the name (the verb) and the hyphen that follows it, and then press TAB. The shell will complete the cmdlet name if a matching cmdlet is found. If multiple matching cmdlet names exist, repeatedly pressing TAB will cycle through all of the available choices. You can fill in more of the name for a partial match. The following example shows how you can use tab expansion when you enter a cmdlet name:

```
New-QAD <TAB>
```

As you press the TAB key in this example, the shell cycles through all the cmdlet names that begin with **New-QAD**.

You can also use tab expansion when you want the shell to complete the partial parameter name that you have entered. In this case, you must specify the full cmdlet name, either by typing it in directly or by using tab expansion. The following example shows how you can use tab expansion when you enter a parameter name:

```
Add-QADGroupMember -m <TAB>
```

As you press the TAB key in this example, the shell completes the *Member* parameter on the **Add-QADGroupMember** cmdlet.

Parameters

Cmdlets use parameters to take information necessary for completing their tasks. Parameters are string elements that follow the name of a cmdlet, either identifying an object and its attributes to act upon, or controlling how the cmdlet performs its task. The name of the parameter is preceded by a hyphen (-) and followed by the value of the parameter as follows:

```
Verb-Noun -ParameterName <ParameterValue>
```

In this example, the hyphen in front of the parameter name indicates that the word immediately following the hyphen is a parameter passed to the cmdlet and the next separate string after the parameter name is the value of the parameter.

Parameter Details

The information displayed by the **Get-Help** cmdlet includes the Parameters section (also called metadata) on each parameter. The following example is an excerpt from the output of the **Get-Help Connect-QADService -Full** command:

```
PARAMETERS
    -Proxy

        Required?                false
        Position?                named
        Default value
        Accept pipeline input?    false
        Accept wildcard characters? false
```

This example from the **Connect-QADService** cmdlet includes some very specific details about the *Proxy* parameter. Some cmdlets may not include such details. However, most cmdlets do include some settings for each parameter as described in the following table.

SETTING	DESCRIPTION
Required?	Indicates whether the cmdlet will run if you do not supply the parameter. When <i>Required?</i> is set to True, the shell prompts you for the parameter if you do not supply a value for this parameter.
Position?	Indicates whether you must specify the parameter name in front of the parameter value. When <i>Position?</i> is set to Named, the parameter name is required. When <i>Position?</i> is set to an integer, the name is not required, only the value (see "Positional Parameters" later in this section).
Default value	Indicates the default value for this parameter if no other value is provided.
Accept pipeline input?	Indicates whether the parameter can receive its value as an input through a pipeline from another cmdlet (see "Pipelining" later in this document).
Accept wildcard characters?	Indicates whether the value of this parameter can contain wildcard characters and can be matched to multiple objects.

Positional Parameters

A positional parameter lets you specify the parameter's value without specifying the parameter's name. A positional parameter has the Position attribute set to an integer in the metadata. This integer indicates the position on the command line where the cmdlet can find the parameter's value.

An example of a positional parameter is the *Identity* parameter. This parameter is always in position 1 if it is available on a cmdlet. The following two commands perform the same task: resetting the password for the user identified by the logon name in the form domain\name:

```
Set-QADUser -Identity 'domain\jsmith' -UserPassword 'P@ssword'
Set-QADUser 'domain\jsmith' -UserPassword 'P@ssword'
```

If a parameter is not a positional parameter, it is considered to be a named parameter. When you enter a command on the command line, you must type the parameter name for a named parameter.

Switch Parameters

Switch parameters are used to set a state for the execution of a cmdlet. A switch parameter does not require a value. If you specify a switch parameter on a command line, the parameter evaluates to True. If you do not specify a switch parameter, it evaluates to False. For example, the *Proxy* parameter on the **Connect-QADService** cmdlet allows you to specify whether to access directory data via Quest One ActiveRoles (*-Proxy* is added on the command line) or by connecting directly to a domain controller (*-Proxy* is omitted).

Identity

The *Identity* parameter is used to specify one of the unique identifiers that refer to a particular object in the directory service. This lets you perform actions on a specific directory object, such as a particular user or group.

The primary unique identifier of an object is always a GUID—a 128-bit identifier, such as 7f5bfccd-fd08-49f5-809d-9ee2f9d7e845. This identifier never repeats and is therefore always unique. However, since a GUID is not easy to type, the *Identity* parameter also accepts values of other identifiers that are unique across a set of objects. Depending on the object you refer to, these could be the distinguished name (DN), security identifier (SID), user principal name (UPN), or pre-Windows 2000 user logon name or group name in the form Domain\Name.

The *Identity* parameter is also considered a positional parameter. The first argument on a cmdlet is assumed to be the *Identity* parameter when no parameter name is specified. This reduces the number of keystrokes when you type commands. For more information about positional parameters, see “Positional Parameters” earlier in this section.

Type of Identifier

When you specify a value for the *Identity* parameter, a cmdlet uses a certain heuristic process to determine the type of the identifier. To avoid ambiguities and improve performance, you can add a prefix to the parameter value in order to explicitly specify the type of the identifier:

```
-Identity '<prefix>=<identifier>'
```

These prefixes are also supported by other parameters that accept object identifiers as parameter values, such as the *SearchRoot*, *ParentContainer*, or *Member* parameter.

The following table lists the supported prefixes.

PREFIX	TREAT THE IDENTIFIER AS
upn	User principal name (UPN). Example: 'upn=user@domain'
dn	Distinguished name (DN). Example: 'dn=cn=user,dc=domain'
account	Pre-Windows 2000 logon name or group name in the form domain\name. Example: 'account=domain\user'
canonical	Canonical name in the form domain/container/.../name. Example: 'canonical=domain/users/user'
sid	Security identifier (SID). Example: 'sid=S-1-5-21-1216921794-1536856817-1513834708-1267'

PREFIX	TREAT THE IDENTIFIER AS
guid	Globally unique identifier (GUID). Example: 'guid=4F881367-74A0-4CED-B9FB-25620A5D40ED'
anr	A value to be resolved using ambiguous name resolution (ANR). Normally, ANR supports the following attributes: <ul style="list-style-type: none"> • displayName • givenName (First Name) • sn (Last Name) • legacyExchangeDN • physicalDeliveryOfficeName (Office Location) • proxyAddresses • name (RDN) • sAMAccountName (pre-Windows 2000 logon name)

Syntax

ActiveRoles Management Shell follows the Windows PowerShell command conventions that help you understand what information is required or optional when you run a cmdlet and how you must present the parameters and their values. The following table lists these command conventions.

SYMBOL	DESCRIPTION
-	A hyphen indicates that the next word on the command line is a parameter. For more information about parameters, see "Parameters" earlier in this document.
< >	<p>Angle brackets are used to indicate parameter values along with the parameter type setting. This setting specifies the form that the parameter's value should take, and refers to the .NET type that determines the kind of value that is permitted as a parameter argument. For example, <Int32> indicates that the parameter argument must be an integer; <String> indicates that the argument must be in the form of a character string. If the string contains spaces, the value must be enclosed in quotation marks or the spaces must be preceded by the escape character (`).</p> <p>The angle brackets are only intended to help you understand how a command should be constructed. You do not type these brackets when you enter the command on the command line.</p>
[]	<p>Square brackets are used to indicate an optional parameter and its value. A parameter and its value that are not enclosed in square brackets are required. If you do not supply a required parameter on the command line, the shell prompts you for that parameter.</p> <p>The square brackets are only intended to help you understand how a command should be constructed. You do not type these brackets when you enter the command on the command line.</p>

In the Help documentation, all cmdlets display their associated parameters in parameter sets. These are groupings of parameters that can be used with each other. Although a cmdlet may have multiple parameter sets, most cmdlets have only one set of parameters. The following example displays the parameter set of the **Add-QADGroupMember** cmdlet:

```
Add-QADGroupMember [[-Identity] <String>] [-Proxy]
[-Service <String>] [ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<AdsiConnection>] -Member <String>
```

In this example:

- The name of the *Identity* parameter is enclosed in square brackets to indicate that you can specify the string value for this parameter without typing `-Identity` (this is a *positional parameter*, see “Parameters” earlier in this document).
- Since *Identity* is an optional parameter with this cmdlet, the `[-Identity] <string>` token is enclosed in square brackets.
- The *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, and *Connection* parameters along with their parameter values are enclosed in square brackets, to indicate that these are optional parameters, so each of these parameters along with their values can be omitted.
- *Member* is a required parameter, and thus it must be specified along with its string value, so the parameter name and value are not enclosed in square brackets.

Pipelining

The term *pipelining* refers to the act of having one cmdlet use the output of another cmdlet when it performs an operation. Pipelining is accomplished by using the pipe character (`|`). To create a pipeline, you connect cmdlets together with the pipe character. The result is that the output of the cmdlet preceding the pipe character is used as input to the cmdlet following the pipe character.

It is important to be aware that the shell does not pipe text between cmdlets. Instead, it pipes objects. From a user perspective, each object represents related information as a unit, making it easier to manipulate the information and extract specific pieces of information.

Thus, for bulk provisioning of user accounts by Quest One ActiveRoles based on data held in a text (CSV) file, you can run the following command (this command implies that the appropriate provisioning policies are configured in Quest One ActiveRoles to auto-populate the attributes, such as `sAMAccountName`, that are required for a user account to exist):

```
Import-Csv c:\temp\data.csv | ForEach-Object -Process  
{New-QADUser -Proxy -ParentContainer 'OU=User,DC=company,DC=com' -Name $_.'user name'}
```

In this example:

- The **Import-Csv** cmdlet produces a set of objects, with each object representing one of the records found in the CSV file specified, and passes (pipes) the objects to the **ForEach-Object** cmdlet.
- The **ForEach-Object** cmdlet applies the specified script block to each of the incoming (piped) objects.
- For each of the incoming objects, the script block runs the **New-QADUser** cmdlet to create a user account with the name set to the value retrieved from the **user name** property of the object. The presence of `-Proxy` ensures that the operation is performed via Quest One ActiveRoles. The script accesses the incoming object through the `$_` variable provided by Windows PowerShell.

Since the **user name** property value is the value found in the “user name” field of the CSV file record represented by the incoming object, the name of the newly created user account is appropriately set up based on the data retrieved from the CSV file.

Aliases

ActiveRoles Management Shell uses the aliasing mechanism provided by Windows PowerShell to assign concise aliases to cmdlet names and parameter names. An alias is an alternate, short name for a cmdlet or parameter. The native Windows PowerShell cmdlets have predefined, built-in aliases for cmdlet names. For example, **gcm** is an alias for **Get-Command**.

The cmdlets that come with ActiveRoles Management Shell do not have aliases for cmdlet names by default. The instructions on how to create an alias for a cmdlet name are given later in this section.

To list all cmdlet name aliases that are defined within your ActiveRoles Management Shell session, type the following command at the PowerShell command-prompt:

```
get-alias
```

To find the aliases for a cmdlet name, type:

```
get-alias | where-object {$_.definition -eq "<cmdlet-name>"}
```

For example, to find the aliases for **Get-Command**, type:

```
get-alias | where-object {$_.definition -eq "Get-Command"}
```

Aliases are helpful when you frequently use certain cmdlets and want to reduce the typing that you must do. When typing a command at the Windows PowerShell command-prompt, you can type aliases in place of cmdlet names and parameter names. For example, you may create the **gqu** alias for the **Get-QADUser** cmdlet and type **gqu** instead of typing **Get-QADUser** every time you need to use that cmdlet. You can create multiple aliases for the same cmdlet. Having aliases for a cmdlet does not prevent you from using the original name of the cmdlet.

Cmdlet parameters may also have aliases, in addition to parameter names. Parameter aliases are predefined and cannot be altered. Neither can you add your own, custom aliases for parameter names. To list all aliases defined for the parameter names specific to a certain cmdlet, type the following command at the Windows PowerShell command prompt:

```
gcm <cmdlet-name> | select -expand parametersets | select cmdname -expand parameters |
where {$_.aliases} | sort name | %{ $lc=""; $lp="" } {if ($lp -ne $_.Name) { $lp = $_.Name;
$_}} | ft name,aliases -auto
```

For example, to find the parameter aliases specific to the **Get-QADUser** cmdlet, type:

```
get-Command Get-QADUser | select -expand parametersets | select cmdname -expand
parameters | where {$_.aliases} | sort name | %{ $lc=""; $lp="" } {if ($lp -ne $_.Name) { $lp
= $_.Name; $_}} | ft name,aliases -auto
```

This command produces a two-column list (see the excerpt below), with parameter names listed in the first column. For each parameter name, the second column displays the alias (or aliases) that can be used in place of the parameter name. Thus, as shows the example below, when you want to use the **ConnectionAccount** parameter, you may type **User** or **ca**.

Name	Aliases
AttributeScopeQuery	{ASQ}
City	{l}
ConnectionAccount	{User, ca}
ConnectionPassword	{Pwd, cp}
Credential	{Cred}
Department	{dept}

DisplayName	{disp}
FirstName	{givenName, fn}
HomePhone	{hp}
Initials	{i}
LastName	{sn, ln}
LdapFilter	{lf}
Manager	{mgr}
MobilePhone	{mobile}

Creating an Alias for a Cmdlet Name

To create aliases for cmdlet names, use the **Set-Alias** cmdlet. For example, to create the **gqu** alias for **Get-QADUser**, type:

```
set-alias gqu get-qaduser
```

If you no longer need an alias, you can delete it by using the **Remove-Item** cmdlet to delete the alias from the **Alias:** drive. For example, to delete the **gqu** alias, type:

```
remove-item alias:gqu
```

Adding an Alias to a Windows PowerShell Profile

Aliases that are created from the command line by using the **Set-Alias** cmdlet during an ActiveRoles Management Shell session can be used when the session is active. After the session is closed, the alias definition is lost. To make your custom alias persistent and available every time that a new ActiveRoles Management Shell session is opened, you have to add the alias definition to your Windows PowerShell profile.

So, to retain your alias definitions, you should add the appropriate set-alias commands to a Windows PowerShell profile. The profile is loaded every time that Windows PowerShell starts.

To load a profile, your Windows PowerShell execution policy must permit you to load configuration files. If it does not, the attempt to load the profile fails and Windows PowerShell displays an error message.

The default execution policy, **Restricted**, does not permit any configuration files, including a Windows PowerShell profile, to be loaded. However, if you want to load configuration files, you can change the execution policy on your system. For information and instructions, type:

```
get-help about_signing
```

To see what the execution policy is in effect on your system, type:

```
get-executionpolicy
```

To change the execution policy on your system, use the **Set-ExecutionPolicy** cmdlet. For example, to enable the loading of Windows PowerShell profiles, change the execution policy to **RemoteSigned**. To do this, type the following command at the Windows PowerShell command-prompt:

```
set-executionpolicy remotesigned
```

Creating and Editing the Windows PowerShell User Profile

A Windows PowerShell user profile is not created automatically. The location of this profile is stored in the **\$profile** variable, so you can determine if your user profile has been created by typing:

```
test-path $profile
```

If the profile exists, the response is **True**; otherwise, it is **False**.

To create your user profile, type:

```
new-item -path $profile -itemtype file -force
```

To open the profile in Notepad, type:

```
notepad $profile
```

Add the set-alias commands to the text in Notepad, one command per string (for example, **set-alias gqu get-qaduser**), save your changes (press Ctrl+S), and then close Notepad. Your alias definitions will be loaded every time that you open ActiveRoles Management Shell.

What's New in Version 1.6

ActiveRoles Version Compatibility

Version 1.6 of ActiveRoles Management Shell requires Windows PowerShell 2.0. Windows PowerShell 1.0 is no longer supported.

Version 1.6 of ActiveRoles Management Shell is compatible with Quest One ActiveRoles of version 6.8 only. This has the following implications:

- Version 1.6 of ActiveRoles Management Shell cannot be installed on a computer that hosts ActiveRoles components of version other than 6.8. For example, if version 6.7 of the Administration Service, MMC Interface or Web Interface is installed on a computer, you receive an error when attempting to install version 1.6 of ActiveRoles Management Shell on that computer. The error message informs you of a conflict with the software that exists on the computer, and advises you to install ActiveRoles Management Shell on a different computer.
- The ActiveRoles Management Shell cmdlets of version 1.6 cannot connect to the ActiveRoles Administration Service of version other than 6.8. For example, if you use the *Proxy* connection parameter in conjunction with the *Service* parameter that specifies a computer running the ActiveRoles Administration Service of version 6.7, you receive an error. The error message informs you that the connection cannot be established because of version incompatibility.

When upgrading the ActiveRoles Administration Service to version 6.8, be aware that version 6.8 of the Administration Service requires ActiveRoles Management Shell of version 1.6. The Administration Service Setup program installs version 1.6 of ActiveRoles Management Shell during the upgrade process so that the resulting installation meets the version compatibility requirements. It is advisable to upgrade the Administration Service by running the Setup program that is included on the Quest One ActiveRoles distribution media, rather than by running the .msi file directly.

What's New in Version 1.4

Version 1.6 of ActiveRoles Management Shell inherits all the new features introduced in version 1.4. This topic summarizes the key new features of version 1.4.

Cmdlets

The following new cmdlets are available in ActiveRoles Management Shell version 1.4 (see cmdlet descriptions in the cmdlet reference, later in this document):

- Get-QADLocalCertificateStore
- New-QADLocalCertificateStore
- Remove-QADLocalCertificateStore
- Get-QADCertificate
- Where-QADCertificate
- Add-QADCertificate
- Import-QADCertificate
- Show-QADCertificate
- Edit-QADCertificate
- Export-QADCertificate
- Remove-QADCertificate
- Remove-QADPrivateKey
- Get-QADCertificateRevocationList
- Add-QADCertificateRevocationList
- Import-QADCertificateRevocationList
- Export-QADCertificateRevocationList
- Remove-QADCertificateRevocationList
- Get-QADPKIObject
- Publish-QADCertificate
- Unpublish-QADCertificate
- Publish-QADCertificateRevocationList
- Unpublish-QADCertificateRevocationList
- Add-QADProxyAddress
- Set-QADProxyAddress
- Remove-QADProxyAddress
- Clear-QADProxyAddress
- Enable-QADEmailAddressPolicy
- Disable-QADEmailAddressPolicy
- Set-QADProgressPolicy
- Get-QADProgressPolicy
- Set-QADInactiveAccountsPolicy
- Get-QADInactiveAccountsPolicy

Parameters

The following table summarizes the parameters added on certain cmdlets in ActiveRoles Management Shell version 1.4 (see parameter descriptions for the respective cmdlets in the cmdlet reference, later in this document).

PARAMETERS	ADDED ON
<ul style="list-style-type: none"> ExpiredFor Inactive InactiveFor NotLoggedOnFor PasswordNotChangedFor 	<ul style="list-style-type: none"> Get-QADUser
<ul style="list-style-type: none"> Inactive InactiveFor NotLoggedOnFor PasswordNotChangedFor 	<ul style="list-style-type: none"> Get-QADComputer
<ul style="list-style-type: none"> PrimaryProxyAddress ProxyAddress SecondaryProxyAddress 	<ul style="list-style-type: none"> Get-QADObject Get-QADGroup Get-QADUser
<ul style="list-style-type: none"> Activity ProgressThreshold ShowProgress 	<ul style="list-style-type: none"> Get-QADComputer Get-QADGroup Get-QADGroupMember Get-QADManagedObject Get-QADMemberOf Get-QADObject Get-QADPasswordSettingsObject Get-QADPasswordSettingsObjectAppliesTo Get-QADUser Get-QARSAccessTemplate Get-QARSAccessTemplateLink Get-QARSWorkflowDefinition
<ul style="list-style-type: none"> Disabled Enabled KeepForeignSecurityPrincipals 	<ul style="list-style-type: none"> Get-QADGroupMember
<ul style="list-style-type: none"> ResolveForeignSecurityPrincipals 	<ul style="list-style-type: none"> Get-QADObject

PARAMETERS	ADDED ON
<ul style="list-style-type: none"> Control 	<ul style="list-style-type: none"> Add-QADGroupMember Add-QADMemberOf Add-QADPasswordSettingsObjectAppliesTo Deprovision-QADUser Disable-QADComputer Disable-QADUser Enable-QADComputer Enable-QADUser Get-QADComputer Get-QADGroup Get-QADGroupMember Get-QADManagedObject Get-QADMemberOf Get-QADObject Get-QADPasswordSettingsObject Get-QADPasswordSettingsObjectAppliesTo Get-QADUser Get-QARSAccessTemplate Get-QARSAccessTemplateLink Get-QARSWorkflowDefinition Move-QADObject New-QADComputer New-QADGroup New-QADObject New-QADPasswordSettingsObject New-QADUser New-QARSAccessTemplateLink Publish-QARSGroup Remove-QADGroupMember Remove-QADMemberOf Remove-QADObject Remove-QADPasswordSettingsObjectAppliesTo Remove-QARSAccessTemplateLink Rename-QADObject Reset-QADComputer Restore-QADDeletedObject Set-QADComputer Set-QADGroup Set-QADObject Set-QADUser Set-QARSAccessTemplateLink Unlock-QADUser Unpublish-QARSGroup

Multi-value SearchRoot Parameter

The data type of the SearchRoot parameter has been changed to support a search within two or more containers at a time. This parameter now accepts an array of objects rather than a single object as it was with earlier versions. This makes it possible for the cmdlet to search multiple containers identified by the SearchRoot parameter value. For example, you can supply an array of strings each of which represents the canonical name of a certain container, to retrieve objects from all of the containers specified.

The following table lists the cmdlets that have the SearchRoot data type changed.

PARAMETER	DATA TYPE IN VERSION 1.4	USED IN CMDLET
SearchRoot	IdentityParameter[]	<ul style="list-style-type: none">• Get-QADObject• Get-QADGroup• Get-QADUser• Get-QADComputer• Get-QADPasswordSettingsObject• Get-QARSWorkflowDefinition• Get-QARSAccessTemplate• Get-QARSAccessTemplateLink

Example

The following example demonstrates how to retrieve user accounts that reside in any of the two specified containers - corp.company.com/employees or corp.company.com/contractors:

```
C:\PS> Get-QADUser -SearchRoot  
'corp.company.com/employees','corp.company.com/contractors'
```

Troubleshooting

In this section you can find information on some issues you may experience when using ActiveRoles Management Shell.

Script blocks in cmdlet parameter values may not work as expected

This issue applies only to the parameters that do not accept input from pipeline. Thus, the following syntax cannot be used to set a password value:

```
Get-QADUser Identity | Set-QADUser -Password {$_ .SamAccountName}
```

An alternative syntax should be used in this case:

```
Get-QADUser Identity | %{Set-QADUser $_ -Password $_.SamAccountName}
```

However, you can use a script block to specify an identity, because the Identity parameter accepts input from pipeline:

```
Import-Csv c:\test.csv | Set-QADUser -Identity {$_ .samAccountName} -Import
```

Not all membership-related parameters can be used in proxy mode

The following parameters cannot be used in conjunction with the Proxy parameter:

- ContainsIndirectMember
- NotContainsIndirectMember
- IndirectMemberOf
- NotIndirectMemberOf

If you attempt to use any of these parameters in proxy mode, you receive an error.

Not all permission management cmdlets can be used in proxy mode

The following cmdlets cannot be used in conjunction with the Proxy parameter:

- Add-QADPermission
- Remove-QADPermission

If you attempt to use any of these cmdlets in proxy mode, you receive an error: "The discretionary ACL cannot be modified as it was not retrieved from the backend store."

Cmdlet Reference - Active Directory

Here you can find information about command-line tools (cmdlets) that are provided by ActiveRoles Management Shell.

This section covers the cmdlets for managing directory data, such as user or group properties. Supported are both Active Directory Domain Services and Active Directory Lightweight Directory Services.

Requirements on Active Directory

ActiveRoles Management Shell retains most of its features and functions when managing Windows 2000 Server based Active Directory. However, certain cmdlets and parameters require Active Directory of a later version:

- **Windows Server 2003** The cmdlets and parameters that rely on the attribute scope query (ASQ) search preference require the Active Directory functional level of Windows Server 2003 or higher. These include the **Get-QADGroupMember** cmdlet and the *AttributeScopeQuery* parameter.
- **Windows Server 2008** The cmdlets for managing Password Settings objects (such as **New-QADPasswordSettingsObject** or **Add-QADPasswordSettingsObjectAppliesTo**) require Windows Server 2008 based Active Directory.

Connect-QADService

Connect to the ActiveRoles Administration Service via the Quest One ActiveRoles ADSI Provider, or to a certain Active Directory domain controller or a certain server running an Active Directory Lightweight Directory Services (AD LDS) instance via the regular LDAP ADSI Provider.

Syntax

```
Connect-QADService [[-Service] <String>] [-Proxy] [-UseGlobalCatalog]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

Parameters

Service

This is the fully qualified domain name, NetBIOS name or IP address of the computer running the Administration Service to connect to, or, if the Proxy parameter is not specified, the fully qualified domain name, NetBIOS name or IP address of the AD domain controller, or AD LDS server to connect to. In case of an AD LDS server, the fully qualified domain name of the server should be specified, with the appropriate port number added to the server name (see examples). If the DNS name of an AD domain is specified as the parameter value, then a connection is established to an appropriate domain controller in that domain.

With this parameter omitted, the Proxy parameter causes a connection to any available Administration Service. If both the Service and Proxy parameters are omitted, a connection is established to a domain controller in the domain of the computer running the cmdlet.

Proxy

If this parameter is present, the cmdlet will use the Quest One ActiveRoles ADSI Provider, so as to establish a connection using Quest One ActiveRoles. Otherwise, the regular Microsoft LDAP ADSI Provider will be used, so as to establish a direct connection to an AD domain or AD LDS server.

ConnectionAccount

This is the user logon name of the account with which you want to connect, in the form DomainName\UserName. If this parameter is omitted, a connection is established with the credentials of the account under which the cmdlet is running.

ConnectionPassword

This is the password of the user account with which you want to connect. Use this parameter in conjunction with ConnectionAccount, to connect with the credentials of an account other than that under which the cmdlet is running.

The parameter value must be a SecureString object. Use the Read-Host cmdlet provided by Windows PowerShell to pass a SecureString object to this parameter.

Credential

This is the user name and password of the user account with which you want to connect, in the form of a PSCredential object. Use the Get-Credential cmdlet provided by Windows PowerShell to pass a PSCredential object to this parameter, if you want to connect with the credentials of an account other than that under which the cmdlet is running.

Connection

With this parameter, the credentials of an earlier established connection can be re-used to establish a new connection (for example, to a different server). Save in a certain variable the object returned by the `Connect-QADService` cmdlet, and then pass that object to this parameter when establishing a new connection.

UseGlobalCatalog

This parameter directs the cmdlet to connect to a domain controller that holds the role of the Global Catalog server. When the `Proxy` parameter is supplied, `UseGlobalCatalog` has no effect.

If `UseGlobalCatalog` is supplied together with the `Service` parameter that specifies a certain domain controller, the cmdlet connects to the specified domain controller if that domain controller is a Global Catalog server. If the `Service` parameter specifies a particular domain, then `UseGlobalCatalog` causes the cmdlet to connect to any available Global Catalog server in that domain. If the `Service` parameter is omitted, then `UseGlobalCatalog` causes the cmdlet to connect to any available Global Catalog server in the domain of the computer running the cmdlet.

Detailed Description

This cmdlet establishes a connection to any available Administration Service, to a specific Administration Service, or directly to a specific Active Directory domain controller or a server running an Active Directory Lightweight Directory Services (AD LDS) instance, with the credentials of the locally logged on user or with the credentials of a specified user. A connection determines the default connection parameters (the server and the security context) for the operations that are performed by the other cmdlets. The default connection parameters are effective until the connection is closed either explicitly or by establishing a new connection, and can be overridden on a per-cmdlet basis.

The cmdlet establishes a connection in the security context of a certain user, so some user credentials must be provided in order to authenticate the user.

The cmdlet makes it possible to specify user credentials in a number of ways through the use of the credential-related parameters *ConnectionAccount*, *ConnectionPassword*, *Credential*, and *Connection*:

- If no connection-related parameters are specified, the cmdlet uses the credentials of the locally logged on user.
- If the *Credential* parameter is specified, the credentials provided by this parameter are used regardless of whether any other credential-related parameters are specified.
- If the *ConnectionAccount* and *ConnectionPassword* parameters are specified while the *Credential* parameter is omitted, the specified user name and password are passed to the cmdlet as the user credentials regardless of whether the *Connection* parameter is specified.
- If the *Connection* parameter is specified while all the other credential-related parameters are omitted, the cmdlet re-uses the credentials that were used to open the existing connection.

The object that is returned by this cmdlet can be passed as the value of the *Connection* parameter to any other cmdlet in this snap-in in order to re-use the connection parameters of the existing connection. Note that the object includes information not only about the user credentials or security context, but also about the server to which the connection is established. So, if you pass the object to a cmdlet and omit the *Service* parameter, the cmdlet will use the server specified by the object you have passed to the cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user:

```
C:\PS> connect-QADService
```

Example 2

Connect to the local Administration Service with the credentials of the locally logged on user:

```
C:\PS> connect-QADService -service 'localhost' -proxy
```

Example 3

Prompt the user for password within the console window (in text mode); then, connect to a specific domain controller with the user name and password specified:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
```

Example 4

Use a dialog box to request a user name and password; then, connect to a specific domain controller with those user name and password, and save the `AdsiConnection` object in a variable for later use:

```
C:\PS> $cred = get-credential
C:\PS> $conn = connect-QADService -service 'server.company.com' -credential $cred
```

Example 5

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user:

```
C:\PS> connect-QADService -service 'server.domain.local:389'
```

Disconnect-QADService

Close the connection, if any exists. A connection could be established by using the **Connect-QADService** cmdlet.

Syntax

```
Disconnect-QADService [-Proxy] [-UseGlobalCatalog] [-Service <String>]  
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential  
<PSCredential>] [-Connection <ArsConnection>]
```

Parameters

This cmdlet takes the same optional connection parameters as the **Connect-QADService** cmdlet. The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see "Parameters" in the "Connect-QADService" section earlier in this document.

Detailed Description

Any connection established using the **Connect-QADService** cmdlet must be finally closed by executing the **Disconnect-QADService** cmdlet. The cmdlet closes the last open connection, if any. If the *Connection* parameter is present, the cmdlet also closes the connection specified by the value of that parameter. If no connection is currently open, the cmdlet attempts to establish a connection in accordance with the connection parameters specified, and then closes the connection.

Examples

Example 1

Close the last open connection, if any:

```
C:\PS> disconnect-QADService
```

Example 2

Close the last open connection and also close the connection defined by an *AdsiConnection* object that was earlier saved in the *\$conn* variable:

```
C:\PS> disconnect-QADService -connection $conn
```

Get-QADUser

Retrieve all users in a domain or container that match the specified conditions. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Get-QADUser [[-Identity] <IdentityParameter>] [-Manager <IdentityParameter>] [-City
<String[]>] [-Company <String[]>] [-Department <String[]>] [-Fax <String[]>] [-FirstName
<String[]>] [-HomePhone <String[]>] [-Initials <String[]>] [-LastName <String[]>]
[-MobilePhone <String[]>] [-Notes <String[]>] [-Office <String[]>] [-Pager <String[]>]
[-PhoneNumber <String[]>] [-PostalCode <String[]>] [-PostOfficeBox <String[]>]
[-SamAccountName <String[]>] [-StateOrProvince <String[]>] [-StreetAddress <String[]>]
[-Title <String[]>] [-UserPrincipalName <String[]>] [-WebPage <String[]>]
[-HomeDirectory <String[]>] [-HomeDrive <String[]>] [-ProfilePath <String[]>]
[-LogonScript <String[]>] [-Email <String[]>] [-ProxyAddress <String[]>]
[-PrimaryProxyAddress <String[]>] [-SecondaryProxyAddress <String[]>] [-Disabled]
[-Enabled] [-Locked] [-AccountExpiresBefore <DateTime>] [-AccountExpiresAfter
<DateTime>] [-AccountNeverExpires] [-PasswordNeverExpires] [-Inactive] [-InactiveFor
<Int32>] [-ExpiredFor <Int32>] [-NotLoggedOnFor <Int32>] [-PasswordNotChangedFor
<Int32>] [-MemberOf <IdentityParameter[]>] [-IndirectMemberOf <IdentityParameter[]>]
[-NotMemberOf <IdentityParameter[]>] [-NotIndirectMemberOf <IdentityParameter[]>]
[-Tombstone] [-Recycled] [-LastKnownParent <IdentityParameter>] [-SecurityMask
<SecurityMasks>] [-SearchRoot <IdentityParameter[]>] [-SearchScope <SearchScope>]
[-AttributeScopeQuery <String>] [-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter
<String>] [-WildcardMode <WildcardMode>] [-SearchAttributes <Object>] [-Description
<String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr <String>] [-Control
<Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore
<DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>]
[-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has a number of optional parameters for searching by individual attributes in the directory, with each parameter name identifying a certain attribute that you can search for attribute values specified by using the respective parameter (see the list of parameters for this cmdlet).

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

AccountExpiresAfter

Retrieve user accounts that are configured to expire after a certain date. Parameter value is a `DateTime` object that specifies the date you want.

AccountExpiresBefore

Retrieve user accounts that are configured to expire before a certain date. Parameter value is a `DateTime` object that specifies the date you want.

AccountNeverExpires

Supply this parameter for the cmdlet to retrieve the user accounts that are configured to never expire.

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also `ShowProgress` and `ProgressThreshold`). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (`displayName`)
- Given-Name (`givenName`)
- Legacy-Exchange-DN (`legacyExchangeDN`)
- ms-DS-Additional-Sam-Account-Name (`msDS-AdditionalSamAccountName`)
- Physical-Delivery-Office-Name (`physicalDeliveryOfficeName`)
- Proxy-Addresses (`proxyAddresses`)
- RDN (`name`)
- SAM-Account-Name (`sAMAccountName`)
- Surname (`sn`)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the `SearchRoot` parameter, and performs the search on the objects represented by the Distinguished Names. The `SearchScope` parameter has no effect in this case. The object to search must be specified by using the `SearchRoot` parameter rather than the `Identity` parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Disabled

Supply this parameter for the cmdlet to search for disabled accounts.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values “as is,” without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Enabled

Supply this parameter for the cmdlet to retrieve only those accounts that are enabled (not disabled).

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExpiredFor

Use this parameter to retrieve accounts that remain in the expired state for at least the number of days specified by the parameter value. This parameter overrides the expiry-related inactivity condition of the Inactive or InactiveFor parameter. Thus, if the ExpiredFor value of 0 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are currently expired, or have the password age of 30 or more days, or have not been used to log on for 30 or more days.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

Inactive

Supply this parameter to retrieve accounts that meet the default inactivity conditions. You can view or change the default inactivity conditions by using the Get-QADInactiveAccountsPolicy or Set-QADInactiveAccountsPolicy cmdlet, respectively. When considering whether an account is inactive, the cmdlet verifies each of these values:

- The number of days that the account remains in the expired state
- The number of days that the password of the account remains unchanged
- The number of days that the account remains unused for logon

If any of these values exceeds a certain, default limit, then the account is considered inactive, and thus is retrieved by the Inactive parameter. The default limits can be overridden by supplying other account-inactivity related parameters, such as InactiveFor, ExpiredFor, NotLoggedOnFor, and PasswordNotChangedFor. Thus, if the NotLoggedOnFor value of 60 is supplied in conjunction with the Inactive parameter, the cmdlet searches for accounts that meet the default expiry-related or password-related inactivity condition, or have not been used to log on for 60 or more days.

To retrieve only those accounts that are not inactive, use the following syntax: -Inactive:\$false

InactiveFor

Use this parameter to retrieve accounts that meet any of the following conditions:

- The account remains in the expired state for at least the number of days specified by the parameter value
- The account does not have its password changed for at least the number of days specified by the parameter value
- The account has not been used to log on for at least the number of days specified by the parameter value

For example, the parameter value of 30 causes the cmdlet to search for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 30 or more days.

The value of this parameter overrides the default inactivity conditions, so the `Inactive` parameter has no effect when used together with this parameter. Similarly, the other account-inactivity related parameters such as `ExpiredFor`, `NotLoggedOnFor` and `PasswordNotChangedFor` override the corresponding conditions of this parameter. Thus, if the `NotLoggedOnFor` value of 60 is supplied in conjunction with the `InactiveFor` value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 60 or more days.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the `SerializeValues` parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the `Get-QADUser` or `Get-QADObject` cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the `Get-QADPSSnapinSettings` cmdlet, respectively. Using the `IncludedProperty` parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

IndirectMemberOf

Retrieve objects that belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has direct or indirect membership in the group specified by this parameter value.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both `LastChangedAfter` and `LastChangedBefore` bounds a time interval for the objects' last change. If you supply only `LastChangedAfter`, there is no upper boundary on the date. Parameter value is a `DateTime` object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both `LastChangedAfter` and `LastChangedBefore` bounds a time interval for the objects' last change. If you supply only `LastChangedBefore`, there is no lower boundary on the date. Parameter value is a `DateTime` object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the `LastChangedAfter` and `LastChangedBefore` parameters. Parameter value is a `DateTime` object that specifies the date you want.

LastKnownParent

When searching for a deleted object by using the Tombstone parameter, specify the DN of the container the object was in before it became a tombstone. This allows you to find objects that were deleted from a particular container.

Note that the lastKnownParent attribute is only set if the object was deleted on a domain controller running Windows Server 2003 or later version of Microsoft Windows Server. Therefore, it is possible that the lastKnownParent attribute value is inaccurate.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Locked

Supply this parameter for the cmdlet to retrieve only those accounts that are currently in the "locked" state.

MemberOf

Retrieve objects that are direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object has direct membership in the group specified by this parameter value.

NotIndirectMemberOf

Retrieve objects that do not belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has neither direct nor indirect membership in the group specified by this parameter value.

NotLoggedOnFor

Use this parameter to retrieve accounts that have not been used to log on for at least the number of days specified by the parameter value. This parameter overrides the logon-related inactivity condition of the Inactive or InactiveFor parameter. Thus, if the NotLoggedOnFor value of 60 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 60 or more days.

NotMemberOf

Retrieve objects that are not direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object does not have direct membership in the group specified by this parameter value.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

PasswordNeverExpires

Supply this parameter for the cmdlet to retrieve the user accounts that have the password configured to never expire.

PasswordNotChangedFor

Use this parameter to retrieve accounts whose password has not been changed for at least the number of days specified by the parameter value. This parameter overrides the password-related inactivity condition of the Inactive or InactiveFor parameter. Thus, if the PasswordNotChangedFor value of 60 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 60 or more days, or have not been used to log on for 30 or more days.

PrimaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a primary (reply-to) e-mail address.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients that have any of the specified e-mail addresses.

Recycled

This parameter has an effect only if all of the following conditions are true:

- A domain is supplied as the SearchRoot parameter value.
- Active Directory Recycle Bin is enabled in that domain.

You can use this parameter in conjunction with the Tombstone parameter for the search results to include both the deleted and recycled objects that meet the search conditions. Without this parameter, the cmdlet returns only deleted objects.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SecondaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a non-primary e-mail address.

SecurityMask

Specify which elements of the object's security descriptor to retrieve. Valid parameter values are:

- 'None' - do not retrieve any security data
- 'Owner' - retrieve the owner data
- 'Group' - retrieve the primary group data
- 'Dacl' - retrieve the discretionary access-control list data
- 'Sacl' - retrieve the system access-control list data

You can supply a combination of these values, separating them by commas. For example, you can supply the parameter value of 'Dacl,Sacl' in order to retrieve both the discretionary and system access-control list data.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Tombstone

Search for deleted objects of the respective object class. The search output is normally intended to be passed (piped in) to the Restore-QADDeletedObject cmdlet for restoring deleted objects.

In a domain with Active Directory Recycle Bin (a feature of Windows Server 2008 R2) this parameter retrieves deleted objects (rather than tombstones, which in that case are referred to as recycled objects). Recycle Bin preserves all attributes on the deleted objects, so you can use a search filter based on any attributes.

In a domain without Active Directory Recycle Bin, deleting an object converts that object to a tombstone. A search using this parameter returns tombstone objects that meet the filtering criteria supplied. Upon deletion of an object only a small number of the object's attributes are saved in the tombstone, with most of the attributes being lost. To search for deleted objects, your search filter should be based on the attributes that are preserved in tombstones.

When the Tombstone parameter is supplied, the search results include the deleted objects or tombstones that match the specified search filter. However, a search filter that matches a live object may not work as expected after the object is deleted. This is because not all attributes are retained in the tombstone. For example, a filter such as (&(objectClass=user)(objectCategory=person)) would not match any tombstone objects since the objectCategory attribute is removed upon object deletion. Conversely, the objectClass attribute is retained on tombstone objects, so a filter of (objectClass=user) would match deleted user objects.

The name of a tombstone object begins with the name of the deleted object, so a search using the Tombstone parameter can be refined by adding a filter based on object name. For example, to search for deleted objects with a name that begins with "John", you can use a filter such as (cn=John*).

It is also possible to find a specific deleted object. If you know the name of the object and the Distinguished Name (DN) of the container the object was in before it was deleted, then you can pass the container's DN to the LastKnownParent parameter and apply a filter of (cn=<name of the object>*) in order to have the cmdlet retrieve that specific object. However, if an object is deleted, a new object with the same DN is created, and then deleted as well, the above search would return more than one object. The returned objects are distinguished by the GUIDs of the deleted objects, with the name of each ending in the GUID of the respective deleted object.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Attribute-specific Parameters

The cmdlet takes a series of optional, attribute-specific parameters allowing you to search by user attributes. The attribute-specific parameters have effect if SearchRoot is specified whereas Identity is not. If you specify SearchRoot only, then the cmdlet returns all users found in the SearchRoot container.

You can use attribute-specific parameters to search for user accounts that have specific values of certain attributes. Thus, to find all user accounts that have the "givenName" attribute set to Martin, you may add the following on the command line: -FirstName Martin. To search for user accounts that have a certain attribute not set specify "" (empty string) as the parameter value.

If a particular attribute is referred to by both the SearchAttributes array and the attribute-specific parameter, the SearchAttributes setting has no effect on that attribute. The cmdlet searches for the attribute value specified by the attribute-specific parameter.

With more than one attribute-specific parameter supplied, the search conditions are combined by using the AND operator, so as to find the user accounts that meet all the specified conditions. Thus, if you supply both the *FirstName* and *LastName* parameters, the cmdlet searches for the user accounts that have the "givenName" attribute set to the FirstName parameter value and the "sn" attribute set to the LastName parameter value.

Each of these parameters accepts the asterisk (*) wildcard character in the parameter value to match zero or more characters (case-insensitive). For instance, a* matches A, ag, Amsterdam, and does not match New York.

The following table lists the attribute-specific parameters you can use with this cmdlet. Each parameter adds a filter condition based on a certain attribute identified by the LDAP display name in the table.

TO SEARCH BY THIS ATTRIBUTE...	USE THIS SYNTAX
l	-City <String[]>
company	-Company <String[]>
description	-Description <String[]>
department	-Department <String[]>
displayName	-DisplayName <String[]>
facsimileTelephoneNumber	-Fax <String[]>
givenName	-FirstName <String[]>
homeDirectory	-HomeDirectory <String[]>
homeDrive	-HomeDrive <String[]>
homePhone	-HomePhone <String[]>
initials	-Initials <String[]>
sn	-LastName <String[]>
mail	-Email <String[]>
manager	-Manager <IdentityParameter[]>
mobile	-MobilePhone <String[]>
name	-Name <String[]>

TO SEARCH BY THIS ATTRIBUTE...	USE THIS SYNTAX
info	-Notes <String[]>
physicalDeliveryOfficeName	-Office <String[]>
pager	-Pager <String[]>
telephoneNumber	-Phone <String[]>
postalCode	-PostalCode <String[]>
postOfficeBox	-PostOfficeBox <String[]>
profilePath	-ProfilePath <String[]>
samAccountName	-SamAccountName <String[]>
scriptPath	-LogonScript <String[]>
st	-StateOrProvince <String[]>
streetAddress	-StreetAddress <String[]>
title	-Title <String[]>
userPrincipalName	-UserPrincipalName <String[]>
wwwHomePage	-WebPage <String[]>

Detailed Description

Use this cmdlet to search an Active Directory domain or container for user accounts that meet certain criteria, or to bind to a certain user account by DN, SID, GUID, UPN, or Domain\Name. You can search by user attributes or specify your search criteria by using an LDAP search filter.

The output of the cmdlet is a collection of objects, with each object representing one of the user accounts found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADUser**, to make changes to the user accounts returned by this cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific user account by Domain\Name, and display the user description. In this example, the NetBIOS name of the domain is assumed to be "MyDomain" and the pre-Windows 2000 name of the user account is assumed to be "MyLogonName":

```
C:\PS> (get-QADUser 'MyDomain\MyLogonName').DirectoryEntry.description
```

Example 2

Connect to a specific domain controller with the credentials of a specific user, bind to a certain user account by SID, display the user description, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> (get-QADUser -identity
'S-1-5-21-1279736177-1630491018-182859109-1305').DirectoryEntry.description
C:\PS> disconnect-QADService
```

Example 3

Connect to any available domain controller with the credentials of the locally logged on user, search for users in a specific container by using an LDAP search filter, and display a list of the users found:

```
C:\PS> get-QADUser -SearchRoot 'company.com/UsersOU' -LdapFilter '(description=a*)'
```

Example 4

Connect to any available domain controller with the credentials of the locally logged on user, find all users in a specific container, and display a list of the users found:

```
C:\PS> get-QADUser -SearchRoot 'company.com/UsersOU'
```

Example 5

Connect to any available domain controller with the credentials of a specific user, search a certain container to find all users with empty title, set a title for each of those users, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -ConnectionAccount 'company\administrator'
-ConnectionPassword $pw
C:\PS> get-QADUser -SearchRoot 'company.com/UsersOU' -title '' | set-QADUser -title
'Contractor'
C:\PS> disconnect-QADService
```

Example 6

Connect to the local Administration Service with the credentials of the locally logged on user, find all users whose names begin with "A" and titles end in "Manager" and modify the description for each of those users; then, disconnect.

```
C:\PS> connect-QADService -service 'localhost' -proxy
C:\PS> get-QADUser -SearchRoot 'company.com/UsersOU' -Name 'A*' -SearchAttributes
@{name='B*';title='*manager'} | set-QADUser -description 'A manager whose name begins
with A'
C:\PS> disconnect-QADService
```

Note that the condition based on the *Name* parameter overrides the condition imposed on the "Name" attribute by the *SearchAttributes* parameter, so you could omit the *Name* parameter and type `name='A*'` instead of `name='B*'` in the value of the *SearchAttributes* parameter, or you could only remove the `name='B*'` entry from the value of the *SearchAttributes* parameter.

Example 7

List the names of the properties specific to a user object:

```
C:\PS> Get-QADUser -IncludeAllProperties -ReturnPropertyNamesOnly
```

Example 8

List the values of all properties of the user account:

```
C:\PS> Get-QADUser JSmith -IncludeAllProperties -SerializeValues | Format-List
```

Example 9

Export the user account to an XML file. Exported are the values of all properties:

```
C:\PS> Get-QADUser jsmith -IncludeAllProperties -SerializeValues | Export-Clixml user.xml
```

Example 10

Find user objects with a non-empty value of the 'homeDirectory' property, and display the values of the 'Name', 'HomeDirectory' and 'msDS-ReplAttributeMetaData' properties for each object found:

```
C:\PS> Get-QADUser -DontUseDefaultIncludedProperties -SearchAttributes @{'homeDirectory='*'} -IncludedProperties 'msDS-ReplAttributeMetaData',homeDirectory | Format-Table name, homeDirectory, 'msDS-ReplAttributeMetaData'
```

Example 11

Export the user object to a CSV file. Then, import that user object from that file:

```
C:\PS> Get-QADuser jsmith -SerializeValues | export-csv user.csv
C:\PS> import-csv user.csv | New-QADUser -ParentContainer 'MyDomain.lab.local/MyOU' -DeserializeValues -Name importedUser -LogonName importedUser -UserPassword 'P@ssw0rd'
```

Example 12

Count all user objects that exist in your Active Directory domain:

```
C:\PS> Get-QADUser -DontUseDefaultIncludedProperties -SizeLimit 0 | Measure-Object
```

Example 13

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, search a specific container to find all AD LDS user objects matching a certain LDAP search filter, and display the name and description of each user object found:

```
C:\PS> get-QADUser -Service 'server.domain.local:389' -SearchRoot '<DN of container>' -LdapFilter '(description=a*)' | Format-List name,description
```

Example 14

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and display the name and description of the AD LDS user object that is identified by DN:

```
C:\PS> get-QADUser '<DN of user object>' -Service 'server.domain.local:389' | Format-List name,description
```

Example 15

Retrieve user accounts from a particular container that are direct or indirect members of a particular group:

```
C:\PS> get-QADUser -SearchRoot '<DN of container>' -IndirectMemberOf  
'domainName\groupName'
```

Example 16

Retrieve all user accounts that were deleted from a particular container:

```
C:\PS> Get-QADUser -Tombstone -LastKnownParent '<DN of container>'
```

Example 17

Retrieve deleted user accounts with the name (RDN) of John Smith:

```
C:\PS> Get-QADUser -Tombstone -Name 'John Smith*'
```

Example 18

Retrieve all user accounts that were deleted from a particular container on the current date:

```
C:\PS> Get-QADUser -Tombstone -LastKnownParent '<DN of container>' -LastChangedOn  
(get-date)
```

Example 19

Retrieve all user accounts that were deleted on September 1, 2008:

```
C:\PS> Get-QADUser -Tombstone -LastChangedOn (get-date -year 2008 -month 9 -day 1)
```

Example 20

View progress of a command that retrieves all domain users:

```
C:\PS> Get-QADUser -ShowProgress -Activity 'Retrieving all domain users'  
-ProgressThreshold 0 | Out-Null
```

Example 21

Retrieve the user accounts that meet any of the default inactivity conditions (inactive accounts):

```
C:\PS> Get-QADUser -Inactive
```

Example 22

Retrieve the user accounts that do not meet any of the default inactivity conditions (active accounts):

```
C:\PS> Get-QADUser -Inactive:$false
```

Example 23

Retrieve the user accounts that remain in the expired state for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 30 or more days:

```
C:\PS> Get-QADUser -InactiveFor 30
```

Example 24

Retrieve the user accounts that meet the default expiry-related or logon-related inactivity condition, or have the password unchanged for 10 or more days:

```
C:\PS> Get-QADUser -Inactive -PasswordNotChangedFor 10
```

Example 25

Retrieve the user accounts that do not meet any of the default inactivity conditions, but remain in the expired state for 20 or more days:

```
C:\PS> Get-QADUser -Inactive:$false -ExpiredFor 20
```

Example 26

Retrieve the user accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 60 or more days:

```
C:\PS> Get-QADUser -InactiveFor 30 -NotLoggedOnFor 60
```

Example 27

For a given mailbox user, list the e-mail addresses that are currently assigned to the mailbox:

```
C:\PS> Get-QADUser DomainName\UserName | Select-Object -ExpandProperty ProxyAddresses
```

Output Object Properties

Properties and methods of the object returned by the Get-QADUser cmdlet can be used to examine and configure various properties of the respective user account. To view a list of all methods and properties that are available, use the following command:

```
get-QADUser 'domainname\username' | get-Member
```

For general information about using properties and methods of PowerShell objects, enter these commands:

```
get-help about_method  
get-help about_property
```

The following table summarizes some properties of a Get-QADUser output object. Using these properties you can view or modify properties on a user account you retrieve with the cmdlet. After setting new property values on the output object, you must call the CommitChanges() method on that object to save the property value changes in the user account (see examples at the end of this section).

PROPERTY	DESCRIPTION
AccountExpires Syntax: Nullable<DateTime>	The date and time after which the user cannot log on.
AccountIsDisabled Syntax: Boolean	A flag to indicate if the account is, or should be, disabled.
AccountIsLockedOut Syntax: Boolean	A flag that indicates if the account is locked because of failed logon attempts.

PROPERTY	DESCRIPTION
Department Syntax: String	The department within the company to which the user belongs.
Description Syntax: String	The text description of the user.
Email Syntax: String	The e-mail address of the user.
Fax Syntax: String	The fax number of the user.
FirstName Syntax: String	The first name of the user.
HomeDirectory Syntax: String	The home directory of the user.
HomeDrive Syntax: String	The drive letter to which the UNC path for the home directory is mapped.
LastLogon Syntax: Nullable<DateTime>	The date and time that the user last logged on using the domain controller from which the user account is retrieved by the cmdlet.
LastLogonTimestamp Syntax: Nullable<DateTime>	The date and time that the user last logged on to the domain.
LastName Syntax: String	The last name of the user.
LogonScript Syntax: String	The logon script path.
Manager Syntax: String	Identifies the account of the user's manager.
MemberOf Syntax: String[]	Array of strings, each of which identifies one of the groups that the user is a member of.
PasswordAge Syntax: Nullable<TimeSpan>	Time that has elapsed since the password was set or last changed.
PasswordExpires Syntax: Nullable<DateTime>	The date and time when the password expires.
PasswordLastSet Syntax: Nullable<DateTime>	The date and time when the password was set or last changed.
PasswordNeverExpires Syntax: Boolean	A flag indicating if the password is configured to never expire.
ProfilePath Syntax: String	The path to the user profile.

Quest One ActiveRoles Management Shell for Active Directory

PROPERTY	DESCRIPTION
TSAllowLogon Syntax: Boolean	A flag indicating if the user is allowed to log on to Terminal Services.
TSBrokenConnectionAction Syntax: Int32	The action to take when a Terminal Services session limit is reached: 1 if the session should be terminated; 0 if the session should be disconnected.
TSConnectClientDrives Syntax: Boolean	A flag indicating whether to reconnect to mapped client drives at logon to Terminal Services.
TSConnectPrinterDrives Syntax: Boolean	A flag indicating whether to reconnect to mapped client printers at logon to Terminal Services.
TSDefaultToMainPrinter Syntax: Boolean	A flag indicating whether to print automatically to the client's default printer when the user is logged on to the Terminal Server.
TSHomeDirectory Syntax: String	The Terminal Services home directory of the user.
TSHomeDrive Syntax: String	The drive letter to which the UNC path for the Terminal Services home directory is mapped.
TSInitialProgram Syntax: String	The path and file name of the application that starts automatically when the user logs on to Terminal Services.
TSMaxConnectionTime Syntax: TimeSpan	Maximum allowed duration of the Terminal Services session.
TSMaxDisconnectionTime Syntax: TimeSpan	Maximum amount of time that a disconnected Terminal Services session remains active on the Terminal Server.
TSMaxIdleTime Syntax: TimeSpan	Maximum amount of time that the Terminal Services session can remain idle.
TSProfilePath Syntax: String	The profile path to use when the user logs on to Terminal Services.
TSReconnectionAction Syntax: Int32	Specifies whether to allow reconnection to a disconnected Terminal Services session from any client computer: 1 if reconnection is allowed from the original client computer only; 0 if reconnection from any client computer is allowed.
TSRemoteControl Syntax: Int32	Specifies whether to allow remote observation or remote control of the user's Terminal Services session: <ul style="list-style-type: none">• 0 Remote control is disabled.• 1 Full control of the user's session, with the user's permission.• 2 Full control of the user's session; the user's permission is not required.• 3 View the session remotely, with the user's permission.• 4 View the session remotely; the user's permission is not required.
TSWorkDirectory Syntax: String	The working directory path to use when the user logs on to Terminal Services.

PROPERTY	DESCRIPTION
UserMustChangePassword Syntax: Boolean	A flag indicating if the user is required to change the password at next logon.

Examples

Example 1

Force a particular user to change the password at next logon:

```
C:\PS> $user = get-QADUser 'DomainName\UserName'
C:\PS> ($user).UserMustChangePassword = $true
C:\PS> ($user).CommitChanges()
```

Example 2

View the **TSAllowLogon** setting on a specific user account:

```
C:\PS> (get-QADUser 'DomainName\AccountName').TSAllowLogon
```

Example 3

Set the **TSMaxIdleTime** property on a specific user account to 15 minutes; then, view the setting:

```
C:\PS> $user = get-QADUser 'DomainName\UserName'
C:\PS> ($user).TSMaxIdleTime = [TimeSpan] ("0:15:0")
C:\PS> ($user).CommitChanges()
C:\PS> ($user).TSMaxIdleTime
```

Set-QADUser

Modify attributes of a user account in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Set-QADUser [-Identity] <IdentityParameter> [-AccountExpires <Nullable`1>]
[-PasswordNeverExpires] [-UserMustChangePassword] [-TsProfilePath <String>]
[-TsHomeDirectory <String>] [-TsHomeDrive <String>] [-TsWorkDirectory <String>]
[-TsInitialProgram <String>] [-TsMaxDisconnectionTime <TimeSpan>] [-TsMaxConnectionTime
<TimeSpan>] [-TsMaxIdleTime <TimeSpan>] [-TsAllowLogon] [-TsRemoteControl <Int32>]
[-TsReconnectionAction <Int32>] [-TsBrokenConnectionAction <Int32>]
[-TsConnectClientDrives] [-TsConnectPrinterDrives] [-TsDefaultToMainPrinter]
[-UserPassword <String>] [-City <String>] [-Company <String>] [-Department <String>]
[-Email <String>] [-Fax <String>] [-FirstName <String>] [-HomeDirectory
<String>] [-HomeDrive <String>] [-HomePhone <String>] [-Initials <String>] [-LastName
<String>] [-LogonScript <String>] [-Manager <IdentityParameter>] [-MobilePhone <String>]
[-Notes <String>] [-Office <String>] [-Pager <String>] [-PhoneNumber <String>]
[-PostalCode <String>] [-PostOfficeBox <String>] [-ProfilePath <String>]
[-SamAccountName <String>] [-StateOrProvince <String>] [-StreetAddress <String>] [-Title
<String>] [-UserPrincipalName <String>] [-WebPage <String>] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has a number of optional parameters for managing individual attributes in the directory, with each parameter name identifying a certain attribute that can be set to a value specified by using the respective parameter (see the list of parameters for this cmdlet).

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

AccountExpires

Set the account expiration date. Parameter value is a *DateTime* object that specifies the date you want. A null *DateTime* object configures the account to never expire.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

PasswordNeverExpires

Set this parameter to 'true' to configure the account so that its password never expires.

TsAllowLogon

Specify whether the user is allowed to log on to the Terminal Services. Parameter value can be 'true' or 'false':

- 'true' if logon is allowed
- 'false' if logon is not allowed

TsBrokenConnectionAction

Specify the action to take when a Terminal Services session limit is reached. Parameter value can be one of these integers:

- **1** (The client session should be terminated.)
- **0** (The client session should be disconnected.)

TsConnectClientDrives

Specify whether to reconnect to mapped client drives at logon to the Terminal Services. Parameter value can be 'true' or 'false':

- 'true' if reconnection is enabled
- 'false' if reconnection is disabled

TsConnectPrinterDrives

Specify whether to reconnect to mapped client printers at logon to the Terminal Services. Parameter value can be 'true' or 'false':

- 'true' if reconnection is enabled
- 'false' if reconnection is disabled

TsDefaultToMainPrinter

Specify whether to print automatically to the client's default printer. Parameter value can be 'true' or 'false':

- 'true' if printing to the client's default printer is enabled
- 'false' if printing to the client's default printer is disabled

TsHomeDirectory

Set the path to the Terminal Services home directory for the user. To set a home directory on the local computer, specify a local path; for example, C:\Path. To set a home directory in a network environment, set the TsHomeDrive parameter and specify a UNC path.

TsHomeDrive

Set the Terminal Services home drive for the user in a network environment. Parameter value is a string containing a drive letter followed by a colon, to which the UNC path for the Terminal Services home directory is mapped. To set a home directory in a network environment, set both this parameter and the TsHomeDirectory parameter.

TsInitialProgram

Set the path and file name of the application that starts automatically when the user logs on to the Terminal Services. To set an initial application to start when the user logs on, set both this parameter and the TsWorkDirectory parameter.

TsMaxConnectionTime

Set maximum duration of the Terminal Services session. After the specified time span has elapsed, the session can be disconnected or terminated. Parameter value is a TimeSpan object that specifies the duration you want.

TsMaxDisconnectionTime

Set maximum amount of time that a disconnected Terminal Services session remains active on the server. After the specified time span has elapsed, the session is terminated. Parameter value is a TimeSpan object that specifies the amount of time you want.

TsMaxIdleTime

Set maximum amount of time that the Terminal Services session can remain idle. After the specified time span has elapsed, the session can be disconnected or terminated. Parameter value is a TimeSpan object that specifies the amount of time you want.

TsProfilePath

Set a roaming or mandatory profile path to use when the user logs on to the Terminal Services. A valid parameter value is a string in the following network path format:
\\ServerName\ProfilesFolderName\UserName

TsReconnectionAction

Specify whether to allow reconnection to a disconnected Terminal Services session from any client computer. Parameter value can be one of these integers:

- **1** (Reconnection is allowed from the original client computer only.)
- **0** (Reconnection from any client computer is allowed.)

TsRemoteControl

Specify whether to allow remote observation or remote control of the user's Terminal Services session. Parameter value can be one of these integers:

- **0** (Remote control is disabled.)

- **1** (The user of remote control has full control of the user's session, with the user's permission.)
- **2** (The user of remote control has full control of the user's session; the user's permission is not required.)
- **3** (The user of remote control can view the session remotely, with the user's permission; the remote user cannot actively control the session.)
- **4** (The user of remote control can view the session remotely, but not actively control the session; the user's permission is not required.)

TsWorkDirectory

Set the Terminal Services working directory path for the user. To set an initial application to start when the user logs on to the Terminal Services, set both this parameter and the TsInitialProgram parameter.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

UserMustChangePassword

Set this parameter to 'true' to configure the user account so that the user is required to change the password upon the next logon.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Attribute-specific Parameters

This cmdlet takes a series of optional, attribute-specific parameters allowing you to make changes to user attributes in Active Directory. Thus, to modify the value of the "givenName", "sn", or "l" attribute, you can use the *FirstName*, *LastName*, or *City* parameter, respectively.

If a particular attribute is referred to by both the ObjectAttributes array and the attribute-specific parameter, the ObjectAttributes setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

The following table lists the attribute-specific parameters you can use with this cmdlet to manage user attributes. Each parameter is intended to manage a certain attribute, identified by the LDAP display name in the table. By using the cmdlet, you can set the value of the attribute to the parameter value specified. To clear the attribute, specify "" (empty string) as the parameter value.

TO MANAGE THIS ATTRIBUTE...	USE THIS SYNTAX
l	-City <String>
company	-Company <String>
description	-Description <String>
department	-Department <String>
displayName	-DisplayName <String>
facsimileTelephoneNumber	-Fax <String>

TO MANAGE THIS ATTRIBUTE...	USE THIS SYNTAX
givenName	-FirstName <String>
homeDirectory	-HomeDirectory <String>
homeDrive	-HomeDrive <String>
homePhone	-HomePhone <String>
initials	-Initials <String>
sn	-LastName <String>
mail	-Email <String>
manager	-Manager <IdentityParameter>
mobile	-MobilePhone <String>
info	-Notes <String>
pager	-Pager <String>
physicalDeliveryOfficeName	-Office <String>
profilePath	-ProfilePath <String>
Use this parameter to set user password	-UserPassword <String>
scriptPath	-LogonScript <String>
telephoneNumber	-Phone <String>
postalCode	-PostalCode <String>
postOfficeBox	-PostOfficeBox <String>
samAccountName	-SamAccountName <String>
st	-StateOrProvince <String>
streetAddress	-StreetAddress <String>
title	-Title <String>
userPrincipalName	-UserPrincipalName <String>
wwwHomePage	-WebPage <String>

Detailed Description

Use this cmdlet to change or remove values of attributes of a user account in Active Directory.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific user account by DN, and modify the user description:

```
C:\PS> set-QADUser 'CN=John Smith,OU=CompanyOU,DC=company,DC=com' -description 'Sales person'
```

Example 2

Connect to a specific domain controller with the credentials of a specific user, bind to a certain user account by SID, modify the user description, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> set-QADUser 'S-1-5-21-1279736177-1630491018-182859109-1305' -description
'Service account'
C:\PS> disconnect-QADService
```

Example 3

Connect to the local Administration Service with the credentials of a specific user, bind to a certain user account by Domain\Name, set or clear certain attributes, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> set-QADUser -identity 'company\jsmith' -ObjectAttributes @{l='New
York';description=''} -UserPassword 'P@ssword'
C:\PS> disconnect-QADService
```

Example 4

Assign two values to a multi-valued attribute such as "otherTelephone". This replaces the current values of the attribute with the specified values:

```
C:\PS> Set-QADUser 'mycompany.com/usersOU/User1' -objectAttributes
@{otherTelephone=@('555-34-67','555-34-68')}
```

Example 5

Add two values to a multi-valued attribute such as "otherTelephone". This appends the specified values to the existing values of the attribute. The existing values are not removed:

```
C:\PS> Set-QADUser 'mycompany.com/usersOU/User1' -objectAttributes
@{otherTelephone=@(Append=@('555-34-67','555-34-68'))}
```

Example 6

Delete the specified values from a multi-valued attribute such as "otherTelephone", leaving the other attribute values intact:

```
C:\PS> Set-QADUser 'mycompany.com/usersOU/User1' -objectAttributes
@{otherTelephone=@(Delete=@('555-34-67','555-34-68'))}
```

Example 7

Delete all values from a multi-valued attribute such as "otherTelephone" (clear the attribute on the user object):

```
C:\PS> Set-QADUser 'mycompany.com/usersOU/User1' -objectAttributes @{otherTelephone=''} 
```

Example 8

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS user object by DN, and modify the description of the AD LDS user object:

```
C:\PS> set-QADUser '<DN of user object>' -Service 'server.domain.local:389' -description 'My AD LDS user object'
```

Example 9

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
set-QADUser 'CN=John Smith,OU=CompanyOU,DC=company,DC=com' -description 'Sales person'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

Example 10

The following example shows how to check each operation within a batch of operations performed via ActiveRoles Management Shell to see if the operation requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example retrieves user accounts held in a given OU, sets their description, and then, for each user account, uses the `OperationStatus` property of the object representing the user account to see if the requested changes to the user account are applied or submitted for approval.

```
Get-QADUser -SearchRoot 'OU=Contractors,DC=company,DC=com' | Set-QADUser -Description 'Contractor' | foreach {$status = $_.OperationStatus
    If($status -eq 'Pending')
    {
        # Operation submitted for approval. Handle this case here.
    }
    elseif($status -eq 'Completed')
    {
        # No approval required. Operation completed.
    }
    else
    {
        # Operation not completed for some reason. You might check other OperationStatus
        # values to determine the status of the operation request.
    }
}
```

New-QADUser

Create a new user account in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
New-QADUser [-Name] <String> -ParentContainer <IdentityParameter> [-UserPassword <String>] [-City <String>] [-Company <String>] [-Department <String>] [-Email <String>] [-Fax <String>] [-FirstName <String>] [-HomeDirectory <String>] [-HomeDrive <String>] [-HomePhone <String>] [-Initials <String>] [-LastName <String>] [-LogonScript <String>] [-Manager <IdentityParameter>] [-MobilePhone <String>] [-Notes <String>] [-Office <String>] [-Pager <String>] [-PhoneNumber <String>] [-PostalCode <String>] [-PostOfficeBox <String>] [-ProfilePath <String>] [-SamAccountName <String>] [-StateOrProvince <String>] [-StreetAddress <String>] [-Title <String>] [-UserPrincipalName <String>] [-WebPage <String>] [-ObjectAttributes <ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>] [-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues] [-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has a number of optional parameters for managing individual attributes in the directory, with each parameter name identifying a certain attribute that can be set to a value specified by using the respective parameter (see the list of parameters for this cmdlet).

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

Name

Set the 'name' attribute to this parameter value on the new object created by this cmdlet in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ParentContainer

Specify the Distinguished Name of the container in which you want the new directory object to be created by this cmdlet.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Attribute-specific Parameters

This cmdlet takes a series of optional, attribute-specific parameters allowing you to set attributes in the newly created account. Thus, to set the value of the "givenName", "sn", or "l" attribute, you can use the *FirstName*, *LastName*, or *City* parameter, respectively.

If a particular attribute is referred to by both the ObjectAttributes array and the attribute-specific parameter, the ObjectAttributes setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

The following table lists the attribute-specific parameters you can use with this cmdlet to manage user attributes. Each parameter is intended to manage a certain attribute, identified by the LDAP display name in the table. By using the cmdlet, you can set the value of the attribute to the parameter value specified.

TO MANAGE THIS ATTRIBUTE...	USE THIS SYNTAX
l	-City <String>
company	-Company <String>
description	-Description <String>
department	-Department <String>
displayName	-DisplayName <String>
facsimileTelephoneNumber	-Fax <String>
givenName	-FirstName <String>
homeDirectory	-HomeDirectory <String>
homeDrive	-HomeDrive <String>
homePhone	-HomePhone <String>
initials	-Initials <String>
sn	-LastName <String>
mail	-Email <String>
manager	-Manager <IdentityParameter>
mobile	-MobilePhone <String>
info	-Notes <String>
physicalDeliveryOfficeName	-Office <String>
pager	-Pager <String>

TO MANAGE THIS ATTRIBUTE...	USE THIS SYNTAX
profilePath	-ProfilePath <String>
Use this parameter to set user password	-UserPassword <String>
scriptPath	-LogonScript <String>
telephoneNumber	-Phone <String>
postalCode	-PostalCode <String>
postOfficeBox	-PostOfficeBox <String>
samAccountName	-SamAccountName <String>
st	-StateOrProvince <String>
streetAddress	-StreetAddress <String>
title	-Title <String>
userPrincipalName	-UserPrincipalName <String>
wwwHomePage	-WebPage <String>

Detailed Description

Use this cmdlet to create a user account in Active Directory and, optionally, set attribute values in the newly created account.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, create a new user account, and set a password for the new account:

```
C:\PS> new-QADUser -name 'user1' -ParentContainer 'OU=companyOU,DC=company,DC=com'
-SamAccountName 'user1' -UserPassword 'P@ssword'
```

Example 2

Connect to the local Administration Service with the credentials of a specific user, create a new user account, set a password for the new account, and then disconnect (this example assumes that a value for the sAMAccountName attribute is to be generated by Quest One ActiveRoles, based on a provisioning policy):

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> new-QADUser -name 'user1' -ParentContainer 'OU=companyOU,DC=company,DC=com'
-UserPassword 'P@ssword'
C:\PS> disconnect-QADService
```

Example 3

Connect to the local Administration Service with the credentials of a specific user, import a CSV file, for each record in the file create a new user account with the name matching the value in the 'user name' column in the CSV file, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-qadService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> import-csv C:\temp\data.csv | %{new-qadUser -ParentContainer
'OU=companyOU,DC=company,DC=com' -name $_.'user name'}
C:\PS> disconnect-qadService
```

In this example, the **%** character preceding the script block is an alias for the **ForEach-Object** cmdlet. The sAMAccountName attribute is assumed to be set by Quest One ActiveRoles. For more information on this example, refer to the "Pipelining" section earlier in this document.

Example 4

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and create a new AD LDS user object in a certain container:

```
C:\PS> new-QADUser -Service 'server.domain.local:389' -Name 'user1' -ParentContainer
'<DN of container>' -UserPassword 'P@ssword'
```

Disable-QADUser

Disable a user account in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Disable-QADUser [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to disable a user account in Active Directory Domain Services or Active Directory Lightweight Directory Services.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user and disable the user account identified by Domain\Name:

```
C:\PS> disable-QADUser 'MyDomain\JSmith'
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and disable the AD LDS user account that is identified by DN:

```
C:\PS> disable-QADUser '<DN of user account>' -Service 'server.domain.local:389'
```

Enable-QADUser

Enable a user account in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Enable-QADUser [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to re-enable a disabled user account in Active Directory Domain Services or Active Directory Lightweight Directory Services.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user and enable the user account identified by Domain\Name:

```
C:\PS> enable-QADUser 'MyDomain\JSmith'
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and enable the AD LDS user account that is identified by DN:

```
C:\PS> enable-QADUser '<DN of user account>' -Service 'server.domain.local:389'
```

Unlock-QADUser

Unlock a user account in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Unlock-QADUser [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to unlock a user account that has been locked out due to a number of failed logon attempts. You can unlock user accounts in both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user and unlock the user account identified by Domain\Name:

```
C:\PS> unlock-QADUser 'MyDomain\JSmith'
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and unlock the AD LDS user account that is identified by DN:

```
C:\PS> unlock-QADUser '<DN of user account>' -Service 'server.domain.local:389'
```

Deprovision-QADUser

Request Quest One ActiveRoles to deprovision a user account. This cmdlet requires a connection to be established to the ActiveRoles Administration Service by supplying the *Proxy* parameter.

Syntax

```
Deprovision-QADUser [-Identity] <IdentityParameter> [-ReportFile <String>] [-Xml]
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the *Proxy* parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

ReportFile

Supply this parameter if you want to save a report on the deprovisioning results to a file in HTML or XML format. The parameter value must be a valid path to a file, including the file name. The cmdlet creates the file if necessary. Omit this parameter if you do not want to save the report in a file. Quest One ActiveRoles preserves the report data regardless of this parameter, so you always have the option to examine the deprovisioning results using the Quest One ActiveRoles console.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

XML

Supply this parameter if you want to save the report on the deprovisioning results in XML format. Omit this parameter if you want to save the report in HTML format.

Detailed Description

Use this cmdlet to deprovision a user account via Quest One ActiveRoles. Quest One ActiveRoles provides the ability to deprovision rather than delete or only disable user accounts. Deprovision refers to a set of actions being performed in order to revoke user access to resources. The deprovision operation on user objects triggers deprovision policies. Quest One ActiveRoles comes with a default policy to automate some commonly-used deprovisioning tasks, and allows the deprovision policies to be adjusted as needed.

This cmdlet originates a request to deprovision the user accounts specified. When processing the request, Quest One ActiveRoles performs all operations prescribed by the deprovision policies.

Examples

Example 1

Connect to any available Quest One ActiveRoles Administration Service with the credentials of the locally logged on user and deprovision the user account identified by Domain\Name:

```
C:\PS> deprovision-QADUser 'MyDomain\JSmith' -Proxy
```

Example 2

Connect to a specific Administration Service with the credentials of the locally logged on user, retrieve a user object using the get-QADUser cmdlet, and pipe the user object into the Deprovision-QADUser cmdlet to deprovision the user account represented by that object:

```
C:\PS> connect-QADService -Service 'myserver.mydomain.lab' -Proxy  
C:\PS> get-QADUser 'MyDomain\JSmith' | deprovision-QADUser
```

Example 3

Connect to a specific Administration Service with the credentials of the locally logged on user, retrieve a user object using the get-QADUser cmdlet, and pipe the user object into the Deprovision-QADUser cmdlet to deprovision the user account represented by that object, with a report on the deprovisioning results being saved in a specific file in HTML format:

```
C:\PS> connect-QADService -Service 'myserver.mydomain.lab' -Proxy
C:\PS> get-QADUser 'MyDomain\JSmith' | deprovision-QADUser -ReportFile 'C:\JSmith.html'
```

Example 4

Connect to a specific Administration Service with the credentials of the locally logged on user, and deprovision all user accounts found in a specific container, with a report on the deprovisioning results for each user account being saved in a separate file:

```
C:\PS> connect-QADService -Service 'myserver.mydomain.lab' -Proxy
C:\PS> get-QADUser -SearchRoot 'mydomain.lab/retired' | deprovision-QADUser -ReportFile
{'C:\DeprovisionReports\' + $_.SamAccountName + '.html'}
```

Get-QADGroup

Retrieve all groups in a domain or container that match the specified conditions. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Get-QADGroup [-Identity <IdentityParameter>] [-SamAccountName <String[]>] [-GroupType <GroupType>] [-GroupScope <GroupScope>] [-Dynamic] [-Empty] [-Keywords <String[]>] [-ManagedBy <IdentityParameter[]>] [-SecondaryOwner <IdentityParameter[]>] [-RequireManagerApproval] [-RequireSecondaryOwnerApproval] [-Published] [-ProxyAddress <String[]>] [-PrimaryProxyAddress <String[]>] [-SecondaryProxyAddress <String[]>] [-ContainsMember <IdentityParameter[]>] [-ContainsIndirectMember <IdentityParameter[]>] [-NotContainsMember <IdentityParameter[]>] [-NotContainsIndirectMember <IdentityParameter[]>] [-MemberOf <IdentityParameter[]>] [-IndirectMemberOf <IdentityParameter[]>] [-NotMemberOf <IdentityParameter[]>] [-NotIndirectMemberOf <IdentityParameter[]>] [-Tombstone] [-Recycled] [-LastKnownParent <IdentityParameter>] [-SecurityMask <SecurityMasks>] [-SearchRoot <IdentityParameter[]>] [-SearchScope <SearchScope>] [-AttributeScopeQuery <String>] [-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode <WildcardMode>] [-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>] [-IncludeAllProperties] [-DontConvertValuesToFriendlyRepresentation] [-SerializeValues] [-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties] [-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity <String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

ContainsIndirectMember

Retrieve groups to which the object or objects specified by this parameter belong, whether directly or because of group nesting. The cmdlet returns a group if the object has direct or indirect membership in that group.

ContainsMember

Retrieve groups that hold the object or objects specified by this parameter. The cmdlet returns a group if the object is a direct member of that group.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Empty

Set this parameter to 'true' for the cmdlet to retrieve only those groups that have no members (empty groups).

Note: A group is considered empty if it has the "member" attribute not set. So, the Empty parameter can retrieve a group that has only those members for which the group is set as the primary group. An example is the Domain Users group, which normally is the primary group for any user account while having the "member" attribute not set.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

GroupScope

Specify the group scope of groups to find. Acceptable values are:

- 'Global'
- 'Universal'
- 'DomainLocal'

GroupType

Specify the group type of groups to find. Acceptable values are:

- 'Security'
- 'Distribution'

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

IndirectMemberOf

Retrieve objects that belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has direct or indirect membership in the group specified by this parameter value.

Keywords

Search by the 'edsvaKeywords' attribute. This parameter has an effect only in conjunction with the Proxy connection parameter because keywords are stored and managed by Quest One ActiveRoles. Parameter value is a string array that specifies one or more keywords to search for.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LastKnownParent

When searching for a deleted object by using the Tombstone parameter, specify the DN of the container the object was in before it became a tombstone. This allows you to find objects that were deleted from a particular container.

Note that the lastKnownParent attribute is only set if the object was deleted on a domain controller running Windows Server 2003 or later version of Microsoft Windows Server. Therefore, it is possible that the lastKnownParent attribute value is inaccurate.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

ManagedBy

Search by the 'managedBy' attribute. Supply the DN, SID, GUID, UPN or Domain\Name of the user, group or contact that is specified in the 'managedBy' attribute of the object to search for.

MemberOf

Retrieve objects that are direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object has direct membership in the group specified by this parameter value.

Name

Search by the 'name' attribute.

NotContainsIndirectMember

Retrieve groups to which the object or objects specified by this parameter do not belong directly or because of group nesting. The cmdlet returns a group if the object has neither direct nor indirect membership in that group.

NotContainsMember

Retrieve groups that do not hold the object or objects specified by this parameter. The cmdlet returns a group if the object is not a direct member of that group.

NotIndirectMemberOf

Retrieve objects that do not belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has neither direct nor indirect membership in the group specified by this parameter value.

NotMemberOf

Retrieve objects that are not direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object does not have direct membership in the group specified by this parameter value.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

PrimaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a primary (reply-to) e-mail address.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients that have any of the specified e-mail addresses.

Published

Search by the 'edsvaPublished' attribute. When the attribute is set to \$true, the group is published, which enables users to submit requests to join or leave that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the group publication status is stored and managed by Quest One ActiveRoles.

Recycled

This parameter has an effect only if all of the following conditions are true:

- A domain is supplied as the SearchRoot parameter value.
- Active Directory Recycle Bin is enabled in that domain.

You can use this parameter in conjunction with the Tombstone parameter for the search results to include both the deleted and recycled objects that meet the search conditions. Without this parameter, the cmdlet returns only deleted objects.

RequireManagerApproval

Search by the 'edsvaApprovalByPrimaryOwnerRequired' attribute. The attribute determines whether changes to the members list of a group require approval by the primary owner (manager) of that group. Parameter value (\$true or \$false) matches the 'edsvaApprovalByPrimaryOwnerRequired' attribute value to search for. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

RequireSecondaryOwnerApproval

Search by the 'edsvaApprovalBySecondaryOwnerRequired' attribute. The attribute determines whether changes to the members list of a group require approval by a secondary owner of that group. Parameter value (\$true or \$false) matches the 'edsvaApprovalBySecondaryOwnerRequired' attribute value to search for. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SamAccountName

Search by the 'sAMAccountName' attribute (pre-Windows 2000 name).

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SecondaryOwner

Search by the 'edsvaSecondaryOwners' attribute. Supply the DN, SID, GUID, UPN or Domain\Name of the user or group that is specified in the 'edsvaSecondaryOwners' attribute of the object to search for. This parameter has an effect only in conjunction with the Proxy connection parameter because the secondary owner settings are stored and managed by Quest One ActiveRoles.

SecondaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a non-primary e-mail address.

SecurityMask

Specify which elements of the object's security descriptor to retrieve. Valid parameter values are:

- 'None' - do not retrieve any security data
- 'Owner' - retrieve the owner data
- 'Group' - retrieve the primary group data
- 'Dacl' - retrieve the discretionary access-control list data
- 'Sacl' - retrieve the system access-control list data

You can supply a combination of these values, separating them by commas. For example, you can supply the parameter value of 'Dacl,Sacl' in order to retrieve both the discretionary and system access-control list data.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Tombstone

Search for deleted objects of the respective object class. The search output is normally intended to be passed (piped in) to the Restore-QADDeletedObject cmdlet for restoring deleted objects.

In a domain with Active Directory Recycle Bin (a feature of Windows Server 2008 R2) this parameter retrieves deleted objects (rather than tombstones, which in that case are referred to as recycled objects). Recycle Bin preserves all attributes on the deleted objects, so you can use a search filter based on any attributes.

In a domain without Active Directory Recycle Bin, deleting an object converts that object to a tombstone. A search using this parameter returns tombstone objects that meet the filtering criteria supplied. Upon deletion of an object only a small number of the object's attributes are saved in the tombstone, with most of the attributes being lost. To search for deleted objects, your search filter should be based on the attributes that are preserved in tombstones.

When the Tombstone parameter is supplied, the search results include the deleted objects or tombstones that match the specified search filter. However, a search filter that matches a live object may not work as expected after the object is deleted. This is because not all attributes are retained in the tombstone. For example, a filter such as (&(objectClass=user)(objectCategory=person)) would not match any tombstone objects since the objectCategory attribute is removed upon object deletion. Conversely, the objectClass attribute is retained on tombstone objects, so a filter of (objectClass=user) would match deleted user objects.

The name of a tombstone object begins with the name of the deleted object, so a search using the Tombstone parameter can be refined by adding a filter based on object name. For example, to search for deleted objects with a name that begins with "John", you can use a filter such as (cn=John*).

It is also possible to find a specific deleted object. If you know the name of the object and the Distinguished Name (DN) of the container the object was in before it was deleted, then you can pass the container's DN to the `LastKnownParent` parameter and apply a filter of `(cn=<name of the object>*)` in order to have the cmdlet retrieve that specific object. However, if an object is deleted, a new object with the same DN is created, and then deleted as well, the above search would return more than one object. The returned objects are distinguished by the GUIDs of the deleted objects, with the name of each ending in the GUID of the respective deleted object.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the `Get-QADPSSnapinSettings` cmdlet, respectively. Normally, this parameter is used in conjunction with `IncludeAllProperties` to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that `WildcardMode` is set to 'LDAP'. You can view or modify this default setting by using the `Get-QADPSSnapinSettings` cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to search an Active Directory domain or container for groups that meet certain criteria, or to bind to a certain group by DN, SID, GUID, or Domain\Name. You can search by group attributes or specify your search criteria by using an LDAP search filter.

The output of the cmdlet is a collection of objects, with each object representing one of the groups found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the groups returned by this cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific group by Domain\Name, and display the description of the group:

```
C:\PS> (get-QADGroup 'MyDom\Administrators').DirectoryEntry.description
```

Example 2

Connect to a specific domain controller with the credentials of a specific user, bind to a certain group by SID, display the description of the group, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> (get-QADGroup -identity
'S-1-5-21-1279736177-1630491018-182859109-1305').DirectoryEntry.description
C:\PS> disconnect-QADService
```

Example 3

Connect to any available domain controller with the credentials of the locally logged on user, search for groups in a specific container by using an LDAP search filter, and display a list of the groups found:

```
C:\PS> get-QADGroup -SearchRoot 'company.com/GroupsOU' -LdapFilter '(description=a*)'
```

Example 4

Connect to any available domain controller with the credentials of the locally logged on user, find all distribution groups in a specific container, and display a list of the groups found:

```
C:\PS> get-QADGroup -SearchRoot 'company.com/GroupsOU' -GroupType 'Distribution'
```

Example 5

Search a certain container to find all groups with the empty Notes field, and set a note for each of those groups:

```
C:\PS> get-QADGroup -SearchRoot 'company.com/GroupsOU' -SearchAttributes @{info=''} |
set-QADObject -ObjectAttributes @{info='A note'}
```

Example 6

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, find all AD LDS groups in a specific container, and display a list of the groups found:

```
C:\PS> get-QADGroup -Service 'server.domain.local:389' -SearchRoot '<DN of container>'
```

Example 7

Retrieve groups from a particular container that have a particular user account as a direct or indirect member:

```
C:\PS> get-QADGroup -SearchRoot '<DN of container>' -ContainsIndirectMember
'domainName\userName'
```

Example 8

Set the user preference for the progress bar appearance policy, to cause a progress bar to appear by default when a command takes longer than 2 seconds to finish; then, view progress of a command that retrieves domain groups, along with progress of a command that builds a list of members for each group:

```
C:\PS> Set-QADProgressPolicy -ShowProgress $true -ProgressThreshold 2 | Out-Null
C:\PS> Get-QADGroup -Activity 'Retrieving groups' | Get-QADGroupMember -Activity
'Building list of group members' | Out-Null
```

Set-QADGroup

Modify attributes of a group in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Set-QADGroup [-Identity] <IdentityParameter> [-Keywords <UpdateStringParameter[]>]
[-Member <UpdateIdentityParameter[]>] [-SecondaryOwner <UpdateIdentityParameter[]>]
[-SamAccountName <String>] [-ManagedBy <IdentityParameter>] [-Notes <String>] [-Email
<String>] [-GroupType <GroupType>] [-GroupScope <GroupScope>] [-RequireManagerApproval]
[-RequireSecondaryOwnerApproval] [-ManagerCanUpdateMembershipList]
[-SecondaryOwnersCanUpdateMembershipList] [-Published] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

Email

Set the 'mail' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

GroupScope

Set the group scope to this parameter value. Acceptable values are:

- 'Global'
- 'Universal'
- 'DomainLocal'

GroupType

Set the group type to this parameter value. Acceptable values are:

- 'Security'
- 'Distribution'

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with `UseDefaultExcludedProperties`, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both `ExcludedProperties` and `IncludedProperties`, the cmdlet does not set the value of that attribute in the directory.

Keywords

Use this parameter to supply keywords for the group. Keywords are words or phrases that could help users find the group in Quest One ActiveRoles client applications such as the Web Interface. Parameter value can be a string array or an associative array that specifies one or more keywords to assign to the group or remove from the group. Some examples of possible parameter values are:

```
-Keywords 'keyword 1','keyword 2'
```

Replace all the existing keywords with the keywords specified.

```
-Keywords @{append=@('keyword 1','keyword 2')}
```

Add the specified keywords without removing the existing keywords.

```
-Keywords @{delete=@('keyword 1','keyword 2')}
```

Remove the specified keywords, leaving the other keywords intact.

```
-Keywords $null
```

Remove all the existing keywords.

This parameter has an effect only in conjunction with the `Proxy connection` parameter because keywords are stored and managed by Quest One ActiveRoles.

ManagedBy

Specify the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as the 'managedBy' attribute value on the object in the directory.

ManagerCanUpdateMembershipList

Use this parameter to specify whether the manager (primary owner) of the group is allowed to add or remove members from that group. Supply the parameter value of `$true` if you want to allow the manager to add or remove group members; supply the parameter value of `$false` to configure the group so that its manager is not allowed to add or remove group members. This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the `Proxy connection` parameter.

Member

Use this parameter to add or remove members from the group. Parameter value can be a string array or an associative array that specifies the identities, such as DN, SID, GUID, UPN or Domain\Name, of one or more objects to add or remove from the group. Some examples of possible parameter values are:

```
-Member 'domain\administrator','domain\user'
```

Replace the existing members with the objects specified.

Quest One ActiveRoles Management Shell for Active Directory

```
-Member @{append=@('domain\administrator','domain\user')}
```

Add the specified objects to the group.

```
-Member @{delete=@('domain\administrator','domain\user')}
```

Remove the specified objects from the group.

```
-Member $null
```

Remove all members from the group.

Note that this parameter only makes changes to the 'member' attribute, and has no effect on the group members that have the group set as the primary group.

Notes

Set the 'info' attribute to this parameter value.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

Published

Set the 'edsvaPublished' attribute to this parameter value. When this attribute is set to \$true, the group is published, which enables self-service users to submit requests to join or leave that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the group publication status is stored and managed by Quest One ActiveRoles.

RequireManagerApproval

Set the 'edsvaApprovalByPrimaryOwnerRequired' attribute to this parameter value (\$true or \$false). The attribute determines whether changes to the members list of a group require approval by the primary owner (manager) of that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

RequireSecondaryOwnerApproval

Set by the 'edsvaApprovalBySecondaryOwnerRequired' attribute to this parameter value (\$true or \$false). The attribute determines whether changes to the members list of a group require approval by a secondary owner of that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

SamAccountName

Set the 'sAMAccountName' attribute (pre-Windows 2000 name) to this parameter value.

SecondaryOwner

Use this parameter to add or remove secondary owners. Parameter value can be a string array or an associative array that specifies the identifiers, such as DN, SID, GUID, UPN or Domain\Name, of one or more users or groups to add or remove from the secondary owner role. Some examples of possible parameter values are:

```
-SecondaryOwner 'domain\administrator','domain\user'
```

Replace the existing identities in the secondary owners list with the identities specified.

```
-SecondaryOwner @{append=@('domain\administrator','domain\user')}
```

Add the specified identities to the secondary owners list, without removing the existing owners.

```
-SecondaryOwner @{delete=@('domain\administrator','domain\user')}
```

Remove the specified identities from the secondary owners list, leaving the other owners intact.

```
-SecondaryOwner $null
```

Clear the secondary owners list, so that no secondary owners are specified.

This parameter has an effect only in conjunction with the Proxy connection parameter because the secondary owner settings are stored and managed by Quest One ActiveRoles.

SecondaryOwnersCanUpdateMembershipList

Use this parameter to specify whether secondary owners of the group are allowed to add or remove members from that group. Supply the parameter value of `$true` if you want to allow the secondary owners to add or remove group members; supply the parameter value of `$false` to configure the group so that its secondary owners are not allowed to add or remove group members. This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the Proxy connection parameter.

UseDefaultExcludedProperties

When set to `'true'`, this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the `Get-QADPSSnapinSettings` cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to change or remove values of attributes of a group in Active Directory.

The cmdlet takes a series of optional, attribute-specific parameters allowing you to make changes to attributes in Active Directory. Thus, to modify the value of the `'description'` or `'displayName'` attribute, you can use the `-Description` or `-DisplayName` parameter, respectively. If a particular attribute is referred to by both the `ObjectAttributes` array and an attribute-specific parameter, the `ObjectAttributes` setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific group by Domain\Name, and modify the description of the group:

```
C:\PS> set-QADGroup 'MyDomain\AMS Managers' -description 'Amsterdam Managers'
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS group object by DN, and modify the description of the AD LDS group object:

```
C:\PS> set-QADGroup '<DN of group object>' -Service 'server.domain.local:389'
-description 'My AD LDS group object'
```

Example 3

Pipe the get-QADGroup output into the set-QADGroup cmdlet to change the pre-Windows 2000 group name (add the "New" suffix to the name of the group returned by get-QADGroup):

```
C:\PS> Get-QADGroup MyTestGroup | %{Set-QADGroup $_ -SamAccountName ($_.SamAccountName
+ "New") }
```

Example 4

Bind to the group by distinguished name and set the group name (pre-Windows 2000):

```
C:\PS> set-QADGroup 'CN=TestGroup,OU=Groups,DC=domain,DC=company,DC=com'
-samaccountname 'My Test Group'
```

Example 5

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using Connect-QADService). The example submits a request to perform a certain operation, uses Get-QARSLastOperation to retrieve the object representing the operation request, and then checks the OperationStatus property of that object to see if the requested changes are applied or submitted for approval.

```
set-QADGroup 'MyDomain\AMS Managers' -description 'Amsterdam Managers'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

New-QADGroup

Create a new group in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
New-QADGroup [-Name] <String> -ParentContainer <IdentityParameter> [-Member
<IdentityParameter[]>] [-Keywords <String[]>] [-SecondaryOwner <IdentityParameter[]>]
[-SamAccountName <String>] [-ManagedBy <IdentityParameter>] [-Notes <String>] [-Email
<String>] [-GroupType <GroupType>] [-GroupScope <GroupScope>] [-RequireManagerApproval]
[-RequireSecondaryOwnerApproval] [-ManagerCanUpdateMembershipList]
[-SecondaryOwnersCanUpdateMembershipList] [-Published] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has a number of optional parameters for managing individual attributes in the directory, with each parameter name identifying a certain attribute that can be set to a value specified by using the respective parameter (see the list of parameters for this cmdlet).

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

GroupScope

Set the group scope to this parameter value. Acceptable values are:

- 'Global'
- 'Universal'
- 'DomainLocal'

GroupType

Set the group type to this parameter value. Acceptable values are:

- 'Security'
- 'Distribution'

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

Keywords

Use this parameter to supply keywords for the group. Keywords are words or phrases that could help users identify the group in Quest One ActiveRoles client applications such as the Web Interface. Parameter value can be a string array that specifies one or more keywords to assign to the group. This parameter has an effect only in conjunction with the Proxy connection parameter because keywords are stored and managed by Quest One ActiveRoles.

ManagedBy

Specify the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as the 'managedBy' attribute value on the object in the directory.

ManagerCanUpdateMembershipList

Use this parameter to specify whether the manager (primary owner) of the group is allowed to add or remove members from that group. Supply the parameter value of \$true if you want to allow the manager to add or remove group members; supply the parameter value of \$false to configure the group so that its manager is not allowed to add or remove group members. This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the Proxy connection parameter.

Member

Set the 'member' attribute. Supply the DN, SID, GUID, UPN or Domain\Name of the object to be added to the group. You can supply a string array of object identifiers to add several objects to the group.

Name

Set the 'name' attribute to this parameter value on the new object created by this cmdlet in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ParentContainer

Specify the Distinguished Name of the container in which you want the new directory object to be created by this cmdlet.

Published

Supply this parameter if you want to publish the group. Publishing a group enables users to submit requests to join or leave that group.

RequireManagerApproval

Set the 'edsvaApprovalByPrimaryOwnerRequired' attribute to this parameter value (\$true or \$false). The attribute determines whether changes to the members list of a group require approval by the primary owner (manager) of that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

RequireSecondaryOwnerApproval

Set by the 'edsvaApprovalBySecondaryOwnerRequired' attribute to this parameter value (\$true or \$false). The attribute determines whether changes to the members list of a group require approval by a secondary owner of that group. This parameter has an effect only in conjunction with the Proxy connection parameter because the approval settings are stored and managed by Quest One ActiveRoles.

SamAccountName

Set the 'sAMAccountName' attribute (pre-Windows 2000 name) to this parameter value.

SecondaryOwner

Set the 'edsvaSecondaryOwners' attribute. Supply the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as a secondary owner. You can supply a string array of object identifiers to specify several secondary owners. This parameter has an effect only in conjunction with the Proxy connection parameter because the secondary owner settings are stored and managed by Quest One ActiveRoles.

SecondaryOwnersCanUpdateMembershipList

Use this parameter to specify whether secondary owners of the group are allowed to add or remove members from that group. Supply the parameter value of \$true if you want to allow the secondary owners to add or remove group members; supply the parameter value of \$false to configure the group so that its secondary owners are not allowed to add or remove group members. This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the Proxy connection parameter.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Attribute-specific Parameters

This cmdlet also takes a series of optional, attribute-specific parameters allowing you to set attributes in the newly created group. Thus, to set the value of the "description", "displayName", or "member" attribute, you can use the *Description*, *DisplayName*, or *Member* parameter, respectively.

If a particular attribute is referred to by both the ObjectAttributes array and the attribute-specific parameter, the ObjectAttributes setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

The following table lists the attribute-specific parameters you can use with this cmdlet to manage attributes of group objects. Each parameter is intended to manage a certain attribute, identified by the LDAP display name in the table. By using the cmdlet, you can set the value of the attribute to the parameter value specified.

TO MANAGE THIS ATTRIBUTE...	USE THIS SYNTAX
description	-Description <String>

displayName	-DisplayName <String>
info	-Notes <String>
mail	-Email <String>

Detailed Description

Use this cmdlet to create a group in Active Directory and, optionally, add members to and set other attribute values in the newly created group.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, and create a new universal distribution group:

```
C:\PS> new-qadGroup -ParentContainer 'OU=companyOU,DC=company,DC=com' -name 'group1'
-samAccountName 'group1' -groupType 'Distribution' -groupscope 'Universal'
```

Example 2

Connect to the local Administration Service with the credentials of a specific user, create a new universal distribution group, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-qadService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> new-qadGroup -ParentContainer 'OU=companyOU,DC=company,DC=com' -name 'group1'
-samAccountName 'group1' -groupType 'Distribution' -groupscope 'Universal'
C:\PS> disconnect-qadService
```

Example 3

Connect to the local Administration Service with the credentials of a specific user, import a CSV file, for each record in the file create a new global security group with the name matching the value in the 'group name' column in the CSV file, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-qadService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> import-csv C:\temp\data.csv | %{new-qadGroup -ParentContainer
'OU=companyOU,DC=company,DC=com' -name $_.'group name' -samAccountName $_.'group name'}
C:\PS> disconnect-qadService
```

Example 4

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and create a new AD LDS group in a certain container:

```
C:\PS> new-QADGroup -Service 'server.domain.local:389' -Name 'group1' -ParentContainer
'<DN of container>'
```

Get-QADGroupMember

Retrieve the members of a group in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Get-QADGroupMember [-Identity] <IdentityParameter> [-KeepForeignSecurityPrincipals]
[-Indirect] [-Type <String>] [-Enabled] [-Disabled] [-PageSize <Int32>] [-SizeLimit
<Int32>] [-LdapFilter <String>] [-WildcardMode <WildcardMode>] [-SearchAttributes
<Object>] [-Description <String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr
<String>] [-Control<Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>]
[-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>]
[-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (*displayName*)
- Given-Name (*givenName*)
- Legacy-Exchange-DN (*legacyExchangeDN*)
- ms-DS-Additional-Sam-Account-Name (*msDS-AdditionalSamAccountName*)
- Physical-Delivery-Office-Name (*physicalDeliveryOfficeName*)

- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

Disabled

Supply this parameter for the cmdlet to retrieve only those accounts that are disabled.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values “as is,” without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Enabled

Supply this parameter for the cmdlet to retrieve only those accounts that are enabled (not disabled).

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

Indirect

Supply this parameter for the cmdlet to retrieve objects that belong to the group because of group nesting, in addition to objects that are direct members of the group.

If this parameter is omitted, the cmdlet retrieves only those objects that are direct members of the group. If this parameter is supplied, the cmdlet takes the immediate list of members of the group and then also recursively expands each group in this list to determine its group memberships to arrive at a complete closed set of the members.

KeepForeignSecurityPrincipals

Supply this parameter if you do not want the cmdlet to resolve the retrieved foreign security principal objects by looking up the corresponding external security principals. Thus, when retrieving members of a group, the cmdlet may encounter a member represented by a foreign security principal object - an object in the forest of the group that represents a security principal (such as a user, computer, or group) that exists in a trusted domain located in a different forest. By default, the cmdlet attempts to look up that security principal based on the information held in the foreign security principal object, and, in case of a successful lookup, outputs the security principal data instead of the foreign security principal object data. The KeepForeignSecurityPrincipals parameter changes this behavior so that the cmdlet outputs the foreign security principal object data "as is," without attempting to look up the corresponding (external) security principals.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria on the group members. Note that the search filter string is case-sensitive.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Type

Specify the type of directory objects to find. The cmdlet searches for objects that have one of the 'objectClass' attribute values set to the Type parameter value.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to retrieve the directory objects that are members of a certain group in Active Directory.

The output of the cmdlet is a collection of objects, with each object representing one of the directory objects found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the directory objects returned by this cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific group by Domain\Name, and display a list of members of the group:

```
C:\PS> get-QADGroupMember 'MyDomain\Administrators'
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS group by DN, and display a list of members of the group:

```
C:\PS> get-QADGroupMember '<DN of group>' -Service 'server.domain.local:389'
```

Example 3

Retrieve users that belong to a particular group (directly or because of group nesting):

```
C:\PS> Get-QADGroupMember 'domainName\groupName' -Type 'user' -Indirect
```

Example 4

Set the user preference for the progress bar appearance policy, to cause a progress bar to appear by default when a command takes longer than 2 seconds to finish; then, view progress of a command that retrieves domain groups, along with progress of a command that builds a list of members for each group:

```
C:\PS> Set-QADProgressPolicy -ShowProgress $true -ProgressThreshold 2 | Out-Null
C:\PS> Get-QADGroup -Activity 'Retrieving groups' | Get-QADGroupMember -Activity
'Building list of group members' | Out-Null
```

Add-QADGroupMember

Add one or more objects to a group in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Add-QADGroupMember [-Identity] <IdentityParameter> [-Member] <IdentityParameter[]>
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

Member

Specify a list of objects to be added to the group. Each list entry is the DN, SID, GUID, UPN or Domain\Name of a directory object. Separate the list entries by commas.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add objects to a group in Active Directory. You can specify a list of objects to add, separating the list entries by commas.

Examples

Example 1

Connect to the local Administration Service with the credentials of a specific user, add two objects (the first one specified by Domain\Name, the second one specified by SID) to the group, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> add-QADGroupMember -identity 'CN=group1,OU=companyOU,DC=company,DC=com' -member
'company\jsmith','S-1-5-21-1279736177-1630491018-182859109-1215'
C:\PS> disconnect-QADService
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS group by DN, and add the object with a certain DN to the group:

```
C:\PS> add-QADGroupMember '<DN of group>' -Service 'server.domain.local:389' -Member
'<DN of object>'
```

Example 3

Add a user from an external trusted domain to a group (the user and the group are located in different forests):

```
C:\PS> $user = Get-QADUser domainName\userName -Service foreign.domain.com
C:\PS> Add-QADGroupMember domainName\groupName $user
```

Example 4

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
add-QADGroupMember 'domainName\groupName' -member 'DomainName\UserName'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

Remove-QADGroupMember

Remove one or more members from a group in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Remove-QADGroupMember [-Identity] <IdentityParameter> [-Member] <IdentityParameter[]>
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

Member

Specify a list of objects to be removed from the group. Each list entry is the DN, SID, GUID, UPN or Domain\Name of a directory object. Separate the list entries by commas.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove members from a group in Active Directory. You can specify a list of objects to remove, separating the list items by commas.

Examples

Example 1

Connect to the local Administration Service with the credentials of a specific user, remove two members (the first one specified by Domain\AccountName, the second one specified by SID) from the group, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> remove-QADGroupMember 'CN=group1,OU=companyOU,DC=company,DC=com' -member
'company\jsmith','S-1-5-21-1279736177-1630491018-182859109-1215'
C:\PS> disconnect-QADService
```

Example 2

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS group by DN, and remove the object with a certain DN from the group:

```
C:\PS> remove-QADGroupMember '<DN of group>' -Service 'server.domain.local:389' -Member
'<DN of object>'
```

Example 3

Remove a user that resides in an external trusted domain, from a group (the user and the group are located in different forests):

```
C:\PS> $user = Get-QADUser domainName\userName -Service foreign.domain.com
C:\PS> Remove-QADGroupMember domainName\groupName $user
```

Get-QADMemberOf

Retrieve group memberships of a particular object in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Get-QADMemberOf [-Identity] <IdentityParameter> [-Indirect] [-PageSize <Int32>]
[-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode <WildcardMode>]
[-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName <String[]>] [-Name
<String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn <DateTime>]
[-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>]
[-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (*displayName*)
- Given-Name (*givenName*)
- Legacy-Exchange-DN (*legacyExchangeDN*)
- ms-DS-Additional-Sam-Account-Name (*msDS-AdditionalSamAccountName*)
- Physical-Delivery-Office-Name (*physicalDeliveryOfficeName*)
- Proxy-Addresses (*proxyAddresses*)

- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values “as is,” without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the DN, SID, GUID, or DomainName\sAMAccountName of the object whose group memberships you want the cmdlet to retrieve. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

Indirect

Supply this parameter on the command line if you want the cmdlet to retrieve groups to which the object specified by the Identity parameter belongs because of group nesting, in addition to groups of which the object is a direct member.

If this parameter is omitted, the cmdlet retrieves only those groups of which the object is a direct member. If this parameter is supplied, the cmdlet takes the immediate group membership list of the object and then also recursively expands each group in this list to determine its group memberships to arrive at a complete set of the groups.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to retrieve the groups to which a particular object belongs in Active Directory. You can pass an object or one of its identity-related attributes to the Identity parameter and have the cmdlet return the object's groups that match the search conditions specified.

The output of the cmdlet is a collection of objects, with each object representing one of the groups found by the cmdlet. You can pipe the output into another cmdlet, such as Set-QADGroup, to make changes to the group objects returned by this cmdlet.

Examples

Example 1

Retrieve groups of which a particular user is a direct member:

```
C:\PS> Get-QADMemberOf 'domainName\userName'
```

Example 2

Retrieve groups whose names begin with DL.Administration and where a particular user has membership whether directly or because of group nesting:

```
C:\PS> Get-QADMemberOf 'domainName\userName' -Indirect -Name 'DL.Administration*'
```

Add-QADMemberOf

Make a particular object a member of particular groups in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Add-QADMemberOf [-Identity] <IdentityParameter> [-Group] <IdentityParameter[]>
[-Replace] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Group

Specify the DN, SID, GUID, or Domain\Name of the group or groups to which you want the cmdlet to add the object specified by the Identity parameter.

Identity

Specify the DN, SID, GUID, or Domain\Name of the object whose group memberships you want the cmdlet to modify.

This parameter is optional since you can pipe into this cmdlet the object returned by a Get-QAD* cmdlet.

Replace

Supply this parameter on the command line if you want the cmdlet to remove the object specified by the Identity parameter from all groups except those specified by the Group parameter.

If this parameter is omitted, the cmdlet adds the object to the groups specified and retains the object in all groups in which the object already has membership. If this parameter is supplied, the cmdlet adds the object to the groups specified and removes the object from any other groups.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add a single object to particular groups in Active Directory. You can specify a list of groups to which you want to add the object, separating the list entries by commas. The cmdlet also provides the option to remove the object from all groups other than those specified.

Examples

Example 1

Add a particular user to a particular group

```
C:\PS> add-QADMemberOf 'domainName\userName' -group 'domainName\groupName'
```

Example 2

Add a particular user to a particular group and remove that user from the other groups:

```
C:\PS> add-QADMemberOf 'domainName\userName' -group 'domainName\groupName' -Replace
```

Example 3

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
add-QADMemberOf 'domainName\userName' -group 'domainName\groupName'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

Remove-QADMemberOf

Remove a particular object from particular groups in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Remove-QADMemberOf [-Identity] <IdentityParameter> [-Group] <IdentityParameter[]>
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

```
Remove-QADMemberOf [-Identity] <IdentityParameter> -RemoveAll [-Control <Hashtable>]
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Group

Specify the DN, SID, GUID, or Domain\Name of the group or groups from which you want the cmdlet to remove the object specified by the Identity parameter.

Identity

Specify the DN, SID, GUID, or Domain\Name of the object whose group memberships you want the cmdlet to modify.

This parameter is optional since you can pipe into this cmdlet the object returned by a Get-QAD* cmdlet.

RemoveAll

Supply this parameter on the command line if you want the cmdlet to remove the object specified by the Identity parameter from all groups.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove a single object from particular groups in Active Directory. You can specify a list of groups from which you want to remove the object, separating the list entries by commas. The cmdlet also provides the option to remove the object from all groups.

Examples

Example 1

Remove a particular user from a particular group:

```
C:\PS> Remove-QADMemberOf 'domainName\userName' -Group 'domainName\groupName'
```

Example 2

Remove a particular user from all groups:

```
C:\PS> Remove-QADMemberOf 'domainName\userName' -RemoveAll
```

Get-QADComputer

Retrieve all computer objects in a domain or container that match the specified conditions.

Syntax

```
Get-QADComputer [[-Identity] <IdentityParameter>] [-ComputerRole <ComputerRole>]
[-ManagedBy <IdentityParameter[]>] [-SamAccountName <String[]>] [-DnsName <String[]>]
[-Location <String[]>] [-OSName <String[]>] [-OSVersion <String[]>] [-OSServicePack
<String[]>] [-Inactive] [-InactiveFor <Int32>] [-NotLoggedOnFor <Int32>]
[-PasswordNotChangedFor <Int32>] [-MemberOf <IdentityParameter[]>] [-IndirectMemberOf
<IdentityParameter[]>] [-NotMemberOf <IdentityParameter[]>] [-NotIndirectMemberOf
<IdentityParameter[]>] [-Tombstone] [-Recycled] [-LastKnownParent <IdentityParameter>]
[-SecurityMask <SecurityMasks>] [-SearchRoot <IdentityParameter[]>] [-SearchScope
<SearchScope>] [-AttributeScopeQuery <String>] [-PageSize <Int32>] [-SizeLimit <Int32>]
[-LdapFilter <String>] [-WildcardMode <WildcardMode>] [-SearchAttributes <Object>]
[-Description <String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr <String>]
[-Control <Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>]
[-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>]
[-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has a number of optional parameters for searching by individual attributes in the directory, with each parameter name identifying a certain attribute that you can search for attribute values specified by using the respective parameter (see the list of parameters for this cmdlet).

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

ComputerRole

Specify one of these parameter values: 'Member' (to search for computers that are not domain controllers) or 'DomainController' (to search for domain controllers only). If this parameter is omitted, the cmdlet searches for both domain controllers and computers that are not domain controllers.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

Inactive

Supply this parameter to retrieve accounts that meet the default inactivity conditions. You can view or change the default inactivity conditions by using the Get-QADInactiveAccountsPolicy or Set-QADInactiveAccountsPolicy cmdlet, respectively. When considering whether an account is inactive, the cmdlet verifies each of these values:

- The number of days that the account remains in the expired state
- The number of days that the password of the account remains unchanged
- The number of days that the account remains unused for logon

If any of these values exceeds a certain, default limit, then the account is considered inactive, and thus is retrieved by the Inactive parameter. The default limits can be overridden by supplying other account-inactivity related parameters, such as InactiveFor, ExpiredFor, NotLoggedOnFor, and PasswordNotChangedFor. Thus, if the NotLoggedOnFor value of 60 is supplied in conjunction with the Inactive parameter, the cmdlet searches for accounts that meet the default expiry-related or password-related inactivity condition, or have not been used to log on for 60 or more days.

To retrieve only those accounts that are not inactive, use the following syntax: -Inactive:\$false

InactiveFor

Use this parameter to retrieve accounts that meet any of the following conditions:

- The account remains in the expired state for at least the number of days specified by the parameter value
- The account does not have its password changed for at least the number of days specified by the parameter value
- The account has not been used to log on for at least the number of days specified by the parameter value

For example, the parameter value of 30 causes the cmdlet to search for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 30 or more days.

The value of this parameter overrides the default inactivity conditions, so the Inactive parameter has no effect when used together with this parameter. Similarly, the other account-inactivity related parameters such as ExpiredFor, NotLoggedOnFor and PasswordNotChangedFor override the corresponding conditions of this parameter. Thus, if the NotLoggedOnFor value of 60 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 60 or more days.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

IndirectMemberOf

Retrieve objects that belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has direct or indirect membership in the group specified by this parameter value.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LastKnownParent

When searching for a deleted object by using the Tombstone parameter, specify the DN of the container the object was in before it became a tombstone. This allows you to find objects that were deleted from a particular container.

Note that the lastKnownParent attribute is only set if the object was deleted on a domain controller running Windows Server 2003 or later version of Microsoft Windows Server. Therefore, it is possible that the lastKnownParent attribute value is inaccurate.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

MemberOf

Retrieve objects that are direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object has direct membership in the group specified by this parameter value.

NotIndirectMemberOf

Retrieve objects that do not belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has neither direct nor indirect membership in the group specified by this parameter value.

NotLoggedOnFor

Use this parameter to retrieve accounts that have not been used to log on for at least the number of days specified by the parameter value. This parameter overrides the logon-related inactivity condition of the Inactive or InactiveFor parameter. Thus, if the NotLoggedOnFor value of 60 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 30 or more days, or have not been used to log on for 60 or more days.

NotMemberOf

Retrieve objects that are not direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object does not have direct membership in the group specified by this parameter value.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

PasswordNotChangedFor

Use this parameter to retrieve accounts whose password has not been changed for at least the number of days specified by the parameter value. This parameter overrides the password-related inactivity condition of the Inactive or InactiveFor parameter. Thus, if the PasswordNotChangedFor value of 60 is supplied in conjunction with the InactiveFor value of 30, the cmdlet searches for accounts that are expired for 30 or more days, or have the password age of 60 or more days, or have not been used to log on for 30 or more days.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

Recycled

This parameter has an effect only if all of the following conditions are true:

- A domain is supplied as the SearchRoot parameter value.
- Active Directory Recycle Bin is enabled in that domain.

You can use this parameter in conjunction with the Tombstone parameter for the search results to include both the deleted and recycled objects that meet the search conditions. Without this parameter, the cmdlet returns only deleted objects.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SecurityMask

Specify which elements of the object's security descriptor to retrieve. Valid parameter values are:

- 'None' - do not retrieve any security data
- 'Owner' - retrieve the owner data
- 'Group' - retrieve the primary group data
- 'Dacl' - retrieve the discretionary access-control list data
- 'Sacl' - retrieve the system access-control list data

You can supply a combination of these values, separating them by commas. For example, you can supply the parameter value of 'Dacl,Sacl' in order to retrieve both the discretionary and system access-control list data.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Tombstone

Search for deleted objects of the respective object class. The search output is normally intended to be passed (piped in) to the Restore-QADDeletedObject cmdlet for restoring deleted objects.

In a domain with Active Directory Recycle Bin (a feature of Windows Server 2008 R2) this parameter retrieves deleted objects (rather than tombstones, which in that case are referred to as recycled objects). Recycle Bin preserves all attributes on the deleted objects, so you can use a search filter based on any attributes.

In a domain without Active Directory Recycle Bin, deleting an object converts that object to a tombstone. A search using this parameter returns tombstone objects that meet the filtering criteria supplied. Upon deletion of an object only a small number of the object's attributes are saved in the tombstone, with most of the attributes being lost. To search for deleted objects, your search filter should be based on the attributes that are preserved in tombstones.

When the Tombstone parameter is supplied, the search results include the deleted objects or tombstones that match the specified search filter. However, a search filter that matches a live object may not work as expected after the object is deleted. This is because not all attributes are retained in the tombstone. For example, a filter such as (&(objectClass=user)(objectCategory=person)) would not match any tombstone objects since the objectCategory attribute is removed upon object deletion. Conversely, the objectClass attribute is retained on tombstone objects, so a filter of (objectClass=user) would match deleted user objects.

The name of a tombstone object begins with the name of the deleted object, so a search using the Tombstone parameter can be refined by adding a filter based on object name. For example, to search for deleted objects with a name that begins with "John", you can use a filter such as (cn=John*).

It is also possible to find a specific deleted object. If you know the name of the object and the Distinguished Name (DN) of the container the object was in before it was deleted, then you can pass the container's DN to the LastKnownParent parameter and apply a filter of (cn=<name of the object>*) in order to have the cmdlet retrieve that specific object. However, if an object is deleted, a new object with the same DN is created, and then deleted as well, the above search would return more than one object. The returned objects are distinguished by the GUIDs of the deleted objects, with the name of each ending in the GUID of the respective deleted object.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Attribute-specific Parameters

The cmdlet also takes a series of optional, attribute-specific parameters allowing you to search by computer attributes. The attribute-specific parameters have effect if SearchRoot is specified whereas Identity is not. If you specify SearchRoot only, then the cmdlet returns all computer objects found in the SearchRoot container.

You can use attribute-specific parameters to search for computer objects that have specific values of certain attributes. With more than one attribute-specific parameter supplied, the search conditions are combined by using the AND operator, so as to find the computer objects that meet all the specified conditions.

If a particular attribute is referred to by both the SearchAttributes array and the attribute-specific parameter, the SearchAttributes setting has no effect on that attribute. The cmdlet searches for the attribute value specified by the attribute-specific parameter.

Each of the attribute-specific parameters accepts the asterisk (*) wildcard character in the parameter value to match zero or more characters (case-insensitive).

The following table lists the attribute-specific parameters you can use with this cmdlet. Each parameter adds a filter condition based on a certain attribute identified by the LDAP display name in the table.

TO SEARCH BY THIS ATTRIBUTE...	USE THIS SYNTAX
description	-Description <String[]>
displayName	-DisplayName <String[]>
samAccountName	-SamAccountName <String[]>
dNSHostName	-DnsName <String[]>
location	-Location <String[]>
managedBy	-ManagedBy <IdentityParameter[]>
operatingSystem	-OSName <String[]>
operatingSystemVersion	-OSVersion <String[]>
operatingSystemServicePack	-OSServicePack <String[]>

Detailed Description

Use this cmdlet to search an Active Directory domain or container for computer objects that meet certain criteria, or to bind to a certain computer object by DN, SID, GUID, or Domain\Name. You can search by computer attributes or specify your search criteria by using an LDAP search filter.

The output of the cmdlet is a collection of objects, with each object representing one of the computer objects found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the computer objects returned by this cmdlet.

Examples

Example 1

Bind to a particular computer by Domain\Name, and display the computer's name and DN. In this example, the NetBIOS name of the domain is assumed to be "MyDomain" and the pre-Windows 2000 name of the computer is assumed to be "MyServer":

```
C:\PS> get-QADComputer 'MyDomain\MyServer$'
```

Example 2

With a specific OU, find all computers that run a particular version of the operating system, and list the names of the computers found. The OU is identified by its canonical name.

```
C:\PS> get-QADComputer -SearchRoot 'company.com/computersOU' -OSName '*Vista*'
```

Example 3

Find all domain controllers in your domain, and list their names and DNs:

```
C:\PS> get-QADComputer -computerRole 'DomainController'
```

Example 4

Find all computers in your domain; for each computer found, display the pre-Windows 2000 computer name along with the operating system name, version, and service pack:

```
C:\PS> Get-QADComputer | format-table -property computername, osname, osversion, osservicepack
```

This command displays the computers in a table with four columns: "ComputerName", "OSName", "OSVersion" and "OSServicePack." The command uses the **Get-QADComputer** cmdlet to get all of the computers. The pipeline operator (|) sends the results to the **Format-Table** cmdlet, which formats the output in a table. The *Property* parameter specifies the properties that appear in the table as columns.

Note: ComputerName, OSName, OSVersion and OSServicePack are just four of the properties of an object returned by the **Get-QADComputer** cmdlet. To see all of the properties, type the following command:

```
C:\PS> get-qadcomputer | get-member
```

Example 5

Connect to a specific domain controller with the credentials of a specific user, and bind to a certain computer account by Domain\Name, display the computer name and description, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> get-QADComputer 'company\computer$' | ft computername, description
C:\PS> disconnect-QADService
```

Example 6

Connect to any available domain controller with the credentials of the locally logged on user, search for computers in a specific container by using an LDAP search filter, and display the name and DN of each computer found:

```
C:\PS> get-QADComputer -SearchRoot 'company.com/ComputersOU' -LdapFilter
'(description=a*)'
```

Example 7

Search a certain container to find all computers with empty description, and set a description for each of those computers:

```
C:\PS> get-QADComputer -SearchRoot 'company.com/ComputersOU' -description '' |
set-QADObject -description 'A description'
```

Set-QADComputer

Make changes to a computer object in Active Directory Domain Services.

Syntax

```
Set-QADComputer [-Identity] <IdentityParameter> [-SecondaryOwner  
<UpdateIdentityParameter[]>] [-TrustForDelegation] [-SamAccountName <String>]  
[-ManagedBy <IdentityParameter>] [-Location <String>] [-Password <String>]  
[-ObjectAttributes <ObjectAttributesParameter>] [-Description <String>] [-DisplayName  
<String>] [-ExcludedProperties <String[]>] [-IncludedProperties <String[]>]  
[-DeserializeValues] [-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy]  
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]  
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection  
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

Location

Set the 'location' attribute to this parameter value.

ManagedBy

Specify the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as the 'managedBy' attribute value on the object in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

Password

Set the password in the computer object to this parameter value.

SamAccountName

Set the 'sAMAccountName' attribute (pre-Windows 2000 name) to this parameter value.

SecondaryOwner

Use this parameter to add or remove secondary owners. Parameter value can be a string array or an associative array that specifies the identifiers, such as DN, SID, GUID, UPN or Domain\Name, of one or more users or groups to add or remove from the secondary owner role. Some examples of possible parameter values are:

```
-SecondaryOwner 'domain\administrator','domain\user'
```

Replace the existing identities in the secondary owners list with the identities specified.

```
-SecondaryOwner @{append=@('domain\administrator','domain\user')}
```

Add the specified identities to the secondary owners list, without removing the existing owners.

```
-SecondaryOwner @{delete=@('domain\administrator','domain\user')}
```

Remove the specified identities from the secondary owners list, leaving the other owners intact.

```
-SecondaryOwner $null
```

Clear the secondary owners list, so that no secondary owners are specified.

This parameter has an effect only in conjunction with the Proxy connection parameter because the secondary owner settings are stored and managed by Quest One ActiveRoles.

TrustForDelegation

Supply the parameter value of \$true or \$false depending on whether or not you want to configure the computer object so that the computer is trusted for delegation. When a computer is trusted for delegation, any service running under the Local System account on that computer can access resources on other computers and impersonate its clients when accessing resources on other computers.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to modify properties of an Active Directory computer object. You can modify some commonly used computer properties by using the corresponding cmdlet parameters. Properties that are not associated with cmdlet parameters can be modified by using the `ObjectAttributes` parameter.

Thus, to modify the value of the 'description' or 'displayName' attribute, you can use the `Description` or `DisplayName` parameter, respectively. If a particular attribute is referred to by both the `ObjectAttributes` array and an attribute-specific parameter, the `ObjectAttributes` setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

Examples

Example 1

Modify the location for a given computer:

```
C:\PS> Set-QADComputer 'lab.local/computers/Comp4' -Location 'AMS/HQ/Building A'
```

Example 2

Set the 'managedBy' attribute value for a given computer using the pre-Windows 2000 logon name of the user:

```
C:\PS> Set-QADComputer 'lab.local/computers/Comp4' -ManagedBy 'domainName\logonName'
```

Example 3

Replace the SPN values for a given computer:

```
C:\PS> Set-QADComputer 'lab.local/computers/Comp4' -objectAttributes  
{servicePrincipalName=@('MSSQLSvc/Comp4.lab.local:1362','ldap/Comp4.lab.local:389')}
```

Example 4

Add two new items to the SPN values for a given computer:

```
C:\PS> Set-QADComputer 'lab.local/computers/Comp4' -objectAttributes  
{servicePrincipalName=@(Append=@('MSSQLSvc/Comp4.lab.local:1362','ldap/Comp4.lab.local:389'))}
```

Example 5

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
Set-QADComputer 'lab.local/computers/Comp4' -ManagedBy 'domainName\logonName'  
$op = get-QARSLastOperation  
If($op.OperationStatus -eq 'Pending')  
{  
    # Operation submitted for approval. Handle this case here.  
}  
elseif($op.OperationStatus -eq 'Completed')  
{  
    # No approval required. Operation completed.  
}  
else  
{  
    # Operation not completed for some reason. You might check other OperationStatus  
    # values to determine the status of the operation request.  
}
```

New-QADComputer

Create a new computer object in Active Directory Domain Services.

Syntax

```
New-QADComputer [-Name] <String> -ParentContainer <IdentityParameter> [-SecondaryOwner
<IdentityParameter[]>] [-TrustForDelegation] [-SamAccountName <String>] [-ManagedBy
<IdentityParameter>] [-Location <String>] [-Password <String>] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

Location

Set the 'location' attribute to this parameter value.

ManagedBy

Specify the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as the 'managedBy' attribute value on the object in the directory.

Name

Specify the name for the new computer object to be created.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ParentContainer

Specify the distinguished name (DN) of the container in which you want this cmdlet to create a new computer object.

Password

Set the password in the computer object to this parameter value.

SamAccountName

Set the 'sAMAccountName' attribute (pre-Windows 2000 name) to this parameter value.

SecondaryOwner

Set the 'edsvaSecondaryOwners' attribute. Supply the DN, SID, GUID, UPN or Domain\Name of the user or group to be set as a secondary owner. You can supply a string array of object identifiers to specify several secondary owners. This parameter has an effect only in conjunction with the Proxy connection parameter because the secondary owner settings are stored and managed by Quest One ActiveRoles.

TrustForDelegation

Supply this parameter to configure the computer object so that the computer is trusted for delegation. In this way you enable any service running under the Local System account on that computer to gain access to resources on other computers, and to impersonate its clients when accessing resources on other computers.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to create a new Active Directory computer object. You can use this cmdlet to provision a computer account before the computer is added to the domain. Note that this cmdlet does not join a computer to a domain.

You can set some commonly used computer properties by using the corresponding cmdlet parameters. Properties that are not associated with cmdlet parameters can be set by using the ObjectAttributes parameter. Thus, to set the value of the 'description' or 'displayName' attribute, you can use the Description or DisplayName parameter, respectively. If a particular attribute is referred to by both the ObjectAttributes array and an attribute-specific parameter, the ObjectAttributes setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

Examples

Example 1

Create a new computer object in the 'Computers' container, and set a particular location value for that object:

```
C:\PS> New-QADComputer -Name 'LAB-SRV3' -SamAccountName 'LAB-SRV3$' -ParentContainer  
'CN=Computers,DC=lab,DC=local' -Location 'AMS/HQ/Building A'
```

Disable-QADComputer

Disable a computer object in Active Directory Domain Services.

Syntax

```
Disable-QADComputer [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, or SID of the computer account you want to disable. The cmdlet makes changes to the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to disable a computer account in Active Directory.

Examples

Example 1

Disable the computer account identified by its Distinguished Name:

```
C:\PS> disable-QADComputer 'CN=LAB-SRV1,CN=Computers,DC=dom,DC=local'
```

Example 2

Disable all computer accounts in the container identified by its Canonical Name:

```
C:\PS> get-QADComputer -SearchRoot 'dom.local/labComputers' | disable-QADComputer
```

Enable-QADComputer

Enable a disabled computer object in Active Directory Domain Services.

Syntax

```
Enable-QADComputer [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, or SID of the computer account you want to enable. The cmdlet makes changes to the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to re-enable a disabled computer account in Active Directory.

Examples

Example 1

Enable the computer account identified by its Distinguished Name:

```
C:\PS> enable-QADComputer 'CN=LAB-SRV1,CN=Computers,DC=dom,DC=local'
```

Example 2

Enable all computer accounts in the container identified by its Canonical Name:

```
C:\PS> get-QADComputer -SearchRoot 'dom.local/labComputers' | enable-QADComputer
```

Reset-QADComputer

Reset a computer object in Active Directory Domain Services.

Syntax

```
Reset-QADComputer [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, or SID of the computer account you want to reset. The cmdlet makes changes to the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to reset a computer account in Active Directory. When resetting a computer account, you reset the secure channel between the computer that uses that account to join the domain and a domain controller in the domain. A secure channel is necessary for a computer to authenticate successfully to the domain. Note that resetting a computer account breaks that computer's connection to the domain and requires it to rejoin the domain.

Examples

Example 1

Reset the computer account identified by its Distinguished Name:

```
C:\PS> reset-QADComputer 'CN=LAB-SRV1,CN=Computers,DC=dom,DC=local'
```

Get-QADObject

Retrieve all directory objects in a domain or container that match the specified conditions. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Get-QADObject [[-Identity] <IdentityParameter>] [-Type <String>]
[-ResolveForeignSecurityPrincipals] [-ProxyAddress <String[]>] [-PrimaryProxyAddress
<String[]>] [-SecondaryProxyAddress <String[]>] [-MemberOf <IdentityParameter[]>]
[-IndirectMemberOf <IdentityParameter[]>] [-NotMemberOf <IdentityParameter[]>]
[-NotIndirectMemberOf <IdentityParameter[]>] [-Tombstone] [-Recycled] [-LastKnownParent
<IdentityParameter>] [-SecurityMask <SecurityMasks>] [-SearchRoot
<IdentityParameter[]>] [-SearchScope <SearchScope>] [-AttributeScopeQuery <String>]
[-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode
<WildcardMode>] [-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName
<String[]>] [-Name <String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn
<DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn
<DateTime>] [-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>]
[-IncludeAllProperties] [-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>]
[-IncludedProperties<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>]
[-ShowProgress] [-Activity <String>] [-ProgressThreshold <Int32>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "memberOf" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

IndirectMemberOf

Retrieve objects that belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has direct or indirect membership in the group specified by this parameter value.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LastKnownParent

When searching for a deleted object by using the Tombstone parameter, specify the DN of the container the object was in before it became a tombstone. This allows you to find objects that were deleted from a particular container.

Note that the lastKnownParent attribute is only set if the object was deleted on a domain controller running Windows Server 2003 or later version of Microsoft Windows Server. Therefore, it is possible that the lastKnownParent attribute value is inaccurate.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

MemberOf

Retrieve objects that are direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object has direct membership in the group specified by this parameter value.

Name

Search by the 'name' attribute.

NotIndirectMemberOf

Retrieve objects that do not belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has neither direct nor indirect membership in the group specified by this parameter value.

NotMemberOf

Retrieve objects that are not direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object does not have direct membership in the group specified by this parameter value.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

PrimaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a primary (reply-to) e-mail address.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients that have any of the specified e-mail addresses.

Recycled

This parameter has an effect only if all of the following conditions are true:

- A domain is supplied as the SearchRoot parameter value.
- Active Directory Recycle Bin is enabled in that domain.

You can use this parameter in conjunction with the Tombstone parameter for the search results to include both the deleted and recycled objects that meet the search conditions. Without this parameter, the cmdlet returns only deleted objects.

ResolveForeignSecurityPrincipals

Supply this parameter if you want the cmdlet to resolve the retrieved foreign security principal objects by looking up the corresponding external security principals. Thus, when retrieving objects, the cmdlet may encounter a foreign security principal object - an object held in a domain within a given forest that represents a security principal (such as a user, computer, or group) that exists in a trusted domain located in a different forest. By default, the cmdlet outputs the foreign security principal object data "as is," without attempting to look up the corresponding (external) security principal based on the data found in the foreign security principal object. With the `ResolveForeignSecurityPrincipals` parameter, once the cmdlet has encountered a foreign security principal object, it attempts to look up the external security principal represented by the foreign security principal object, and, in case of a successful lookup, outputs the security principal data instead of the foreign security principal object data.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the `IncludeAllProperties` parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the `Get-QADUser` or `Get-QADObject` cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which `SearchRoot` is the topmost object (sub-tree search). This default behavior can be altered by using the `SearchScope` parameter.

The search criteria are defined by the `LdapFilter` parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an `Identity` value is supplied. If you want this parameter to have effect, do not supply any `Identity` parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SecondaryProxyAddress

Specify one or more e-mail addresses to retrieve Exchange recipients for which any of the specified e-mail addresses is set as a non-primary e-mail address.

SecurityMask

Specify which elements of the object's security descriptor to retrieve. Valid parameter values are:

- 'None' - do not retrieve any security data
- 'Owner' - retrieve the owner data
- 'Group' - retrieve the primary group data
- 'Dacl' - retrieve the discretionary access-control list data
- 'Sacl' - retrieve the system access-control list data

You can supply a combination of these values, separating them by commas. For example, you can supply the parameter value of 'Dacl,Sacl' in order to retrieve both the discretionary and system access-control list data.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Tombstone

Search for deleted objects of the respective object class. The search output is normally intended to be passed (piped in) to the Restore-QADDeletedObject cmdlet for restoring deleted objects.

In a domain with Active Directory Recycle Bin (a feature of Windows Server 2008 R2) this parameter retrieves deleted objects (rather than tombstones, which in that case are referred to as recycled objects). Recycle Bin preserves all attributes on the deleted objects, so you can use a search filter based on any attributes.

In a domain without Active Directory Recycle Bin, deleting an object converts that object to a tombstone. A search using this parameter returns tombstone objects that meet the filtering criteria supplied. Upon deletion of an object only a small number of the object's attributes are saved in the tombstone, with most of the attributes being lost. To search for deleted objects, your search filter should be based on the attributes that are preserved in tombstones.

When the Tombstone parameter is supplied, the search results include the deleted objects or tombstones that match the specified search filter. However, a search filter that matches a live object may not work as expected after the object is deleted. This is because not all attributes are retained in the tombstone. For example, a filter such as (&(objectClass=user)(objectCategory=person)) would not match any tombstone objects since the objectCategory attribute is removed upon object deletion. Conversely, the objectClass attribute is retained on tombstone objects, so a filter of (objectClass=user) would match deleted user objects.

The name of a tombstone object begins with the name of the deleted object, so a search using the Tombstone parameter can be refined by adding a filter based on object name. For example, to search for deleted objects with a name that begins with "John", you can use a filter such as (cn=John*).

It is also possible to find a specific deleted object. If you know the name of the object and the Distinguished Name (DN) of the container the object was in before it was deleted, then you can pass the container's DN to the LastKnownParent parameter and apply a filter of (cn=<name of the object>*) in order to have the cmdlet retrieve that specific object. However, if an object is deleted, a new object with the same DN is created, and then deleted as well, the above search would return more than one object. The returned objects are distinguished by the GUIDs of the deleted objects, with the name of each ending in the GUID of the respective deleted object.

Type

Specify the type of directory objects to find. The cmdlet searches for objects that have one of the 'objectClass' attribute values set to the Type parameter value.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to search an Active Directory domain or container for directory objects that meet certain criteria, or to bind to a certain object by DN, SID, GUID, UPN, or Domain\Name. You can search by object attributes or specify your search criteria by using an LDAP search filter.

The output of the cmdlet is a collection of objects, with each object representing one of the directory objects found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the directory objects returned by this cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific object by DN, and display the description of the object:

```
C:\PS> (get-QADObject 'CN=John
Smith,OU=CompanyOU,DC=company,DC=com').DirectoryEntry.description
```

Example 2

Connect to a specific domain controller with the credentials of a specific user, bind to a certain object by SID, display the description of the object, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount
'company\administrator' -ConnectionPassword $pw
C:\PS> (get-QADObject -identity
'S-1-5-21-1279736177-1630491018-182859109-1305').DirectoryEntry.description
C:\PS> disconnect-QADService
```

Example 3

Connect to any available domain controller with the credentials of the locally logged on user, search for objects in a specific container by using an LDAP search filter, and display a list of the objects found:

```
C:\PS> get-QADObject -SearchRoot 'company.com/UsersOU' -LdapFilter '(description=a*)'
```

Example 4

Connect to any available domain controller with the credentials of the locally logged on user, find all computer objects in a specific container, and display a list of the objects found:

```
C:\PS> get-QADObject -SearchRoot 'company.com/ComputersOU' -Type Computer
```

Example 5

Connect to any available domain controller with the credentials of a specific user, search a certain container to find all groups with the empty Notes field, set a note for each of those groups, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -ConnectionAccount 'company\administrator'
-ConnectionPassword $pw
C:\PS> get-QADObject -SearchRoot 'company.com/GroupsOU' -Type Group -SearchAttributes
@{info='' } | set-QADObject -ObjectAttributes @{info='A note'}
C:\PS> disconnect-QADService
```

Example 6

List the names of all properties of organizationalUnit objects. Sort the list by property name:

```
C:\PS> get-QADObject -Type 'organizationalUnit' -IncludeAllProperties
-ReturnPropertyNamesOnly | ForEach-Object {$_} | Sort-Object
```

Example 7

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, find all AD LDS objects in a specific container, and display a list of the objects found:

```
C:\PS> get-QADObject -Service 'server.domain.local:389' -SearchRoot '<DN of container>'
```

Example 8

Retrieve all objects that were deleted on the current date:

```
C:\PS> Get-QADObject -Tombstone -LastChangedOn (get-date)
```

Example 9

Retrieve all foreign security principal objects from the current domain, replacing each one, if possible, with the corresponding external security principal data in the output stream:

```
C:\PS> Get-QADObject -ResolveForeignSecurityPrincipals -Type foreignSecurityPrincipal
```

Get-QADManagedObject

Retrieve objects for which a particular user, contact or group is the manager (primary owner) or a secondary owner.

Syntax

```
Get-QADManagedObject [-Identity] <IdentityParameter> [-Type <String>]
[-ConsiderSecondaryOwnership] [-ConsiderInheritedOwnership] [-PageSize <Int32>]
[-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode <WildcardMode>]
[-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName <String[]>] [-Name
<String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn <DateTime>]
[-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>]
[-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also ShowProgress and ProgressThreshold). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)

- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

ConsiderInheritedOwnership

This parameter causes the cmdlet to retrieve the objects for which the given identity inherits the owner role from a group that is assigned as an owner. Without this parameter, the cmdlet retrieves only the objects for which the identity itself is assigned as an owner (that is, specified in the 'managedBy' or 'edsvaSecondaryOwners' attribute of the object).

This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the Proxy connection parameter.

ConsiderSecondaryOwnership

This parameter causes the cmdlet to retrieve the objects for which the given identity is a manager (primary owner) or secondary owner. Without this parameter, the cmdlet retrieves only the objects for which that identity is a manager (primary owner).

This parameter requires a connection to Quest One ActiveRoles, and therefore it should be used in conjunction with the Proxy connection parameter.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, UPN or SID of a user, group or contact. The cmdlet searches for the objects for which the specified user, group or contact is the manager (primary owner) or a secondary owner.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Type

Specify the type of directory objects to find. The cmdlet searches for objects that have one of the 'objectClass' attribute values set to the Type parameter value.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

For a particular identity (user, group or contact), you can use this cmdlet to search an Active Directory domain or container for directory objects such as groups, computers or organizational units that:

- Have the given identity designated as the manager in Active Directory (the identity is specified in the managedBy attribute of the object)
- Have the given identity designated as a secondary owner in Quest One ActiveRoles (the identity is specified in the edsvaSecondaryOwners attribute of the object)
- Have a group designated as the manager, with the given identity belonging to that group (so the identity inherits the manager role from the group)
- Have a group designated as a secondary owner, with the given identity belonging to that group (so the identity inherits the secondary owner role from the group)

In Quest One ActiveRoles, the identity that is designated as the manager of an object is referred to as the primary owner of that object. The primary owner role may also be inherited from a group that is designated as the manager. The cmdlet allows you to retrieve the objects for which a particular identity holds the owner role, whether primary, secondary, or both. It is possible to specify whether you want the search results to include the objects for which the given identity inherits the owner role from a group.

By default, the cmdlet searches for only the objects that have the specified identity designated as the manager in Active Directory. You can broaden the search by using the *ConsiderSecondaryOwnership* or *ConsiderInheritedOwnership* parameter.

The output of the cmdlet is a collection of objects, with each object representing one of the directory objects found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the directory objects returned by this cmdlet.

Examples

Example 1

Retrieve the groups for which the specified user is assigned as the manager:

```
C:\PS> get-QADManagedObject 'domainName\userName' -Type 'group'
```

Example 2

Retrieve the groups for which the specified user is assigned as the manager (primary owner) or as a secondary owner:

```
C:\PS> get-QADManagedObject 'domainName\userName' -Type 'group' -Proxy  
-ConsiderSecondaryOwnership
```

Example 3

Retrieve the objects for which the specified user meets any of the following requirements:

- The user is assigned as the manager of the object
- The user belongs to any group that is assigned as the manager of the object

```
C:\PS> get-QADManagedObject 'domainName\userName' -Proxy -ConsiderInheritedOwnership
```

Set-QADObject

Modify attributes of an object in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Set-QADObject [-Identity] <IdentityParameter> [-ObjectAttributes  
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]  
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]  
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]  
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]  
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the `Serialize` parameter). For examples of how to export and import an object, see documentation on the `Get-QADUser` cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the `DomainName\sAMAccountName`, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with `UseDefaultExcludedProperties`, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both `ExcludedProperties` and `IncludedProperties`, the cmdlet does not set the value of that attribute in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the `Get-QADPSSnapinSettings` cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to change or remove values of attributes of an object in Active Directory.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific object by DN, and modify the description of the object:

```
C:\PS> set-QADObject 'CN=John Smith,OU=CompanyOU,DC=company,DC=com' -description 'Sales person'
```

Example 2

Connect to a specific domain controller with the credentials of a specific user, bind to a certain object by SID, modify the description of the object, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'server.company.com' -ConnectionAccount 'company\administrator' -ConnectionPassword $pw
C:\PS> set-QADObject -identity 'S-1-5-21-1279736177-1630491018-182859109-1305' -description 'Service account'
C:\PS> disconnect-QADService
```

Example 3

Connect to the local Administration Service with the credentials of a specific user, bind to a certain object by Domain\Name, set or clear certain attributes, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount 'company\administrator' -ConnectionPassword $pw
C:\PS> set-QADObject -identity 'company\associates' -ObjectAttributes @{info='';description='All company associates'}
C:\PS> disconnect-QADService
```

Example 4

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS object by DN, and modify the description of the object:

```
C:\PS> set-QADObject '<DN of object>' -Service 'server.domain.local:389' -description 'My AD LDS object'
```

Example 5

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
set-QADObject 'CN=John Smith,OU=CompanyOU,DC=company,DC=com' -description 'Sales person'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

New-QADObject

Create a new object of in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
New-QADObject [-Name] <String> -ParentContainer <IdentityParameter> -Type <String>
[-NamingProperty <String>] [-ObjectAttributes <ObjectAttributesParameter>]
[-Description <String>] [-DisplayName <String>] [-ExcludedProperties <String[]>]
[-IncludedProperties <String[]>] [-DeserializeValues] [-UseDefaultExcludedProperties]
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

Name

Set the 'name' attribute to this parameter value on the new object created by this cmdlet in the directory.

NamingProperty

Supply the LDAP name of the naming attribute specific to the object class of the directory object you want to create. The naming attribute qualifies the object's relative Distinguished Name. If this parameter is omitted, the naming attribute is assumed to be 'cn', which is suitable for most object classes. Other possible values are 'ou' (naming attribute of the organizationalUnit object class) and 'dc' (naming attribute of the domain object class).

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ParentContainer

Specify the Distinguished Name of the container in which you want the new directory object to be created by this cmdlet.

Type

Specify the object class of the directory object to be created. This is the name of a schema class object, such as User or Group. The cmdlet creates a directory object of the object class specified by the value of this parameter.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to create a directory object of the specified schema class and a particular name in the container. The cmdlet also allows for setting other properties (for example, the mandatory properties) on the new object.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, and create a new computer object:

```
C:\PS> new-qadObject -ParentContainer 'OU=ComputersOU,DC=company,DC=com' -type  
'computer' -name 'comp1' -ObjectAttributes @{sAMAccountName='comp1$'}
```

Example 2

Connect to the local Administration Service with the credentials of a specific user, create a new organizational unit, and then disconnect:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString  
C:\PS> connect-qadService -service 'localhost' -proxy -ConnectionAccount  
'company\administrator' -ConnectionPassword $pw  
C:\PS> new-qadObject -ParentContainer 'OU=companyOU,DC=company,DC=com' -type  
'organizationalUnit' -NamingProperty 'ou' -name 'Child OU'  
C:\PS> disconnect-qadService
```

Example 3

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, and create a new AD LDS user object in a certain container:

```
C:\PS> new-QADObject -Service 'server.domain.local:389' -ParentContainer '<DN of  
container>' -Type 'user' -Name 'John Smith'
```

Move-QADObject

Move the specified object to a different location (container) in Active Directory.

Syntax

```
Move-QADObject [-Identity] <IdentityParameter> -NewParentContainer <IdentityParameter>
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

NewParentContainer

Specify the Distinguished Name of the destination container (the container to which you want to move the object).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to move an object between containers within an Active Directory domain (the cmdlet cannot move an object to a different domain). An object to move can be specified by DN, SID, GUID, UPN or Domain\Name, or it can be located by using a **Get-** cmdlet and then piped into the **Move-** cmdlet. The destination container can be specified by DN or GUID.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a particular user object by Domain\Name, and move the object to the specified container:

```
C:\PS> move-QADObject 'MyDomain\JSmith' -NewParentContainer  
'MyDomain.company.com/NewYork/Users'
```

Rename-QADObject

Change the name of the specified object in Active Directory.

Syntax

```
Rename-QADObject [-Identity] <IdentityParameter> -NewName <String> [-Control
<Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount
<String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

NewName

Specify the new name to assign to the directory object (set the 'name' attribute to this parameter value).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to rename an object in Active Directory. The cmdlet sets the *name* attribute of the object to the value specified, thus causing the corresponding change to the distinguished name of the object. An object to rename can be specified by DN, SID, GUID, UPN or Domain\Name, or it can be located by using a **Get-** cmdlet and then piped into the **Rename-** cmdlet.

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific user object by Domain\Name, and assign the new name to the object:

```
C:\PS> rename-QADObject 'MyDomain\JSmith' -NewName 'Jane Smith'
```

Remove-QADObject

Delete the specified objects in Active Directory. Supported are both Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS).

Syntax

```
Remove-QADObject [-Identity] <IdentityParameter> [-DeleteTree] [-Force] [-Control
<Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount
<String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

DeleteTree

Deletes the specified object along with all of its descendant objects (the entire sub-tree). If you omit this parameter, the cmdlet fails to delete container objects that hold any child objects.

Force

Overrides restrictions that prevent the command from succeeding, just so the changes do not compromise security. Suppresses the warning or confirmation messages that could appear during changes caused by execution of the command. This parameter is useful when you run the command programmatically, so interactive input is inappropriate.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to delete objects in Active Directory. An object to delete can be specified by DN, SID, GUID, UPN or Domain\Name, or it can be located by using a **Get-** cmdlet and then piped into the **Remove-** cmdlet (see examples).

Examples

Example 1

Connect to any available domain controller with the credentials of the locally logged on user, bind to a specific user object by Domain\Name, and delete the object:

```
C:\PS> remove-QADObject 'MyDomain\JSmith'
```

Example 2

Connect to any available domain controller with the credentials of the locally logged on user, and delete all user objects that are located in a specific container:

```
C:\PS> get-QADUser -searchRoot 'mydomain.company.com/usersOU' | remove-QADObject  
-confirm
```

Example 3

Connect to the local Administration Service with the credentials of a specific user, and delete a certain container along with all objects that are located in that container:

```
C:\PS> $pw = read-host "Enter password" -AsSecureString  
C:\PS> connect-QADService -service 'localhost' -proxy -ConnectionAccount  
'company\administrator' -ConnectionPassword $pw  
C:\PS> remove-QADObject 'mydomain.company.com/usersOU' -deleteTree -force  
C:\PS> disconnect-QADService
```

Example 4

Connect to the AD LDS instance on 'server.domain.local:389' with the credentials of the locally logged on user, bind to a specific AD LDS object by DN, and delete the object:

```
C:\PS> remove-QADObject '<DN of object>' -service 'server.domain.local:389' -confirm
```

Restore-QADDeletedObject

Undelete objects in Active Directory by restoring tombstones back into normal objects. This cmdlet requires an Active Directory domain controller running Windows Server 2003 or later. If Active Directory Recycle Bin (a feature of Windows Server 2008 R2) is enabled, this cmdlet restores deleted objects from Recycle Bin.

Syntax

```
Restore-QADDeletedObject -TargetObject <IGenericDirectoryObject> [-RestoreChildren]
[-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

RestoreChildren

This parameter causes the cmdlet to restore both the target object and all of its deleted descendants. Without this parameter, only the target object is restored.

TargetObject

Supply tombstone objects to restore. This parameter is normally used to receive the output of a Get-QAD cmdlet searching for deleted objects (see Tombstone parameter on Get-QAD* cmdlets).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to recover deleted objects in Active Directory. Pipe a deleted object into this cmdlet in order to restore (undelete) that object. Deleted objects can be retrieved by using an appropriate Get-QAD* cmdlet with the Tombstone parameter.

When an object is deleted, it is not physically removed from the Active Directory database. Instead, Active Directory marks the object as deleted, clears most of the object's attributes, renames the object and moves it to a special container. The object becomes a tombstone. This cmdlet causes Active Directory to perform the necessary operations on the tombstone to reanimate the object, which effectively results in the object being undeleted.

The object's attributes that were cleared upon object deletion are not restored. However, certain attributes, the most important of which are identity-related attributes such as objectGUID and objectSid, are retained in the tombstone, and restored when the object is undeleted. Which attributes are retained in tombstones is determined by Active Directory. Thus, the attributes that have the 0x8 bit set in the searchFlags attribute of the attributeSchema definition are retained.

The cmdlet relies on the "restore deleted objects" feature of Active Directory. To enable this feature, at least one domain controller in the domain must be running on Windows Server 2003 or a later version of Microsoft Windows Server. Ensure that the cmdlet is connected to such a domain controller. Normally, only domain administrators are allowed to restore deleted objects. For information about access rights required to restore deleted objects and limitations that apply to restoring deleted objects, see topic "Restoring Deleted Objects" in MSDN Library at <http://msdn.microsoft.com>.

If Active Directory Recycle Bin (a feature of Windows Server 2008 R2) is enabled, the deletion of an object does not turn the object into a tombstone. The object is marked as deleted and moved to a special container, but the attributes of the object are not cleared. In this case, the cmdlet restores the deleted object with all attributes, including the link-valued attributes such as Member Of. As a result, the object is restored to the same state it was in immediately before deletion. For example, a restored user account regains all group memberships that it had at the time of deletion.

Examples

Example 1

Restore a user account that was deleted from a particular container and had the name (RDN) of John Smith:

```
C:\PS> Get-QADUser -Tombstone -LastKnownParent '<DN of container>' -Name 'John Smith*' |  
Restore-QADDeletedObject
```

Example 2

Restore all user accounts that were deleted from a particular container on the current date:

```
C:\PS> Get-QADUser -Tombstone -LastKnownParent '<DN of container>' -LastChangedOn  
(get-date) | Restore-QADDeletedObject
```

Example 3

Restore all user accounts that were deleted on September 1, 2008:

```
C:\PS> Get-QADUser -Tombstone -LastChangedOn (get-date -year 2008 -month 9 -day 1) |  
Restore-QADDeletedObject
```

Example 4

Restore a deleted container and all objects that existed in that container when it was deleted:

```
C:\PS> Get-QADObject <DN of container> -Tombstone | Restore-QADDeletedObject  
-RestoreChildren
```

New-QADPasswordSettingsObject

Create a new Password Settings object (PSO). Active Directory version of Windows Server 2008 or later is required.

Syntax

```
New-QADPasswordSettingsObject [-Name] <String> [-ParentContainer <IdentityParameter>]
[-AppliesTo <IdentityParameter[]>] [-Precedence <Int32>] [-ReversibleEncryptionEnabled]
[-PasswordHistoryLength <Int32>] [-PasswordComplexityEnabled] [-MinimumPasswordLength
<Int32>] [-MinimumPasswordAge <TimeSpanAndDaysParameter>] [-MaximumPasswordAge
<TimeSpanAndDaysParameter>] [-LockoutThreshold <Int32>] [-ResetLockoutCounterAfter
<TimeSpanAndMinutesParameter>] [-LockoutDuration <TimeSpanAndMinutesParameter>]
[-ObjectAttributes <ObjectAttributesParameter>] [-Description <String>] [-DisplayName
<String>] [-ExcludedProperties <String[]>] [-IncludedProperties <String[]>]
[-DeserializeValues] [-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

AppliesTo

Specify a list of users or groups to which you want the Password Settings object to apply. Each list entry is the DN, SID, GUID, UPN or Domain\Name of a user or group. Separate the list entries by commas.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Confirm

Prompts you for confirmation before executing the command.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

LockoutDuration

Specify the Lockout Duration setting for locked out user accounts (set the 'msDS-LockoutDuration' attribute to this parameter value).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of minutes). This must be a negative value (see examples).

LockoutThreshold

Specify the Lockout Threshold setting for lockout of user accounts (set the 'msDS-LockoutThreshold' attribute to this parameter value).

MaximumPasswordAge

Specify the Maximum Password Age setting for user accounts (set the 'msDS-MaximumPasswordAge' attribute to this parameter value).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of days). This must be a negative value (see examples).

MinimumPasswordAge

Specify the Minimum Password Age setting for user accounts (set the 'msDS-MinimumPasswordAge' attribute to this parameter value).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of days). This must be a negative value (see examples).

MinimumPasswordLength

Specify the Minimum Password Length setting for user accounts (set the 'msDS-MinimumPasswordLength' attribute to this parameter value).

Name

Set the 'name' attribute to this parameter value on the new object created by this cmdlet in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ParentContainer

Specify the Distinguished Name of the container in which you want the new directory object to be created by this cmdlet.

PasswordComplexityEnabled

Specify either 'true' or 'false' to determine the password complexity status for user accounts (set the 'msDS-PasswordComplexityEnabled' attribute to this parameter value).

PasswordHistoryLength

Specify the Password History Length setting for user accounts (set the 'msDS-PasswordHistoryLength' attribute to this parameter value).

Precedence

Specify the password settings precedence for the Password Settings object (set the 'msDS-PasswordSettingsPrecedence' attribute to this parameter value).

ResetLockoutCounterAfter

Specify the Observation Window setting for lockout of user accounts (set the 'msDS-LockoutObservationWindow' attribute to this parameter value).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of minutes). This must be a negative value (see examples).

ReversibleEncryptionEnabled

Specify either 'true' or 'false' to determine the password reversible encryption status for user accounts (set the 'msDS-PasswordReversibleEncryptionEnabled' attribute to this parameter value).

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to create a Password Settings object (PSO) and set attribute values in the newly created object.

This cmdlet takes a series of attribute-specific parameters allowing you to set attributes in the newly created Password Settings object. If a particular attribute is referred to by both the ObjectAttributes array and an attribute-specific parameter, the ObjectAttributes setting has no effect on that attribute. The cmdlet sets the attribute to the value specified by the attribute-specific parameter.

Examples

Example 1

Create a new PSO named myPso1 with LockoutDuration of 40 min, Precedence of 10 and MaximumPasswordAge of 45 days, 3 hours and 23 minutes and default values for the other parameters, and apply it to two groups, and display operation results:

```
C:\PS> New-QADPasswordSettingsObject -Name 'myPso1' -LockoutDuration 40 -Precedence 10  
-MaximumPasswordAge (new-timespan -days -45 -hour -3 -minute -23) -AppliesTo  
'myDomain\Account Operators','myDomain\Event Log Readers' | Format-List
```

Get-QADPasswordSettingsObject

Retrieve Password Settings objects that match the specified conditions. Active Directory version of Windows Server 2008 or later is required.

Syntax

```
Get-QADPasswordSettingsObject [[-Identity] <IdentityParameter>] [-Precedence <Int32[]>]
[-ReversibleEncryptionEnabled] [-PasswordHistoryLength <Int32[]>]
[-PasswordComplexityEnabled] [-MinimumPasswordLength <Int32[]>] [-MinimumPasswordAge
<TimeSpanAndDaysParameter[]>] [-MaximumPasswordAge <TimeSpanAndDaysParameter[]>]
[-LockoutThreshold <Int32[]>] [-ResetLockoutCounterAfter
<TimeSpanAndMinutesParameter[]>] [-LockoutDuration <TimeSpanAndMinutesParameter[]>]
[-MemberOf <IdentityParameter[]>] [-IndirectMemberOf <IdentityParameter[]>]
[-NotMemberOf <IdentityParameter[]>] [-NotIndirectMemberOf <IdentityParameter[]>]
[-Tombstone] [-Recycled] [-LastKnownParent <IdentityParameter>] [-SecurityMask
<SecurityMasks>] [-SearchRoot <IdentityParameter[]>] [-SearchScope <SearchScope>]
[-AttributeScopeQuery <String>] [-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter
<String>] [-WildcardMode <WildcardMode>] [-SearchAttributes <Object>] [-Description
<String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr <String>] [-Control
<Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore
<DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>]
[-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, 'msDS-PSOAppliesTo'). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to 'msDS-PSOAppliesTo', the cmdlet searches the collection of the users and security groups to which the SearchRoot object is applied.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

IndirectMemberOf

Retrieve objects that belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has direct or indirect membership in the group specified by this parameter value.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LastKnownParent

When searching for a deleted object by using the Tombstone parameter, specify the DN of the container the object was in before it became a tombstone. This allows you to find objects that were deleted from a particular container.

Note that the lastKnownParent attribute is only set if the object was deleted on a domain controller running Windows Server 2003 or later version of Microsoft Windows Server. Therefore, it is possible that the lastKnownParent attribute value is inaccurate.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

LockoutDuration

Specify the Lockout Duration setting of Password Settings objects to find (search by the 'msDS-LockoutDuration' attribute).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of minutes). This must be a negative value.

LockoutThreshold

Specify the Lockout Threshold setting of Password Settings objects to find (search by the 'msDS-LockoutThreshold' attribute).

MaximumPasswordAge

Specify the Maximum Password Age setting of the objects to find (search by the 'msDS-MaximumPasswordAge' attribute).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of days). This must be a negative value.

MemberOf

Retrieve objects that are direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object has direct membership in the group specified by this parameter value.

MinimumPasswordAge

Specify the Minimum Password Age setting of Password Settings objects to find (search by the 'msDS-MinimumPasswordAge' attribute).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of days). This must be a negative value.

MinimumPasswordLength

Specify the Minimum Password Length setting of Password Settings objects to find (search by the 'msDS-MinimumPasswordLength' attribute).

Name

Search by the 'name' attribute.

NotIndirectMemberOf

Retrieve objects that do not belong to the group or groups specified by this parameter, whether directly or because of group nesting. The cmdlet returns an object if the object has neither direct nor indirect membership in the group specified by this parameter value.

NotMemberOf

Retrieve objects that are not direct members of the group or groups specified by this parameter. The cmdlet returns an object if the object does not have direct membership in the group specified by this parameter value.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

PasswordComplexityEnabled

Specify either 'true' or 'false' to find Password Settings objects that enable or disable the password complexity requirements for user accounts (search by the 'msDS-PasswordComplexityEnabled' attribute).

PasswordHistoryLength

Specify the Password History Length setting of Password Settings objects to find (search by the 'msDS-PasswordHistoryLength' attribute).

Precedence

Specify the password settings precedence of Password Settings objects to find (search by the 'msDS-PasswordSettingsPrecedence' attribute).

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

Recycled

This parameter has an effect only if all of the following conditions are true:

- A domain is supplied as the SearchRoot parameter value.
- Active Directory Recycle Bin is enabled in that domain.

You can use this parameter in conjunction with the Tombstone parameter for the search results to include both the deleted and recycled objects that meet the search conditions. Without this parameter, the cmdlet returns only deleted objects.

ResetLockoutCounterAfter

Specify the Observation Window setting of Password Settings objects to find (search by the 'msDS-LockoutObservationWindow' attribute).

Parameter value can be represented as any of the following: Int64, IADsLargeInteger, DateTime, TimeSpan, string (a string representation of Int64, DateTime or TimeSpan), or Int (a number of minutes). This must be a negative value.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

ReversibleEncryptionEnabled

Specify either 'true' or 'false' to find Password Settings objects that enable or disable password reversible encryption for user accounts (search by the 'msDS-PasswordReversibleEncryptionEnabled' attribute).

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SecurityMask

Specify which elements of the object's security descriptor to retrieve. Valid parameter values are:

- 'None' - do not retrieve any security data
- 'Owner' - retrieve the owner data
- 'Group' - retrieve the primary group data
- 'Dacl' - retrieve the discretionary access-control list data
- 'Sacl' - retrieve the system access-control list data

You can supply a combination of these values, separating them by commas. For example, you can supply the parameter value of 'Dacl,Sacl' in order to retrieve both the discretionary and system access-control list data.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Tombstone

Search for deleted objects of the respective object class. The search output is normally intended to be passed (piped in) to the Restore-QADDeletedObject cmdlet for restoring deleted objects.

In a domain with Active Directory Recycle Bin (a feature of Windows Server 2008 R2) this parameter retrieves deleted objects (rather than tombstones, which in that case are referred to as recycled objects). Recycle Bin preserves all attributes on the deleted objects, so you can use a search filter based on any attributes.

In a domain without Active Directory Recycle Bin, deleting an object converts that object to a tombstone. A search using this parameter returns tombstone objects that meet the filtering criteria supplied. Upon deletion of an object only a small number of the object's attributes are saved in the tombstone, with most of the attributes being lost. To search for deleted objects, your search filter should be based on the attributes that are preserved in tombstones.

When the Tombstone parameter is supplied, the search results include the deleted objects or tombstones that match the specified search filter. However, a search filter that matches a live object may not work as expected after the object is deleted. This is because not all attributes are retained in the tombstone. For example, a filter such as `(&(objectClass=user)(objectCategory=person))` would not match any tombstone objects since the `objectCategory` attribute is removed upon object deletion. Conversely, the `objectClass` attribute is retained on tombstone objects, so a filter of `(objectClass=user)` would match deleted user objects.

The name of a tombstone object begins with the name of the deleted object, so a search using the Tombstone parameter can be refined by adding a filter based on object name. For example, to search for deleted objects with a name that begins with "John", you can use a filter such as `(cn=John*)`.

It is also possible to find a specific deleted object. If you know the name of the object and the Distinguished Name (DN) of the container the object was in before it was deleted, then you can pass the container's DN to the `LastKnownParent` parameter and apply a filter of `(cn=<name of the object>*)` in order to have the cmdlet retrieve that specific object. However, if an object is deleted, a new object with the same DN is created, and then deleted as well, the above search would return more than one object. The returned objects are distinguished by the GUIDs of the deleted objects, with the name of each ending in the GUID of the respective deleted object.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the `Get-QADPSSnapinSettings` cmdlet, respectively. Normally, this parameter is used in conjunction with `IncludeAllProperties` to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to search an Active Directory domain or container for Password Settings objects that meet certain search criteria, or to bind to a certain Password Settings object by DN or GUID. You can search by object attributes or specify your search criteria by using an LDAP search filter.

The output of the cmdlet is a collection of objects, with each object representing one of the objects found by the cmdlet. You can pipe the output into another cmdlet, such as **Set-QADObject**, to make changes to the Password Settings objects returned by this cmdlet.

The cmdlet takes a series of attribute-specific parameters allowing you to search by object attributes. The attribute-specific parameters have effect if SearchRoot is specified whereas Identity is not. If you specify SearchRoot only, then the cmdlet returns all Password Settings objects found in the SearchRoot container.

You can use attribute-specific parameters to search for objects that have specific values of certain attributes. Thus, to find all Password Settings objects that have the password settings precedence set to 1, you may add the following on the command line: "-Precedence 1". To search for Password Settings objects that have a certain attribute not set specify "" (empty string) as the parameter value.

If a particular attribute is referred to by both the SearchAttributes array and an attribute-specific parameter, the SearchAttributes setting has no effect on that attribute. The cmdlet searches for the attribute value specified by the attribute-specific parameter.

With more than one attribute-specific parameter supplied, the search conditions are combined by using the AND operator, so as to find the objects that meet all the specified conditions.

Examples

Example 1

Find a Password Settings object by name, and display properties of the object found:

```
C:\PS> Get-QADPasswordSettingsObject -Name 'myPsol' | Format-List
```

Example 2

Find all Password Settings objects in your domain, and, for each object found, list the users and groups that the object is applied to:

```
C:\PS> Get-QADPasswordSettingsObject -IncludedProperties 'msDS-PSOAppliesTo' |  
Format-Table Name, 'msDS-PSOAppliesTo'
```

Example 3

Query on the 'msDS-PSOApplied' attribute to retrieve and display the distinguished names of the Password Settings objects that are (explicitly) applied to the user object specified:

```
C:\PS> Get-QADUser 'john smith' -DontUseDefaultIncludedProperties -IncludedProperties  
'msDS-PSOApplied' | Format-Table 'msDS-PSOApplied'
```

Example 4

Query on the 'msDS-PSOApplied' attribute to retrieve the distinguished names of the Password Settings objects that are (explicitly) applied to the user object specified, and store the names in a variable named \$psos:

```
C:\PS> $psos = (Get-QADUser 'john smith'  
-DontUseDefaultIncludedProperties -IncludedProperties  
'msDS-PSOApplied').'msDS-PSOApplied'
```

Example 5

Query on the 'msDS-ResultantPso' attribute to retrieve the distinguished name of the PSO that ultimately applies to the user specified (based on the RSoP calculation rules). If there is no PSO that applies to the user, either directly or by virtue of group membership, the query returns NULL:

```
C:\PS> Get-QADUser 'john smith' -DontUseDefaultIncludedProperties -IncludedProperties  
'msDS-ResultantPso' | Format-Table 'msDS-ResultantPso'
```

Get-QADPasswordSettingsObjectAppliesTo

Retrieve objects to which a particular Password Settings object is applied. Active Directory version of Windows Server 2008 or later is required.

Syntax

```
Get-QADPasswordSettingsObjectAppliesTo [-Identity] <IdentityParameter> [-Type <String>]
[-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode
<WildcardMode>] [-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName
<String[]>] [-Name <String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn
<DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn
<DateTime>] [-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>]
[-IncludeAllProperties] [-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (*displayName*)
- Given-Name (*givenName*)
- Legacy-Exchange-DN (*legacyExchangeDN*)
- ms-DS-Additional-Sam-Account-Name (*msDS-AdditionalSamAccountName*)
- Physical-Delivery-Office-Name (*physicalDeliveryOfficeName*)
- Proxy-Addresses (*proxyAddresses*)

- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values “as is,” without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the `LdapFilter` parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the `Get-QADPSSnapinSettings` cmdlet, respectively.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the `Set-QADProgressPolicy` cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the `IncludeAllProperties` parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the `Get-QADUser` or `Get-QADObject` cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Type

Specify the type of directory objects to find. The cmdlet searches for objects that have one of the 'objectClass' attribute values set to the Type parameter value.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to retrieve users or groups to which a particular Password Settings object is applied. The cmdlet searches the 'msDS-PSOAppliesTo' attribute of the Password Settings object (supplied through the Identity parameter) for objects that meet the search conditions specified.

The output of the cmdlet is a collection of objects, with each object representing one of the directory objects found by the cmdlet. You can pipe the output into another cmdlet, such as Set-QADObject, to make changes to the directory objects found by this cmdlet.

Examples

Example 1

Retrieve all groups affected by the Password Settings object named MyPSO (the groups to which the Password Settings object applies):

```
C:\PS> Get-QADPasswordSettingsObjectAppliesTo 'MyPSO' -Type 'Group'
```

Example 2

Retrieve users and groups affected by the MyPSO Password Settings object that contain the word Administration somewhere in the Description attribute:

```
C:\PS> Get-QADPasswordSettingsObjectAppliesTo 'MyPSO' -Description '*Administration*'
```

Add-QADPasswordSettingsObjectAppliesTo

Add PSO links on a Password Settings object. Active Directory version of Windows Server 2008 or later is required.

Syntax

```
Add-QADPasswordSettingsObjectAppliesTo [-Identity] <IdentityParameter> [-AppliesTo]
<IdentityParameter[]> [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service
<String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

AppliesTo

Specify a list of users or groups to which you want the Password Settings object to apply. Each list entry is the DN, SID, GUID, UPN or Domain\Name of a user or group. Separate the list entries by commas.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to apply a Password Settings object to users or global security groups. You can specify a list of users and groups, separating the list entries by commas. The cmdlet adds (appends) the specified distinguished names of the users or groups to the 'msDS-PSOAppliesTo' attribute of the Password Settings object, without removing the names that already exist in the attribute.

Examples

Example 1

Apply the Password Settings object to the user object, and display operation results:

```
C:\PS> Add-QADPasswordSettingsObjectAppliesTo 'myPsol' -AppliesTo 'JSmith' | Format-List
```

Example 2

Find a Password Settings object by name and add a PSO link that points to a particular user object (so the Password Settings object applies to that user):

```
C:\PS> Get-QADPasswordSettingsObject -Name 'myPsol' |  
Add-QADPasswordSettingsObjectAppliesTo -AppliesTo 'JSmith'
```

Example 3

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
Add-QADPasswordSettingsObjectAppliesTo 'myPsol' -AppliesTo 'JSmith'  
$op = get-QARSLastOperation  
If($op.OperationStatus -eq 'Pending')  
{  
    # Operation submitted for approval. Handle this case here.  
}  
elseif($op.OperationStatus -eq 'Completed')  
{  
    # No approval required. Operation completed.  
}  
else  
{  
    # Operation not completed for some reason. You might check other OperationStatus  
    # values to determine the status of the operation request.  
}
```

Remove-QADPasswordSettingsObjectAppliesTo

Remove PSO links on a Password Settings object. Active Directory version of Windows Server 2008 or later is required.

Syntax

```
Remove-QADPasswordSettingsObjectAppliesTo [-Identity] <IdentityParameter> [-AppliesTo]
<IdentityParameter[]> [-Control<Hashtable>] [-Proxy] [-UseGlobalCatalog] [-Service
<String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

AppliesTo

Specify a list of users or groups that you want the Password Settings object to no longer apply to. Each list entry is the DN, SID, GUID, UPN or Domain\Name of a user or group. Separate the list entries by commas.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove PSO links on a Password Settings object in order to have the Password Settings object no longer apply to certain users or groups. You can specify a list of users and groups, separating the list entries by commas. The cmdlet removes the specified distinguished names of the users or groups from the 'msDS-PSOAppliesTo' attribute of the Password Settings object.

Examples

Example 1

Find a Password Settings object by name, remove a PSO link that points to the group (so the Password Settings object no longer applies to that group), and display operation results:

```
C:\PS> Get-QADPasswordSettingsObject -Name 'myPsol' |  
Remove-QADPasswordSettingsObjectAppliesTo -AppliesTo 'myDomain\Account Operators' |  
Format-List
```

Get-QADRootDSE

Retrieve the rootDSE object from the current directory server (domain controller).

Syntax

```
Get-QADRootDSE [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection <ArsConnection>]
```

Parameters

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Detailed Description

This cmdlet returns the rootDSE object containing data about the directory server. The rootDSE object is retrieved from a domain controller that is specific to the current connection. Thus, if connection parameters are supplied to choose a certain domain controller, the cmdlet retrieves the rootDSE object from that domain controller.

You can use the rootDSE object to get distinguished names of the domain, schema, and configuration containers, and other data about the server and the contents of its directory data tree. For information about attributes supported by rootDSE, refer to the "RootDSE" topic in the Active Directory Schema documentation in the MSDN Library (<http://msdn.microsoft.com>).

When connected to Quest One ActiveRoles, the cmdlet retrieves the rootDSE object containing information about the Quest One ActiveRoles namespaces. For information about attributes supported by the Quest One ActiveRoles rootDSE, refer to the Quest One ActiveRoles SDK and Resource Kit documentation (this documentation is normally installed with the Quest One ActiveRoles Administration Service).

Examples

Example 1

List the distinguished names of the domain, schema, and configuration containers for the current connection (this command retrieves and displays the values of the following attributes from rootDSE: defaultNamingContext, schemaNamingContext, and configurationNamingContext):

```
C:\PS> get-QADRootDSE | Format-List {$_["defaultNamingContext"]},  
{$_["schemaNamingContext"]}, {$_["configurationNamingContext"]}
```

Example 2

Identify the domain controller that is used by the current connection. The output of this command is the distinguished name of the server object for that domain controller in the configuration container (the command displays the value of the serverName attribute retrieved from rootDSE):

```
C:\PS> (get-QADRootDSE)["serverName"]
```

Example 3

Connect to any available Quest One ActiveRoles Administration Service and then retrieve the fully qualified domain name of the computer running the Administration Service to which you have connected:

```
C:\PS> connect-QADService -proxy  
C:\PS> (get-QADRootDSE)["edsvaServiceFullDns"]
```

Get-QADPermission

Retrieve access control entries (ACEs) that meet the conditions you want. Every object returned by this cmdlet represents an access control entry (ACE) in the discretionary access control list (DACL) of a certain directory object.

Syntax

```
Get-QADPermission [-Identity] <IdentityParameter> [-Inherited] [-SchemaDefault]
[-UseTokenGroups] [-UseExtendedMatch] [-Allow] [-Deny] [-ApplyTo
<ArsSecurityInheritance[]>] [-Rights <ActiveDirectoryRights>] [-Account
<IdentityParameter[]>] [-Property <String[]>] [-PropertySet <String[]>] [-ExtendedRight
<String[]>] [-ValidatedWrite <String[]>] [-ChildType <String[]>] [-ApplyToType
<String[]>] [-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount
<String>] [-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Account

Supply the identity (such as name, Distinguished Name, Domain\Name, SID, etc.) of a security principal (user, group, computer account, etc.). The cmdlet will retrieve ACEs that determine access rights of that security principal on the directory object specified. You can supply identities of multiple accounts.

Allow

Retrieve ACEs that allow access to the directory object specified.

ApplyTo

Depending on the cmdlet you use, this parameter lets you either retrieve or add ACEs that have a particular inheritance type set. Valid parameter values are:

- 'ThisObjectOnly' - Indicates no inheritance. The ACE information is only used on the object on which the ACE is set. ACE information is not inherited by any descendents of the object.
- 'All' - Indicates inheritance that includes the object on which the ACE is set, the object's immediate children, and the descendents of the object's children.
- 'ChildObjects' - Indicates inheritance that includes the object's immediate children and the descendants of the object's children, but not the object itself.

- 'ThisObjectAndImmediateChildObjects' - Indicates inheritance that includes the object itself and its immediate children. It does not include the descendents of its children.
- 'ImmediateChildObjectsOnly' - Indicates inheritance that includes the object's immediate children only, not the object itself or the descendents of its children.

ApplyToType

Retrieve ACEs that can be inherited by objects of a particular class. Property value is the LDAP display name of the classSchema object for the object class you want. (This parameter causes the cmdlet to search by the InheritedObjectType setting on the ACEs.)

You can specify multiple classes, separating the name of the classes by commas. If you do so, the cmdlet retrieves ACEs that can be inherited by objects of any of the classes specified.

ChildType

Retrieve ACEs that control the right to create or delete child objects of a particular class. Parameter value is the LDAP display name of the classSchema object for the child object's class. (This parameter causes the cmdlet to search by the ObjectType setting on the ACEs.)

You can specify multiple classes, separating the names of the classes by commas. If you do so, the cmdlet retrieves ACEs that control the right to create or delete child objects of any of the classes specified.

Deny

Retrieve ACEs that deny access to the directory object specified.

ExtendedRight

Retrieve ACEs that determine the specified extended rights on the directory object. Parameter value is a string array of the names of the extended rights you want. For a list of possible extended rights, see topic "Extended Rights" in the MSDN Library at <http://msdn.microsoft.com>. For more information about extended rights, see topic "Control Access Rights" in the MSDN Library.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to find.

The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

Inherited

Retrieve ACEs that come from security descriptors of the ancestors of the directory object (ACEs that are inherited from the parent container object).

Property

Retrieve ACEs that determine access to the specified attributes of the directory object. Parameter value is a string array of the LDAP display names of the attributes you want.

PropertySet

Retrieve ACEs that determine access to the specified property sets of the directory object. Specify the names of the property sets you want, separating names by commas. For a list of possible property sets, see topic "Property Sets" in the MSDN Library at <http://msdn.microsoft.com>.

Rights

Depending on the cmdlet you use, this parameter lets you either retrieve or add ACEs that have particular access rights set. Valid parameter values are as follows (for descriptions of these access rights see topic "ActiveDirectoryRights Enumeration" in the MSDN Library at <http://msdn.microsoft.com>):

- 'Delete'
- 'ReadControl'
- 'WriteDacl'
- 'WriteOwner'
- 'Synchronize'
- 'AccessSystemSecurity'
- 'GenericRead'
- 'GenericWrite'
- 'GenericExecute'
- 'GenericAll'
- 'CreateChild'
- 'DeleteChild'
- 'ListChildren'
- 'Self'
- 'ReadProperty'
- 'WriteProperty'
- 'DeleteTree'
- 'ListObject'
- 'ExtendedRight'

Parameter value can be any combination of the listed values, separated by commas. For example, the parameter value of 'ReadProperty,WriteProperty' allows you to retrieve or add ACEs that have both the ReadProperty and WriteProperty access rights set.

SchemaDefault

Retrieve ACEs that came from the default security descriptor defined in the classSchema object for the directory object's class.

UseExtendedMatch

Retrieve not only ACEs with the specified access rights setting but also ACEs with other access rights settings that effectively give the same level of access as the rights setting specified.

For example, the `-Rights 'ReadProperty'` parameter alone causes the cmdlet to retrieve only ACEs that have the `ReadProperty` access right set, whereas the combination of parameters such as `-Rights 'ReadProperty' -UseExtendedMatch` also retrieves ACEs that have the `GenericRead` or `GenericAll` access right set.

UseTokenGroups

Retrieve ACEs that apply not only to the specified security principal (SID) itself but also to any of the groups to which the account belongs whether directly or because of group nesting.

ValidatedWrite

Retrieve ACEs that determine the specified validated writes on the directory object. Parameter value is a string array of the names of the validated writes you want. For a list of possible validated writes, see topic "Validated Writes" in the MSDN Library at <http://msdn.microsoft.com>.

Detailed Description

Use this cmdlet to retrieve access control entries (ACEs) from the discretionary access control list (DACL) of a particular object or objects in the directory (directory objects).

The directory objects can be specified using the *Identity* parameter. Another option is to use pipelining: pass the output of the appropriate Get-QAD cmdlet to this cmdlet, with the *-SecurityMask Dacl* parameter supplied for the Get- cmdlet.

The cmdlet returns the objects representing the ACEs that meet the conditions you define using parameters of the cmdlet. You can use pipelining to pass the objects returned by this cmdlet to another cmdlet. For example, you can pass them to the `Remove-QADPermission` cmdlet in order to delete the respective ACEs from the DACL.

Examples

Example 1

Retrieve the ACEs that are explicitly set on a particular object (the ACEs that are neither inherited from the parent container nor received from the default security descriptor of the respective classSchema object):

```
C:\PS> Get-QADObject 'DistinguishedNameOfTheObject' -SecurityMask Dacl |
Get-QADPermission
```

Example 2

Retrieve all ACEs from the DACL of a particular object (including the ACEs that are inherited from the parent container or received from the default security descriptor of the respective classSchema object):

```
C:\PS> Get-QADObject 'DistinguishedNameOfTheObject' -SecurityMask Dacl |
Get-QADPermission -Inherited -SchemaDefault
```

Example 3

Retrieve the ACEs on a particular object that have any of the specified groups set as the trustee:

```
C:\PS> Get-QADObject 'DistinguishedNameOfTheObject' -SecurityMask Dacl |
Get-QADPermission -Account ('domainName\groupName1','domainName\groupName2')
```

Example 4

Retrieve the ACEs on a particular object that have the trustee set either to the specified user account or to any of the groups to which the user account belongs (whether directly or because of group nesting):

```
C:\PS> Get-QADObject 'DistinguishedNameOfTheObject' -SecurityMask Dacl |  
Get-QADPermission -Account 'domain\user' -UseTokenGroups
```

Example 5

Retrieve the ACEs on a particular object that determine Read access to properties of the object:

```
C:\PS> Get-QADObject 'DistinguishedNameOfTheObject' -SecurityMask Dacl |  
Get-QADPermission -Rights 'ReadProperty'
```

Example 6

Retrieve the ACEs on a particular user account that are configured with the WriteProperty access right for the 'sAMAccountName' or 'name' property:

```
C:\PS> Get-QADUser 'domain\user' -SecurityMask Dacl | Get-QADPermission -Rights  
'WriteProperty' -Property ('sAMAccountName','name')
```

Example 7

Retrieve all the ACEs that allow write access to the 'sAMAccountName' or 'name' property of a particular user account:

```
C:\PS> Get-QADUser 'domain\user' -SecurityMask Dacl | Get-QADPermission -Rights  
'WriteProperty' -UseExtendedMatch -Inherited -SchemaDefault -Allow -Property  
( 'sAMAccountName', 'name' )
```

Example 8

Copy the ACEs that are configured on a particular directory object (not including the inherited ACEs or the schema default ACEs) to another directory object:

```
C:\PS> Get-QADPermission 'DistinguishedNameOfSourceObject' | Add-QADPermission  
'DistinguishedNameOfDestinationObject'
```

Example 9

Delete all the deny-type ACEs that are configured on a particular directory object (not including the inherited ACEs or the schema default ACEs):

```
C:\PS> Get-QADPermission 'DistinguishedNameOfObject' -Deny | Remove-QADPermission
```

Add-QADPermission

Add access control entries (ACEs) to the discretionary access control list (DACL) of a certain directory object or objects.

Syntax

```
Add-QADPermission [-Identity] <IdentityParameter> -Account <IdentityParameter[]>
[-Rights <ActiveDirectoryRights>] [-Deny] [-ApplyTo <ArsSecurityInheritance>] [-Property
<String[]>] [-PropertySet <String[]>] [-ExtendedRight <String[]>] [-ValidatedWrite
<String[]>] [-ChildType <String[]>] [-ApplyToType <String[]>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

```
Add-QADPermission [-Identity] <IdentityParameter> -InputPermission <ArsPermission>
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Account

Supply the identity (such as name, Distinguished Name, Domain\Name, SID, etc.) of a security principal (user, group, computer account, etc.). The cmdlet will add ACEs that determine access rights of that security principal on the directory object specified. You can supply identities of multiple accounts.

ApplyTo

Depending on the cmdlet you use, this parameter lets you either retrieve or add ACEs that have a particular inheritance type set. Valid parameter values are:

- 'ThisObjectOnly' - Indicates no inheritance. The ACE information is only used on the object on which the ACE is set. ACE information is not inherited by any descendents of the object.
- 'All' - Indicates inheritance that includes the object on which the ACE is set, the object's immediate children, and the descendents of the object's children.
- 'ChildObjects' - Indicates inheritance that includes the object's immediate children and the descendants of the object's children, but not the object itself.

- 'ThisObjectAndImmediateChildObjects' - Indicates inheritance that includes the object itself and its immediate children. It does not include the descendents of its children.
- 'ImmediateChildObjectsOnly' - Indicates inheritance that includes the object's immediate children only, not the object itself or the descendents of its children.

ApplyToType

Supply this parameter for the cmdlet to add ACEs that can be inherited by objects of specific classes (rather than all classes). Parameter value is a string array of LDAP display names, each of which identifies the classSchema object for the object class you want.

If you do not supply this parameter, the cmdlet adds ACEs that can be inherited by objects of any class. (This parameter causes the cmdlet to configure the InheritedObjectType setting on the ACEs.)

ChildType

Supply this parameter for the cmdlet to add ACEs that control the right to create or delete child objects of particular classes (rather than all classes). Parameter value is a string array of LDAP display names, each of which identifies the classSchema object for a child object's class you want.

If you do not supply this parameter, the cmdlet configures ACEs to control the right to create or delete child objects of any class. (This parameter causes the cmdlet to configure the ObjectType setting on the ACEs.)

Confirm

Prompts you for confirmation before executing the command.

Deny

Supply this parameter for the cmdlet to add ACEs that deny (rather than allow) access. If you do not supply this parameter, the cmdlet configures ACEs to allow access.

ExtendedRight

Supply this parameter for the cmdlet to add ACEs that determine specific extended rights on the directory object. Parameter value is a string array of the names of the extended rights you want. For a list of possible extended rights, see topic "Extended Rights" in the MSDN Library at <http://msdn.microsoft.com>. For more information about extended rights, see topic "Control Access Rights" in the MSDN Library.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

InputPermission

This parameter is used to identify the object or objects representing ACEs to add. The parameter accepts parameter values from the pipeline. Thus, when you use pipelining to pass to this cmdlet the objects returned by the Get-QADPermission cmdlet, you should not supply this parameter on the command line. Another option is to save the object in a variable and then supply that variable as a parameter value.

Property

Supply this parameter for the cmdlet to add ACEs that determine access to specific attributes of the directory object (rather than all attributes). Parameter value is a string array of the LDAP display names of the attributes you want.

If you supply neither this parameter nor the PropertySet parameter, the cmdlet configures ACEs to determine access to all attributes.

PropertySet

Supply this parameter if want the cmdlet to add ACEs that determine access to specific property sets of the directory object (rather than all attributes). Parameter value is a string array of the names of the property sets you want. For a list of possible property sets, see topic "Property Sets" in the MSDN Library at <http://msdn.microsoft.com>.

If you supply neither this parameter nor the Property parameter, the cmdlet configures ACEs to determine access to all attributes.

Rights

Depending on the cmdlet you use, this parameter lets you either retrieve or add ACEs that have particular access rights set. Valid parameter values are as follows (for descriptions of these access rights see topic "ActiveDirectoryRights Enumeration" in the MSDN Library at <http://msdn.microsoft.com>):

- 'Delete'
- 'ReadControl'
- 'WriteDacl'
- 'WriteOwner'
- 'Synchronize'
- 'AccessSystemSecurity'
- 'GenericRead'
- 'GenericWrite'
- 'GenericExecute'
- 'GenericAll'
- 'CreateChild'
- 'DeleteChild'
- 'ListChildren'
- 'Self'
- 'ReadProperty'
- 'WriteProperty'
- 'DeleteTree'
- 'ListObject'
- 'ExtendedRight'

Parameter value can be any combination of the listed values, separated by commas. For example, the parameter value of 'ReadProperty,WriteProperty' allows you to retrieve or add ACEs that have both the ReadProperty and WriteProperty access rights set.

ValidatedWrite

Supply this parameter for the cmdlet to add ACEs that determine validated writes on the directory object. Parameter value is a string array of the names of the validated writes you want. For a list of possible validated writes, see topic "Validated Writes" in the MSDN Library at <http://msdn.microsoft.com>.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add access control entries (ACEs) to the discretionary access control list (DACL) of a particular object or objects in the directory (directory objects).

The directory objects can be specified using the *Identity* parameter. Another option is to use pipelining: pass the output of the appropriate Get-QAD cmdlet to this cmdlet, with the *-SecurityMask Dacl* parameter supplied for the Get- cmdlet (see examples).

The objects representing ACEs to add can be either passed to this cmdlet through the pipeline or created by the cmdlet itself. In the latter case you use cmdlet parameters to configure the ACEs that you want the cmdlet to add. If you opt to use pipelining, you can have Get-QADPermission retrieve ACEs and then pass the output of that cmdlet to the Add-QADPermission cmdlet so as to copy certain ACEs from one directory object to another directory object (see examples).

Examples

Example 1

Give a certain group full access to a certain organizational unit (OU) and all objects in that OU:

```
C:\PS> Add-QADPermission 'DistinguishedNameOfTheOU' -Account 'domainName\groupName'
-Rights 'GenericAll'
```

Example 2

Deny a certain group permission to modify the sAMAccountName property as well as the properties that are part of the General Information or Web Information property set on a certain user account:

```
C:\PS> Add-QADPermission 'domainName\userName' -Deny -Account 'domainName\groupName'
-Rights 'WriteProperty' -PropertySet ('General-Information','Web-Information')
-Property 'sAMAccountName' -ApplyTo 'ThisObjectOnly'
```

Example 3

Authorize a given group to create user accounts in a particular organizational unit (OU) or in organizational units that are (immediate) children of that OU:

```
C:\PS> Add-QADPermission 'DistinguishedNameOfTheOU' -Account 'domainName\groupName'
-Rights 'CreateChild' -ChildType 'user' -ApplyTo 'ThisObjectAndImmediateChildObjects'
-ApplyToType 'organizationalUnit'
```

Example 4

Authorize a given group to view or modify the group membership list of any group in a particular organizational unit (OU):

```
C:\PS> Add-QADPermission 'DistinguishedNameOfTheOU' -Account 'domainName\groupName'
-Rights 'ReadProperty,WriteProperty' -Property 'member' -ApplyToType 'group'
```

Example 5

Deny a given user account permission to modify the group membership list of any group in a particular organizational unit (OU):

```
C:\PS> Get-QADGroup -SearchRoot 'DistinguishedNameOfTheOU' -SecurityMask 'Dacl' |
Add-QADPermission -Account 'domainName\UserName' -Deny -Rights 'WriteProperty' -Property
'member'
```

Example 6

Authorize a given group to view or modify any property that is part of the Personal Information property set on any user account in a particular organizational unit (OU):

```
C:\PS> Add-QADPermission 'DistinguishedNameOfTheOU' -Account 'domainName\groupName'
-Rights 'ReadProperty,WriteProperty' -PropertySet 'Personal-Information' -ApplyTo
'ChildObjects' -ApplyToType 'user'
```

Example 7

Copy the ACEs that are configured on a particular directory object (not including the inherited or schema default ACEs) to another directory object:

```
C:\PS> Get-QADPermission 'DistinguishedNameOfSourceObject' | Add-QADPermission
'DistinguishedNameOfDestinationObject'
```

Remove-QADPermission

Delete access control entries (ACEs) from the discretionary access control list (DACL) of a directory object or objects.

Syntax

```
Remove-QADPermission [-InputPermission] <ArsPermission> [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

InputPermission

This parameter is used to identify the object or objects representing the ACEs to delete. The parameter accepts parameter values from the pipeline. Thus, when you use pipelining to pass to this cmdlet the objects returned by the Get-QADPermission cmdlet, you should not supply this parameter on the command line (see examples). Another option is to save the object in a variable and then supply that variable as a parameter value.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to delete access control entries (ACEs) from the discretionary access control list (DACL) of an object or objects in the directory (directory objects).

The objects representing ACEs to remove can be passed to this cmdlet through the pipeline. You can have Get-QADPermission retrieve ACEs and then pass the output of that cmdlet to the Add-QADPermission cmdlet so as to delete ACEs from the directory object or objects from which the ACEs have been retrieved (see examples).

Examples

Example 1

Delete all the deny-type ACEs that are configured on a particular directory object (not including the inherited ACEs or the schema default ACEs):

```
C:\PS> Get-QADPermission 'DistinguishedNameOfObject' -Deny | Remove-QADPermission
```

Get-QADObjectSecurity

Retrieve security information, such as the owner information or the security descriptor in a string format, from a directory object or objects.

Syntax

```
Get-QADObjectSecurity [-Identity] <IdentityParameter> -Owner [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>]
```

```
Get-QADObjectSecurity [-Identity] <IdentityParameter> -Sddl [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

Owner

Supply this parameter for the cmdlet to return an object that represents the owner of the directory object specified by the Identity parameter.

Sddl

Supply this parameter for the cmdlet to return the directory object's security descriptor in a string format.

The string format is defined by the security descriptor definition language (SDDL). You can use the string format to store or transmit the security descriptor. For a description of the string format, see topic "Security Descriptor Definition Language" in the MSDN Library at <http://msdn.microsoft.com>.

Detailed Description

Use this cmdlet to retrieve security information from an object in the directory (directory object). Thus, you can get an object representing the owner of a particular directory object. You can also have this cmdlet return the security descriptor of a directory object in a string format defined by the security descriptor definition language (SDDL).

Examples

Example 1

Get the object that represents the owner of a particular group:

```
C:\PS> Get-QADObjectSecurity 'domainName\groupName' -Owner
```

Example 2

For a particular directory object, list the security descriptor in a string format:

```
C:\PS> Get-QADObjectSecurity 'DistinguishedNameOfTheObject' -SDDL
```

Example 3

For every computer object held in the Computers container in domain dom.lab.local, list the distinguished name of the owner of the computer object:

```
C:\PS> Get-QADComputer -SearchRoot 'dom.lab.local/Computers' -SecurityMask 'Owner' |  
ForEach-Object {$computer=$_; Get-QADObjectSecurity $_ -Owner | Select-Object  
@{Name='Computer'; Expression={$computer.DN}}, @{Name='Owner'; Expression={$_.DN}}}
```

Set-QADObjectSecurity

Update security information on a directory object or objects. You can change the owner of an object or change the option that governs protection of an object from the effects of inherited rights.

Syntax

```
Set-QADObjectSecurity [-Identity] <IdentityParameter> -Owner <IdentityParameter>
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

```
Set-QADObjectSecurity [-Identity] <IdentityParameter> -LockInheritance [-Remove]
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

```
Set-QADObjectSecurity [-Identity] <IdentityParameter> -UnlockInheritance [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the DomainName\sAMAccountName, UPN or SID of the object you want the cmdlet to act upon. The cmdlet makes changes to, or otherwise manages, the object identified by this parameter value. When you pipe an object into the cmdlet, this parameter is used to receive the object.

LockInheritance

Supply this parameter for the cmdlet to configure the security descriptor on the directory object so that access control entries (ACEs) that are set on the discretionary access control list (DACL) of the parent container, and any objects above the parent container in the directory hierarchy, are not applied to the DACL of that directory object.

Owner

Specify the identity (name, Distinguished Name, Domain\Name, SID, etc.) of the security principal that you want the cmdlet to set as the owner of the directory object specified by the Identity parameter. Another option is to get an object representing the owner you want, save the object in a variable, and supply that variable as a value for this parameter.

Remove

This parameter can be used in conjunction with the LockInheritance parameter to remove the inherited ACEs from the directory object.

If you supply this parameter, the cmdlet removes the ACEs that were previously applied (inherited) from the parent and keeps only those ACEs that are explicitly defined on the directory object.

If you do not supply this parameter, the cmdlet copies the ACEs that were previously applied from the parent, merging them with the ACEs that are explicitly defined on the directory object.

UnlockInheritance

Supply this parameter for the cmdlet to configure the security descriptor on the directory object so that access control entries (ACEs) originating from the parent container are applied to the DACL of that directory object in accord with the inheritance flags set on those ACEs.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

You can use this cmdlet to perform any of the following tasks on a particular directory object (each of these tasks implies certain changes to the security descriptor of the directory object):

- Set a given security principal to be the owner of that object.
- Specify whether access control entries (ACEs) that are set on the discretionary access control list (DACL) of the parent container, and any objects above the parent container in the directory hierarchy, are applied to the object's DACL.

With the latter task, consider that ACEs can be set on a container object, such as an organizationalUnit, domainDNS, container, and so on, and propagated to child objects based on the inheritance flags set on those ACEs. If you want to explicitly control the ACEs on a certain sensitive object, such as a private OU or a special user, you can prevent ACEs from being propagated to the object by its parent container or its parent container's predecessors.

Examples

Example 1

For a given directory object, set a certain group as the owner of the object:

```
C:\PS> Set-QADObjectSecurity 'DistinguishedNameOfTheObject' -Owner  
'domainName\userName'
```

Example 2

Prevent a certain user account from inheriting ACEs from the parent object and remove the ACEs that were previously applied from the parent object or its ancestors. As a result, access to the user account is controlled by only those ACEs that are explicitly set on the account:

```
C:\PS> Set-QADObjectSecurity 'domainName\userName' -LockInheritance -Remove
```

Example 3

Configure security settings on a particular user account to allow inheritable ACEs from the parent container to propagate to that user account, merging them with those ACEs that are explicitly set on the user account:

```
C:\PS> Set-QADObjectSecurity 'domain\user' -UnlockInheritance
```

Example 4

For every computer object held in a given organizational unit (OU), set the owner of the computer object to the Administrators domain local group:

```
C:\PS> Get-QADComputer -SearchRoot 'DistinguishedNameOfTheOU' -SecurityMask 'Owner' |  
Set-QADObjectSecurity -Owner 'domainName\administrators'
```

Add-QADProxyAddress

Add e-mail addresses for an Exchange recipient.

Syntax

```
Add-QADProxyAddress [-Address] <String> -CurrentAddresses <ProxyAddressChangesUI> [-Type <ProxyAddressType>] [-CustomType <String>] [-Primary] [-WhatIf] [-Confirm]
```

```
Add-QADProxyAddress [-Address] <String> -DirObject <IGenericDirectoryObject> [-Type <ProxyAddressType>] [-CustomType <String>] [-Primary] [-WhatIf] [-Confirm]
```

Parameters

Address

Use this parameter to specify the e-mail address to add, such as 'jsmith@company.com'. You could prepend the address with a prefix to specify the address type, such as 'X400:C=US;A=;O=Exchange;S=Smith;G=John'. If you do not add a prefix, you can specify the address type through the Type or CustomType parameter. By default, the address type is set to SMTP.

Confirm

Prompts you for confirmation before executing the command.

CurrentAddresses

This parameter is intended to receive an output object of an *-QADProxyAddress cmdlet, and is instrumental in a scenario where multiple e-mail address changes need to be applied to a single recipient.

CustomType

Use this parameter to specify an arbitrary custom address type. The e-mail address type is the identifying proxy information for the e-mail address. Exchange uses this information to determine how to process the e-mail address. The e-mail address type cannot exceed 9 characters.

DirObject

This parameter is intended to receive the object that represents the Exchange recipient. This could be an output object of a Get-QAD* cmdlet.

Primary

Supply this parameter to set the e-mail address as the primary or "reply to" address. When the recipient sends an e-mail message, this is the e-mail address that is seen by other users and is the address to which they reply.

A recipient can have multiple e-mail addresses for a specific address type. This allows the recipient to receive messages that are addressed to any one of these e-mail addresses. However, a single address must be designated as the "reply to" address for each address type. An e-mail address that is designated as the "reply to" address is also considered the primary address for the specific address type.

Type

Use this parameter to specify the address type of the e-mail address to add. The possible parameter values are:

- SMTP (this address type is used if the Type and CustomType parameters are omitted)
- X400
- MS (MS Mail address type)
- CCMAIL (ccMail address type)
- MSA (MS Mail address type)
- NOTES (Lotus Notes address type)
- GWISE (Novell GroupWise address type)

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add an e-mail address for an Exchange recipient. You can add an SMTP e-mail address (default), or you can an e-mail address of a custom type, such as an X400, GroupWise, or Lotus Notes address type. It is also possible to specify an arbitrary custom e-mail address type. Since Exchange does not validate custom addresses for proper formatting, you must ensure that the custom address you specify complies with the format requirements for that address type. Because X.400 addresses are considered custom addresses in Exchange, they are also not validated. Therefore, you must provide the correct syntax when specifying an X.400 address.

The cmdlet allows you to identify the Exchange recipient by passing through the pipeline the corresponding output object of a Get-QAD* cmdlet. In this case, the DirObject parameter is used to receive the output object. Another way to identify the Exchange recipient is by passing the output of an *-QADProxyAddress cmdlet, in which case it is the CurrentAddresses parameter that receives the output object (see examples).

Examples

Example 1

Add three e-mail addresses for a particular user mailbox:

```
C:\PS> Get-QADUser company\jsmith | Add-QADProxyAddress -Address  
'smtp:jsmith@company.com' | Add-QADProxyAddress -Type SMTP -Address  
'john.smith@company.com' -Primary | Add-QADProxyAddress -CustomType 'sip' -Address  
'john.smith@company.com'
```

This command adds two SMTP addresses, one of which is set as the primary SMTP address, and a single e-mail address of a custom type. The first instance of Add-QADProxyAddress receives the user object through the DirObject parameter, whereas the other Add-QADProxyAddress instances rely on the CurrentAddresses parameter to receive the object that identifies the recipient.

Example 2

For a particular user mailbox, replace all the existing e-mail addresses with three other e-mail addresses:

```
C:\PS> Get-QADUser company\jsmith | Clear-QADProxyAddress | Add-QADProxyAddress -Address
'smtp:jsmith@company.com' | Add-QADProxyAddress -Type SMTP -Address
'john.smith@company.com' -Primary | Add-QADProxyAddress -CustomType 'sip' -Address
'john.smith@company.com'
```

In this command, `Clear-QADProxyAddress` removes the existing addresses, and then `Add-QADProxyAddress` adds two SMTP addresses, one of which is set as the primary SMTP address, and a single e-mail address of a custom type. `Clear-QADProxyAddress` receives the user object through the `DirObject` parameter, whereas the `Add-QADProxyAddress` instances rely on the `CurrentAddresses` parameter to receive the object that identifies the recipient.

Example 3

For a given user mailbox, replace the e-mail addresses that match a particular pattern with a new primary SMTP address:

```
C:\PS> Get-QADUser company\jsmith | Add-QADProxyAddress -Address
'smtp:john.smith@company.com' -Primary | Remove-QADProxyAddress -Pattern '*@company.com'
```

Example 4

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using `Connect-QADService`). The example submits a request to perform a certain operation, uses `Get-QARSLastOperation` to retrieve the object representing the operation request, and then checks the `OperationStatus` property of that object to see if the requested changes are applied or submitted for approval.

```
Get-QADUser company\jsmith | Add-QADProxyAddress -Address 'smtp:jsmith@company.com'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

Set-QADProxyAddress

Change existing e-mail addresses for an Exchange recipient.

Syntax

```
Set-QADProxyAddress [-From] <String> [[-To] <String>] -CurrentAddresses  
<ProxyAddressChangesUI> [-MakePrimary] [-WhatIf] [-Confirm]
```

```
Set-QADProxyAddress [-From] <String> [[-To] <String>] -DirObject  
<IGenericDirectoryObject> [-MakePrimary] [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

CurrentAddresses

This parameter is intended to receive an output object of an *-QADProxyAddress cmdlet, and is instrumental in a scenario where multiple e-mail address changes need to be applied to a single recipient.

DirObject

This parameter is intended to receive the object that represents the Exchange recipient. This could be an output object of a Get-QAD* cmdlet.

From

Use this parameter to specify the e-mail address or set of e-mail addresses to change. Parameter value may include the asterisk (*) wildcard to match any string of characters. Thus a parameter value of '*@company.com' specifies a set of addresses each of which ends in @company.com.

MakePrimary

Supply this parameter to set each of the e-mail addresses specified by the From parameter, as the primary or "reply to" address. When the recipient sends an e-mail message, this is the e-mail address that is seen by other users and is the address to which they reply.

A recipient can have multiple e-mail addresses for a specific address type. This allows the recipient to receive messages that are addressed to any one of these e-mail addresses. However, a single address must be designated as the "reply to" address for each address type. An e-mail address that is designated as the "reply to" address is also considered the primary address for the specific address type.

To

Use this parameter to specify the new address to which you want to change the address specified by the From parameter. You can use the asterisk (*) wildcard in this parameter provided that the asterisk wildcard is also used in the From parameter. The characters that are represented by the asterisk wildcard in this parameter will be identical to the corresponding characters in the From parameter (see examples).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to change an existing e-mail address, or a set of e-mail addresses, for an Exchange recipient. You can set a particular address as the primary or "reply to" address. It is also possible to edit an address, which effectively replaces the old address with the new one. Multiple addresses can be changed at a time by using the asterisk (*) wildcard to represent any string of characters in an e-mail address. This allows you, for example, to change the address suffix for all addresses at a time by identifying the target set of addresses as '*@OldSuffix' and using a pattern of '*@NewSuffix' to specify the desired changes.

The cmdlet allows you to identify the Exchange recipient by passing through the pipeline the corresponding output object of a Get-QAD* cmdlet. In this case, the DirObject parameter is used to receive the output object. Another way to identify the Exchange recipient is by passing the output of an *-QADProxyAddress cmdlet, in which case it is the CurrentAddresses parameter that receives the output object (see examples for the Add-QADProxyAddress cmdlet).

Examples

Example 1

For all mailbox users, set the e-mail address with a particular suffix as the primary address:

```
C:\PS> Get-QADUser | Set-QADProxyAddress -From '*@source.com' -MakePrimary
```

Example 2

For all mailbox users, change the e-mail addresses with a particular suffix to replace the existing suffix with a new one:

```
C:\PS> Get-QADUser | Set-QADProxyAddress -From '*@before.com' -To '*@after.com'
```

Example 3

The following example shows how to check if an operation performed via ActiveRoles Management Shell requires approval. It is assumed that connection has been established to Quest One ActiveRoles (for instance, by using Connect-QADService). The example submits a request to perform a certain operation, uses Get-QARSLastOperation to retrieve the object representing the operation request, and then checks the OperationStatus property of that object to see if the requested changes are applied or submitted for approval.

```
Get-QADUser company\jsmith | Set-QADProxyAddress -From '*@before.com' -To '*@after.com'
$op = get-QARSLastOperation
If($op.OperationStatus -eq 'Pending')
{
    # Operation submitted for approval. Handle this case here.
}
elseif($op.OperationStatus -eq 'Completed')
{
    # No approval required. Operation completed.
}
else
{
    # Operation not completed for some reason. You might check other OperationStatus
    # values to determine the status of the operation request.
}
```

Remove-QADProxyAddress

Remove e-mail addresses for an Exchange recipient.

Syntax

```
Remove-QADProxyAddress [-Pattern] <String> -CurrentAddresses <ProxyAddressChangesUI> [-WhatIf] [-Confirm]
```

```
Remove-QADProxyAddress [-Pattern] <String> -DirObject <IGenericDirectoryObject> [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

CurrentAddresses

This parameter is intended to receive an output object of an *-QADProxyAddress cmdlet, and is instrumental in a scenario where multiple e-mail address changes need to be applied to a single recipient.

DirObject

This parameter is intended to receive the object that represents the Exchange recipient. This could be an output object of a Get-QAD* cmdlet.

Pattern

Use this parameter to specify the e-mail address or set of e-mail addresses to remove. Parameter value can include the asterisk (*) wildcard to match any string of characters. Thus, a parameter value of '*@company.com' specifies a set of addresses each of which ends in @company.com.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove an e-mail address, or a set of e-mail addresses, for an Exchange recipient. Multiple addresses can be removed at a time by using the asterisk (*) wildcard to represent any string of characters in an e-mail address. This allows you, for example, to remove all e-mail addresses that have a particular suffix, by identifying the target set of addresses as '*@Suffix'.

The cmdlet allows you to identify the Exchange recipient by passing through the pipeline the corresponding output object of a Get-QAD* cmdlet. In this case, the DirObject parameter is used to receive the output object. Another way to identify the Exchange recipient is by passing the output of an *-QADProxyAddress cmdlet, in which case it is the CurrentAddresses parameter that receives the output object (see examples for the Add-QADProxyAddress cmdlet).

Examples

Example 1

For all mailbox users, remove the e-mail addresses that have a particular suffix:

```
C:\PS> Get-QADUser | Remove-QADProxyAddress -Pattern '*@company.com'
```

Clear-QADProxyAddress

Remove all e-mail addresses for an Exchange recipient.

Syntax

```
Clear-QADProxyAddress -CurrentAddresses <ProxyAddressChangesUI> [-WhatIf] [-Confirm]
```

```
Clear-QADProxyAddress -DirObject <IGenericDirectoryObject> [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

CurrentAddresses

This parameter is intended to receive an output object of an *-QADProxyAddress cmdlet, and is instrumental in a scenario where multiple e-mail address changes need to be applied to a single recipient.

DirObject

This parameter is intended to receive the object that represents the Exchange recipient. This could be an output object of a Get-QAD* cmdlet.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove all e-mail addresses for an Exchange recipient. This cmdlet is instrumental in a scenario where you need to replace all the existing addresses. In this case, you could use a pipeline in which Clear-QADProxyAddress removes the existing addresses and then passes the output object through the pipeline for Add-QADProxyAddress to configure new addresses. The cmdlet allows you to identify the Exchange recipient by passing through the pipeline the corresponding output object of a Get-QAD* cmdlet. In this case, the DirObject parameter is used to receive the output object. Another way to identify the Exchange recipient is by passing the output of an *-QADProxyAddress cmdlet, in which case it is the CurrentAddresses parameter that receives the output object (see examples).

Examples

Example 1

Replace all e-mail addresses for a particular user mailbox with a single, primary SMTP address:

```
C:\PS> Get-QADUser company\jsmith | Clear-QADProxyAddress | Add-QADProxyAddress  
'smtp:john.smith@company.com' -Primary
```

In this example, Clear-QADProxyAddress receives the user object through the DirObject parameter, whereas Add-QADProxyAddress relies on the CurrentAddresses parameter to receive the object that identifies the recipient.

Enable-QADEmailAddressPolicy

Enable the e-mail address policy for an Exchange recipient.

Syntax

```
Enable-QADEmailAddressPolicy [-Identity] <IdentityParameter> [-Control <Hashtable>]
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Use this parameter to specify the recipient object. You can supply the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the Domain\Name, UPN or SID of the object. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to configure an Exchange recipient to have the recipient's e-mail addresses automatically updated based on changes made to e-mail address policies in your Exchange organization. To specify the recipient object, such as a mailbox user or a mail-enabled (distribution) group, use the Identity parameter. For example, you can retrieve a mailbox user by means of Get-QADUser, and then pipe the output user object into this cmdlet in order to enable the effect of the e-mail address policies on the user's mailbox.

Examples

Example 1

Enable the e-mail address policy for a particular user mailbox:

```
C:\PS> Get-QADUser DomainName\UserName | Enable-QADEmailAddressPolicy
```

Disable-QADEmailAddressPolicy

Disable the e-mail address policy for an Exchange recipient.

Syntax

```
Disable-QADEmailAddressPolicy [-Identity] <IdentityParameter> [-Control <Hashtable>]
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Use this parameter to specify the recipient object. You can supply the Distinguished Name (DN), Canonical Name, GUID or, if applicable, the Domain\Name, UPN or SID of the object. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to configure an Exchange recipient so that the recipient's e-mail addresses are not affected by the changes made to e-mail address policies in your Exchange organization. To specify the recipient object, such as a mailbox user or a mail-enabled (distribution) group, use the Identity parameter. For example, you can retrieve a mailbox user by means of Get-QADUser, and then pipe the output user object into this cmdlet in order to remove the user's mailbox from the effect of the e-mail address policies.

Examples

Example 1

Disable the e-mail address policy for a particular user mailbox:

```
C:\PS> Get-QADUser DomainName\UserName | Disable-QADEmailAddressPolicy
```

Cmdlet Reference - Quest One ActiveRoles

Here you can find information about command-line tools (cmdlets) that are provided by ActiveRoles Management Shell.

This section covers the cmdlets for managing configuration data and other data specific to Quest One ActiveRoles.

All cmdlets of this category require a connection to be established to the Quest One ActiveRoles Administration Service by supplying the *Proxy* parameter.

Publish-QARSGroup

Publish a group by setting the edsvaPublished attribute.

Syntax

```
Publish-QARSGroup [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Keywords  
<UpdateStringParameter[]>] [-RequireManagerApproval] [-RequireSecondaryOwnerApproval]  
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]  
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection  
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, UPN or SID of the group to publish. The cmdlet publishes the group specified by this parameter. When you pipe an object into the cmdlet, this parameter is used to receive the object.

Keywords

Use this parameter to supply keywords for the group. Keywords are words or phrases that could help users identify the group in Quest One ActiveRoles client applications such as the Web Interface.

Parameter value can be a string array or an associative array that specifies one or more keywords to assign to the group or remove from the group. Some examples of possible parameter values are:

```
-Keywords 'keyword 1','keyword 2'
```

Replace all the existing keywords with the keywords specified.

```
-Keywords @{append=@('keyword 1','keyword 2')}
```

Add the specified keywords without removing the existing keywords.

```
-Keywords @{delete=@('keyword 1','keyword 2')}
```

Remove the specified keywords, leaving the other keywords intact.

```
-Keywords $null
```

Remove all the existing keywords.

RequireManagerApproval

Use this parameter to specify whether user requests to join or leave the group require approval by the primary owner (manager) of the group. The parameter value of `$true` configures the group to require approval. If you omit this parameter, or use the parameter value of `$false`, the group is configured so that approval is not required.

RequireSecondaryOwnerApproval

Use this parameter to specify whether user requests to join or leave the group require approval by a secondary owner of the group. The parameter value of `$true` configures the group to require approval. If you omit this parameter, or use the parameter value of `$false`, the group is configured so that approval is not required.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to publish groups by setting the `edsvaPublished` attribute. Publishing a group makes the group joinable by other people based on owner approval.

Examples

Example 1

Publish a group so that the user requests to join or leave the group require approval by the manager of the group:

```
C:\PS> Publish-QARSGroup 'DomainName\GroupName' -RequireManagerApproval $true -Proxy
```

Example 2

When publishing a group, add the "Published for Self-Service" expression to the list of keywords on that group:

```
C:\PS> Publish-QARSGroup 'DomainName\GroupName' -Keywords @{append=@('Published for Self-Service')} -RequireManagerApproval $true -Proxy
```

Unpublish-QARSGroup

Stop publishing a group by clearing the edsvaPublished attribute.

Syntax

```
Unpublish-QARSGroup [-Identity] <IdentityParameter> [-Control <Hashtable>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

Specify the Distinguished Name (DN), Canonical Name, GUID, Domain\Name, UPN or SID of the group to un-publish. The cmdlet un-publishes the group specified by this parameter. When you pipe an object into the cmdlet, this parameter is used to receive the object.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to clear the edsvaPublished attribute on a group, which causes the group not to be published. Publishing a group makes the group joinable by other people based on owner approval.

Examples

Unpublish the specified group:

```
C:\PS> Unpublish-QARSGroup 'DomainName\GroupName' -Proxy
```

Get-QARSAccessTemplate

Retrieve Access Template objects from Quest One ActiveRoles.

Syntax

```
Get-QARSAccessTemplate [[-Identity] <IdentityParameter>] [-Predefined] [-SearchRoot
<IdentityParameter[]>] [-SearchScope <SearchScope>] [-AttributeScopeQuery <String>]
[-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode
<WildcardMode>] [-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName
<String[]>] [-Name <String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn
<DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn
<DateTime>] [-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>]
[-IncludeAllProperties] [-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (*displayName*)
- Given-Name (*givenName*)
- Legacy-Exchange-DN (*legacyExchangeDN*)
- ms-DS-Additional-Sam-Account-Name (*msDS-AdditionalSamAccountName*)

- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

If you want the cmdlet to retrieve a single Access Template, specify the name, Canonical Name, or Distinguished Name of the Access Template as the value of this parameter. If you want to search for Access Templates by other properties (for example, using an LDAP filter), omit this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Predefined

Set this parameter to 'true' for the cmdlet to retrieve only those Access Templates that are marked "predefined" in Quest One ActiveRoles. The predefined Access Templates are installed with Quest One ActiveRoles, and cannot be modified or deleted.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to retrieve Quest One ActiveRoles Access Template objects that meet the conditions you specify. Each Access Template object contains information about a certain Access Template. Access Template objects can be used as input to *-QARSAccessTemplateLink cmdlets for managing Access Template links. For background information about Access Templates, see Quest One ActiveRoles Administrator Guide.

Examples

Example 1

Connect to any available Administration Service and list the names of all pre-defined Quest One ActiveRoles Access Templates located in a certain container:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplate -SearchRoot 'Configuration/Access Templates/Builtin'
-Predefined $true | format-List Name, ParentContainerDN
```

Example 2

List the general-purpose Access Templates for Active Directory data management that are included with Quest One ActiveRoles by default:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplate -SearchRoot 'Configuration/Access Templates/Active
Directory' -SearchScope 'OneLevel' -Predefined $true | format-List Name
```

Get-QARSAccessTemplateLink

Retrieve Access Template Link objects from Quest One ActiveRoles.

Syntax

```
Get-QARSAccessTemplateLink [[-Identity] <IdentityParameter>] [-DirectoryObject
<IdentityParameter[]>] [-Trustee <IdentityParameter[]>] [-AccessTemplate
<IdentityParameter[]>] [-Enabled] [-Disabled] [-AppliedTo <ATLinkFlags>]
[-SynchronizedToAD] [-Predefined] [-SearchRoot <IdentityParameter[]>] [-SearchScope
<SearchScope>] [-AttributeScopeQuery <String>] [-PageSize <Int32>] [-SizeLimit <Int32>]
[-LdapFilter <String>] [-WildcardMode <WildcardMode>] [-SearchAttributes <Object>]
[-Description <String[]>] [-DisplayName <String[]>] [-Name <String[]>] [-Anr <String>]
[-Control <Hashtable>] [-CreatedOn <DateTime>] [-CreatedAfter <DateTime>]
[-CreatedBefore <DateTime>] [-LastChangedOn <DateTime>] [-LastChangedAfter <DateTime>]
[-LastChangedBefore <DateTime>] [-IncludeAllProperties]
[-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

AccessTemplate

Specify the identity (such as name, Distinguished Name, etc.) of an Access Template for the cmdlet to retrieve Access Template links that apply the Access Template specified.

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also *ShowProgress* and *ProgressThreshold*). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)
- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AppliedTo

Depending on the cmdlet you use, this parameter lets you either retrieve or configure Access Template links that have specific settings for permission inheritance. Valid parameter values are:

- 'This' - Indicates no inheritance. The Access Template link information is only used on the object to which the Access Template is applied. Access Template link information is not inherited by any descendents of the object.
- 'ThisObjectAndAllChildObjects' - Indicates inheritance that includes the object to which the Access Template is applied, the object's immediate children, and the descendents of the object's children.
- 'ThisObjectAndImmediateChildObjects' - Indicates inheritance that includes the object itself and its immediate children. It does not include the descendents of its children.
- 'AllChildObjects' - Indicates inheritance that includes the object's immediate children and the descendants of the object's children, but not the object itself.
- 'ImmediateChildObjects' - Indicates inheritance that includes the object's immediate children only, not the object itself or the descendents of its children.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DirectoryObject

Specify the identity (such as name, Distinguished Name, Domain\Name, etc.) of a directory object for the cmdlet to retrieve Access Template links that determine Quest One ActiveRoles security settings on that object.

Disabled

Supply this parameter for the cmdlet to retrieve only those Access Template links that are configured to have no effect in Quest One ActiveRoles (disabled links).

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values “as is,” without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Enabled

Supply this parameter for the cmdlet to retrieve only those Access Template links that are configured to have effect in Quest One ActiveRoles (enabled links).

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

If you want the cmdlet to retrieve a single Access Template Link object by name, specify the name, Canonical Name, or Distinguished Name of the respective AT Link object (located in the 'Configuration/AT Links' container in the Quest One ActiveRoles Configuration namespace). If you want to search for AT Links by other properties, omit this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the `SerializeValues` parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the `Get-QADUser` or `Get-QADObject` cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the `Get-QADPSSnapinSettings` cmdlet, respectively. Using the `IncludedProperty` parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both `LastChangedAfter` and `LastChangedBefore` bounds a time interval for the objects' last change. If you supply only `LastChangedAfter`, there is no upper boundary on the date. Parameter value is a `DateTime` object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both `LastChangedAfter` and `LastChangedBefore` bounds a time interval for the objects' last change. If you supply only `LastChangedBefore`, there is no lower boundary on the date. Parameter value is a `DateTime` object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the `LastChangedAfter` and `LastChangedBefore` parameters. Parameter value is a `DateTime` object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Predefined

Set this parameter to 'true' for the cmdlet to retrieve only those Access Template links that are marked "predefined" in Quest One ActiveRoles. The predefined Access Template links are installed with Quest One ActiveRoles, and cannot be modified or deleted.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SynchronizedToAD

Set this parameter to 'true' for the cmdlet to retrieve only those Access Template links that are configured with the option to propagate permission settings to Active Directory. To retrieve only those links that do not propagate permission settings to Active Directory, set this parameter to 'false'.

Trustee

Specify the identity (such as name, Distinguished Name, Domain\Name, SID, etc.) of a security principal object (such as user or group) for the cmdlet to retrieve Access Template links that determine access rights given to that object in Quest One ActiveRoles.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to retrieve Quest One ActiveRoles Access Template Link objects (also referred to as Access Template links) that meet the conditions you specify. Each Access Template link contains information on how a certain Access Template is applied to determine access rights of a certain security principal (Trustee) on a certain directory object (securable object). Access Template Link objects can be used as input to *-QARSAccessTemplateLink cmdlets for managing Access Template link data. For background information about Access Templates, see Quest One ActiveRoles Administrator Guide.

Examples

Example 1

Connect to any available Administration Service and, for every Access Template link, list the distinguished names of the following entities:

- **Directory object** The securable object to which the given link applies an Access Template.
- **Access Template** The Access Template that is applied by the given link.
- **Trustee** The security principal whose access rights on the securable object are specified by the given link.

Namely, this command retrieves and displays the values of the DirectoryObject, AccessTemplate, and Trustee properties of the objects returned by the cmdlet:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink | format-List DirectoryObject, AccessTemplate, Trustee
```

Example 2

Given the name of an Access Template, list all objects on which the Access Template determines security settings (for each link that is based on that Access Template, list the securable object to which the link is applied). This command retrieves and displays the value of the DirectoryObject property of the objects returned by the cmdlet:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AR Server Security - Active Directory Container' | format-List DirectoryObject
```

Example 3

Given the name of an Access Template, list all objects that have their access rights defined by using the given Access Template (for each link that is based on that Access Template, list the security principal to which the link points). This command retrieves and displays the value of the Trustee property of the objects returned by the cmdlet:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AR Server Security - Active Directory Container' | format-List Trustee
```

Example 4

Given the pre-Windows 2000 name of a group, list all the Access Templates that determine access rights of that group (find all Access Template links that have the given group set as the security principal, and then, for every such link, list the Access Template on which the link is based and the securable object to which the link is applied):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -Trustee 'domainName\groupName' | format-List DirectoryObject, AccessTemplate
```

Example 5

Given the name of an Quest One ActiveRoles Managed Unit (MU), list all the Access Templates that determine security settings on that MU (find all Access Template links that have the given MU set as the securable object, and then, for every such link, list the Access Template on which the link is based and the security principal to which the link points):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -DirectoryObject 'Configuration/Managed
Units/ManagedUnitName' | format-List Trustee, AccessTemplate
```

Example 6

For a given organizational unit (OU), list the objects in that OU that have native Active Directory permission settings defined by using any Access Template (find all the Access Templates linked to any object in the OU with the option to synchronize the resulting permission settings to Active Directory):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QADObject -SearchRoot 'OrganizationalUnitName' | %{get-QARSAccessTemplateLink
-DirectoryObject $_ -SynchronizedToAD $true | format-List AccessTemplate,
DirectoryObject}
```

Example 7

Given the name of an Access Template and the name of an organizational unit (OU), remove all security settings on that OU that are determined by that Access Template (remove all links that are based on the given Access Template and applied to the given OU):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -DirectoryObject
'OrganizationalUnitName' -Predefined $false | remove-QARSAccessTemplateLink -Confirm
```

Example 8

Given the name of an Access Template and the pre-Windows 2000 name of a group, revoke all access rights from that group that are defined by using that Access Template (remove all links that are based on the given Access Template and point to the given group):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -Trustee
'domainName\groupName' -Predefined $false | remove-QARSAccessTemplateLink -Confirm
```

Set-QARSAccessTemplateLink

Make changes to existing links of Quest One ActiveRoles Access Templates.

Syntax

```
Set-QARSAccessTemplateLink [-Identity] <IdentityParameter> [-AccessTemplate
<IdentityParameter>] [-Trustee <IdentityParameter>] [-Enabled] [-AppliedTo
<ATLinkFlags>] [-SynchronizedToAD] [-Disabled] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties <String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

AccessTemplate

Specify the identity (such as name, Distinguished Name, etc.) of an Access Template you want. The cmdlet configures the link to apply that Access Template.

AppliedTo

Depending on the cmdlet you use, this parameter lets you either retrieve or configure Access Template links that have specific settings for permission inheritance. Valid parameter values are:

- 'This' - Indicates no inheritance. The Access Template link information is only used on the object to which the Access Template is applied. Access Template link information is not inherited by any descendents of the object.
- 'ThisObjectAndAllChildObjects' - Indicates inheritance that includes the object to which the Access Template is applied, the object's immediate children, and the descendents of the object's children.
- 'ThisObjectAndImmediateChildObjects' - Indicates inheritance that includes the object itself and its immediate children. It does not include the descendents of its children.
- 'AllChildObjects' - Indicates inheritance that includes the object's immediate children and the descendants of the object's children, but not the object itself.
- 'ImmediateChildObjects' - Indicates inheritance that includes the object's immediate children only, not the object itself or the descendents of its children.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DisplayName

Set the 'displayName' attribute to this parameter value.

Disabled

Supply this parameter for the cmdlet to configure the link to have no effect in Quest One ActiveRoles (disabled link).

Enabled

Supply this parameter for the cmdlet to configure the link to have effect in Quest One ActiveRoles (enabled link).

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

Identity

You can specify the name, Canonical Name, or Distinguished Name of the Access Template link (so as to identify the respective object located in the 'Configuration/AT Links' container in the Quest One ActiveRoles Configuration namespace).

Normally, pipelining is used to identify links: pass the output of the appropriate Get- cmdlet to this cmdlet. If you do so, the Identity parameter should not be supplied on the command line.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with UseDefaultExcludedProperties, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both ExcludedProperties and IncludedProperties, the cmdlet does not set the value of that attribute in the directory.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SynchronizedToAD

Set this parameter to 'true' for the cmdlet to configure the link so as to propagate permission settings to Active Directory. To disable the propagation of the permission settings that result from the link, set this parameter to 'false'.

Trustee

Specify the identity (such as name, Distinguished Name, Domain\Name, SID, etc.) of a security principal (such as a user or group) you want. The cmdlet configures the link to determine access rights of that security principal (sets the specified object as the Trustee).

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to modify existing links of Access Templates in Quest One ActiveRoles. This cmdlet takes Access Template links returned by the respective Get- cmdlet, makes changes to the link data, and commits the changes to Quest One ActiveRoles. Each Access Template link contains information on how a certain Access Template is applied to determine access rights of a certain security principal (Trustee) on a certain directory object (securable object). For background information about Access Templates, see Quest One ActiveRoles Administrator Guide.

Examples

Example 1

Connect to any available Administration Service. Then, for every Access Template link on a given Quest One ActiveRoles Managed Unit, set a particular group as Trustee. This ensures that only members of that group have access to that Managed Unit in Quest One ActiveRoles:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -DirectoryObject 'Configuration/Managed
Units/ManagedUnitName' -Predefined $false | set-QARSAccessTemplateLink -Trustee
'DomainName\GroupName' | out-Null
```

Example 2

For a given organizational unit (OU) and a given Access Template applied on that OU, ensure that the permission settings defined by the Access Template on any object held in the OU are synchronized to Active Directory (on the respective Access Template link, enable the options to synchronize permission settings to AD and to apply them on both the OU and all child objects):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QADObject 'OrganizationalUnitName' -Type organizationalUnit |
%{get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -DirectoryObject $_
-Predefined $false} | set-QARSAccessTemplateLink -SynchronizedToAD $true -AppliedTo
'ThisObjectAndAllChildObjects' | out-Null
```

Example 3

For a given organizational unit (OU) and a given Access Template, ensure that the permission settings defined by the Access Template on any object held in the OU are not synchronized to Active Directory (disable the permission synchronization option for each link that is based on that Access Template and applied to any object held in that OU):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QADObject -SearchRoot 'OrganizationalUnitName' | %{get-QARSAccessTemplateLink
-AccessTemplate 'AccessTemplateName' -DirectoryObject $_ -SynchronizedToAD $true
-Predefined $false} | set-QARSAccessTemplateLink -SynchronizedToAD $false | out-Null
```

New-QARSAccessTemplateLink

Use this cmdlet to apply Quest One ActiveRoles Access Templates.

Syntax

```
New-QARSAccessTemplateLink [[-Name] <String>] -DirectoryObject <IdentityParameter>
-AccessTemplate <IdentityParameter>-Trustee <IdentityParameter> [-AppliedTo
<ATLinkFlags>] [-SynchronizedToAD] [-Disabled] [-ObjectAttributes
<ObjectAttributesParameter>] [-Description <String>] [-DisplayName <String>]
[-ExcludedProperties <String[]>] [-IncludedProperties<String[]>] [-DeserializeValues]
[-UseDefaultExcludedProperties] [-Control <Hashtable>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

AccessTemplate

Specify the identity (such as name, Distinguished Name, etc.) of an Access Template you want. The cmdlet configures the link to apply that Access Template.

AppliedTo

Depending on the cmdlet you use, this parameter lets you either retrieve or configure Access Template links that have specific settings for permission inheritance. Valid parameter values are:

- 'This' - Indicates no inheritance. The Access Template link information is only used on the object to which the Access Template is applied. Access Template link information is not inherited by any descendents of the object.
- 'ThisObjectAndAllChildObjects' - Indicates inheritance that includes the object to which the Access Template is applied, the object's immediate children, and the descendents of the object's children.
- 'ThisObjectAndImmediateChildObjects' - Indicates inheritance that includes the object itself and its immediate children. It does not include the descendents of its children.
- 'AllChildObjects' - Indicates inheritance that includes the object's immediate children and the descendants of the object's children, but not the object itself.
- 'ImmediateChildObjects' - Indicates inheritance that includes the object's immediate children only, not the object itself or the descendents of its children.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Description

Set the 'description' attribute to this parameter value.

DeserializeValues

Supply this parameter if the input you pass to the cmdlet contains serialized attribute values (for instance, when importing a directory object from a text file that was created using the Serialize parameter). For examples of how to export and import an object, see documentation on the Get-QADUser cmdlet.

DirectoryObject

Specify the identity (such as name, Distinguished Name, Domain\Name, etc.) of a directory object you want. The cmdlet configures the link to apply the Access Template to that object (to determine security settings on that object).

Disabled

Supply this parameter for the cmdlet to configure the link to have no effect in Quest One ActiveRoles (disabled link).

DisplayName

Set the 'displayName' attribute to this parameter value.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. You could use this parameter when importing attribute values from a text file, in order to prevent some attributes found in the file from being set in the directory.

IncludedProperties

Use this parameter to specify explicitly the attributes that you want the cmdlet to set in the directory. Supply a list of the attribute LDAP display names as the parameter value. When used together with `UseDefaultExcludedProperties`, this parameter allows you to have the cmdlet set some attributes that would not be set otherwise.

Note: If a particular attribute is listed in both `ExcludedProperties` and `IncludedProperties`, the cmdlet does not set the value of that attribute in the directory.

Name

Optionally, specify a name for the link to create. If you omit this parameter, a name is auto-generated.

ObjectAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet sets the specified attributes to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to set. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SynchronizedToAD

Set this parameter to 'true' for the cmdlet to configure the link so as to propagate permission settings to Active Directory. To disable the propagation of the permission settings that result from the link, set this parameter to 'false'.

Trustee

Specify the identity (such as name, Distinguished Name, Domain\Name, SID, etc.) of a security principal (such as a user or group) you want. The cmdlet configures the link to determine access rights of that security principal (sets the specified object as the Trustee).

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to make changes to certain attributes in the directory. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the `Get-QADPSSnapinSettings` cmdlet, respectively.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Using this cmdlet, you can apply Access Templates in Quest One ActiveRoles. The operation of applying an Access Template boils down to creation of an Access Template link. This cmdlet can take Access Template objects returned by the respective Get- cmdlet and create Access Template links, thus applying the Access Templates. Each Access Template link contains information on how a certain Access Template is applied to determine access rights of a certain security principal (Trustee) on a certain directory object (securable object). For background information about Access Templates, see Quest One ActiveRoles Administrator Guide.

Examples

Example 1

Give a certain group full control access to a certain Managed Unit in Quest One ActiveRoles. This command applies the appropriate pre-defined Access Template, creating an Access Template link on the Managed Unit, with the given group set as Trustee. The default permission inheritance setting (ThisObjectAndAllChildObjects) causes the Access Template link information to be used on any object in the managed domains:

```
C:\PS> connect-QADService -Proxy
C:\PS> new-QARSAccessTemplateLink -AccessTemplate 'Configuration/Access
Templates/Active Directory/All Objects - Full Control' -DirectoryObject
'Configuration/Managed Units/ManagedUnitName' -Trustee 'DomainName\GroupName'
```

Example 2

Connect to any available Administration Service. Then, configure security settings in Quest One ActiveRoles so as to give any authenticated user read access to any object in the Active Directory domains that are registered with Quest One ActiveRoles (managed domains). This command applies the appropriate pre-defined Access Template, creating an Access Template link on each of the domainDNS objects representing the managed domains, with Authenticated Users set as Trustee. The default permission inheritance setting (ThisObjectAndAllChildObjects) causes the Access Template link information to be used on any object in the managed domains:

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QADObject -SearchRoot 'CN=Active Directory' -Type 'domainDNS' |
%{new-QARSAccessTemplateLink -AccessTemplate 'Configuration/Access Templates/Active
Directory/All Objects - Read All Properties' -DirectoryObject $_ -Trustee 'Authenticated
Users'}
```

Remove-QARSAccessTemplateLink

Delete Access Template links in Quest One ActiveRoles.

Syntax

```
Remove-QARSAccessTemplateLink [-Identity] <IdentityParameter> [-Control <Hashtable>]
[-Proxy] [-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the *Proxy* parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

Identity

You can specify the name, Canonical Name, or Distinguished Name of the Access Template link (so as to identify the respective object located in the 'Configuration/AT Links' container in the Quest One ActiveRoles Configuration namespace).

Normally, pipelining is used to identify links: pass the output of the appropriate Get- cmdlet to this cmdlet. If you do so, the Identity parameter should not be supplied on the command line.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to delete existing links of Access Templates in Quest One ActiveRoles. This cmdlet takes Access Template links returned by the respective Get- cmdlet, and requests Quest One ActiveRoles to delete those links. Each Access Template link contains information on how a certain Access Template is applied to determine access rights of a certain security principal (Trustee) on a certain directory object (securable object). For background information about Access Templates, see Quest One ActiveRoles Administrator Guide.

Examples

Example 1

Connect to any available Administration Service, create a new Access Template link, and then delete the link you created:

```
C:\PS> connect-QADService -Proxy
C:\PS> new-QARSAccessTemplateLink newATLink -AccessTemplate 'Configuration/Access
Templates/Active Directory/All Objects - Full Control' -DirectoryObject 'CN=Active
Directory' -Trustee 'Authenticated Users'
C:\PS> remove-QARSAccessTemplateLink newATLink -Confirm
```

Example 2

Given the name of an Access Template, ensure that the Access Template is no longer applied to any object (delete all links that are based on that Access Template):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -Predefined
$false | remove-QARSAccessTemplateLink -Confirm
```

Example 3

Given the name of an Access Template and the name of an organizational unit (OU), remove all security settings on that OU that are determined by that Access Template (remove all links that are based on the given Access Template and applied to the given OU):

```
C:\PS> connect-QADService -Proxy
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -DirectoryObject
'OrganizationalUnitName' -Predefined $false | remove-QARSAccessTemplateLink -Confirm
```

Example 4

Given the name of an Access Template and the pre-Windows 2000 name of a group, revoke all access rights from that group that are defined by using that Access Template (remove all links that are based on the given Access Template and point to the given group):

```
C:\PS> connect-QADService -Proxy
```

```
C:\PS> get-QARSAccessTemplateLink -AccessTemplate 'AccessTemplateName' -Trustee  
'domainName\groupName' -Predefined $false | remove-QARSAccessTemplateLink -Confirm
```

Get-QARSOperation

Retrieve operation records from Quest One ActiveRoles. Each operation record represents a certain change request, whether pending or completed, in Quest One ActiveRoles.

Syntax

```
Get-QARSOperation [[-TargetObject] <IdentityParameter[]>] [-OperationID <String[]>]
[-InitiatedAfter <DateTimeParameter>] [-InitiatedBefore <DateTimeParameter>]
[-InitiatedOn <DayParameter>] [-InitiatedRecently <RelativeDateTimeParameter>]
[-CompletedAfter <DateTimeParameter>] [-CompletedBefore <DateTimeParameter>]
[-CompletedOn <DayParameter>] [-CompletedRecently <RelativeDateTimeParameter>]
[-OperationType <OperationType[]>] [-OperationStatus <OperationStatus[]>] [-InitiatedBy
<IdentityParameter[]>] [-InitiatedByMe] [-TargetObjectType <String[]>]
[-ParentContainer <IdentityParameter[]>] [-AttributesChanges <Hashtable>]
[-ChangedAttributes <String[]>] [-SizeLimit <Int32>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

AttributeChanges

Retrieve records relating to requests for changing the specified attributes to the specified values. You can use this parameter to limit your search based on particular changes requested or made to particular attributes. Parameter value is an associative array that specifies the attributes and values you want.

Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name of an attribute and the value set on the attribute in accord with the change request. Examples:

```
@{description="Text"}
Search for requests that change Description to the text specified.
```

```
@{description="Text*"}
Search for requests that change Description to any value that begins with the text specified.
```

```
@{description="$null"}
Search for requests that clear Description.
```

When multiple attributes are specified, the search returns the records that involve changes to all those attributes (the search conditions are combined using a logical AND operator).

For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ChangedAttributes

Retrieve records relating to change requests targeted at the attribute specified. Parameter value can be an array of strings, with each string representing the LDAP display name of an attribute (such as 'description' or 'SAMAccountName'). You can use this parameter to limit your search based on any changes requested or made to particular attributes. When multiple attributes are specified, the search returns the records that involve changes to any of those attributes (the search conditions are combined using a logical OR operator).

CompletedAfter

Retrieve operation records for the change requests that were completed after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedBefore

Retrieve operation records for the change requests that were completed before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedOn

Retrieve operation records for the change requests that were completed within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedRecently

Retrieve operation records for the change requests that were completed during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the change requests completed during the last 3 days.

InitiatedAfter

Retrieve operation records for the change requests that occurred after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

InitiatedBefore

Retrieve operation records for the change requests that occurred before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

InitiatedBy

Retrieve operation records for the changes that were requested by the security principal (such as a user) specified. Parameter value can be an array of strings, with each string representing the Distinguished Name (DN), Canonical Name, Domain\Name, User Principal Name, SID or GUID of a security principal object in Active Directory. You can use this parameter to retrieve information about changes to directory data made by a particular person (user activity).

InitiatedByMe

Retrieve operation records for the changes that were requested by the security principal (such as a user) in whose security context the cmdlet is currently running.

InitiatedOn

Retrieve operation records for the change requests that occurred within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

InitiatedRecently

Retrieve operation records for the change requests that occurred during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the records regarding change requests that occurred during the last 3 days.

OperationID

Retrieve an operation record by ID. Parameter value is the string ID of the operation to retrieve (you can view operation ID in the Approval section of the Quest One ActiveRoles Web Interface).

OperationStatus

Retrieve operation records for the change requests that are completed (the changes are applied or rejected) or pending (awaiting approval). Parameter value can be one of the following:

- Canceled
- Completed
- Denied
- Failed
- InProgress
- Pending

If this parameter is omitted, operation records are retrieved for any requests.

OperationType

Retrieve records for the change requests of the category specified. Valid parameter values are:

- Create
- Delete
- Copy
- Modify
- GroupMembershipChange

- Move
- Rename
- Deprovision
- UndoDeprovision

Parameter value can be any combination of the listed values, separated by commas. For example, 'Create,Modify' limits the search to the requests for changing attributes of existing objects or creation of new objects. If this parameter is omitted, records are retrieved regardless of the change request category.

ParentContainer

Retrieve operation records for the change requests targeted at directory objects that reside in the container (such as an organizational unit) specified. Parameter value can be an array of strings, with each string representing the Distinguished Name (DN), Canonical Name, or GUID of a container object in the directory. You can use this parameter to retrieve information about changes made to directory object in a particular container.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

TargetObject

Retrieve operation records for the change requests targeted at the directory object specified. Parameter value can be an array of strings, with each string representing the Distinguished Name (DN), Canonical Name, Domain\Name, User Principal Name, SID or GUID of a directory object. You can use this parameter to retrieve information about changes made to a particular object in the directory (change history).

TargetObjectType

Retrieve operation records for the change requests targeted at the directory objects of the object class specified. Parameter value can be an array of strings, with each string representing the LDAP display name of an object class (such as 'user' or 'organizationalUnit'). You can use this parameter to retrieve information about changes made to directory objects of a particular type.

Detailed Description

Use this cmdlet to retrieve information about requests to make changes to directory data using Quest One ActiveRoles. Each request causes an operation record to be created in the Quest One ActiveRoles Management History data store. The cmdlet provides access to operation records, allowing you to examine:

- **Change History** - Information on changes that were made to particular pieces of directory data (directory objects and their attributes) using Quest One ActiveRoles.
- **User Activity** - Information on directory data changes that were made by particular users.

For a discussion of the Management History feature, see "Management History" in the Quest One ActiveRoles Administrator Guide.

An operation record may represent a change request that is waiting for approval in Quest One ActiveRoles. Such operation records are created whenever the requested changes require approval in accord with the Quest One ActiveRoles approval rules, and include information about the associated approval tasks. Once all approval tasks associated with an operation are completed, the respective operation record is marked completed and the requested changes are either applied or denied depending the approvers' resolutions on the approval tasks (Approve or Reject). For more information about the change approval function, see "Approval Workflow" in the Quest One ActiveRoles Administrator Guide.

The objects returned by this cmdlet represent operation records that meet the search conditions specified, allowing you to access information about the respective change requests. You can discover what changes were requested, when and by whom change requests were initiated, when the requested changes were applied, as well as get identities of operation records for further processing such as retrieval and accomplishment of approval tasks.

Return Type

Type of object returned by the cmdlet:

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.Operation

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.Operation Properties

Objects of the Output.Operation object type contain information about Quest One ActiveRoles operation records, representing change requests in the Quest One ActiveRoles Management History data store.

This object type exposes the following properties:

PROPERTY	DESCRIPTION
ID Syntax: <string>	An identifier of the operation record.
OperationGuid Syntax: <Guid>	Globally unique identifier of the operation record.
Type Syntax: <Enumeration>	Identifies the category of the change request represented by the operation record. Can be one of the following: <ul style="list-style-type: none">• Create• Delete• Copy• Modify• GroupMembershipChange• Move• Rename• Deprovision• UndoDeprovision

PROPERTY	DESCRIPTION
Status Syntax: <Enumeration>	Indicates the status of the change request, such as whether the changes are applied or pending (awaiting approval). Property value can be one of the following: <ul style="list-style-type: none"> • Canceled • Completed • Denied • Failed • InProgress • Pending
InitiatorInfo Syntax: <Output.PrincipalInformation>	Object containing information about the identity (for example, a user) that requested the changes. The properties exposed by this object type are listed later in this section.
TargetObjectInfo Syntax: <Output.ObjectInformation>	Object containing information about the directory object (for example, a user account or a group) to which the changes were requested. The properties exposed by this object type are listed later in this section.
Initiated Syntax: <DateTime>	Date and time that the changes were requested.
Completed Syntax: <DateTime>	Date and time that the change request was completed (the changes were applied or rejected).
TasksCount Syntax: <int>	Number of the approval tasks associated with the change request.
AttributeChanges Syntax: <AttributeChangeInfo[]>	Collection of objects containing information on the attribute changes that were requested. The properties exposed by this object type are listed later in this section.
Controls Syntax: <ControlInfo[]>	Collection of objects containing information about the control codes that were added to the change request. This object type exposes two properties: Name and Value, identifying the name and the value of the control code, respectively. For information about Quest One ActiveRoles control codes, see the Quest One ActiveRoles SDK and Resource Kit documentation.
NewObjectDN Syntax: <string>	For a change request of the Rename or Move category, identifies the Distinguished Name of the operation target object after applying the requested changes.
SourceObjectDN Syntax: <string>	For a change request of the Copy category (creation of a new object by copying an existing object), identifies the Distinguished Name of the object to copy.

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.PrincipalInformation Properties

Objects of the Output.PrincipalInformation object type contain information about Active Directory security principal objects (such as users or groups).

This object type exposes the following properties:

PROPERTY	DESCRIPTION
DN Syntax: <string>	Distinguished Name of the security principal object in Active Directory.
Guid Syntax: <Guid>	GUID of the security principal object in Active Directory.
Sid Syntax: <Principal.SecurityIdentifier>	SID of the security principal object in Active Directory.
NTAccountName Syntax: <string>	Pre-Windows 2000 name of the security principal object, in the form of Domain\Name where Domain is the NetBIOS name of the Active Directory domain in which the object is defined.
ObjectClass Syntax: <string>	Identifies the type of the security principal object (such as User or Group).
Host Syntax: <string>	Identifies the computer running the Quest One ActiveRoles client application used by the security principal.
Site Syntax: <string>	Identifies the site of the computer running the Quest One ActiveRoles client application used by the security principal.
IsDSAdmin Syntax: <bool>	Indicates whether the security principal is an AR Server Admin (DS Admin) role holder, and thus has full access to Quest One ActiveRoles.

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.ObjectInformation Properties

Objects of the Output.ObjectInformation object type contain information about directory objects.

This object type exposes the following properties:

PROPERTY	DESCRIPTION
DN Syntax: <string>	Distinguished Name of the directory object.
Guid Syntax: <Guid>	GUID of the directory object.
Sid Syntax: <Principal.SecurityIdentifier>	SID of the directory object (only applies to security principal objects).
NTAccountName Syntax: <string>	Pre-Windows 2000 name of the directory object (only applies to security principal objects), in the form of Domain\Name where Domain is the NetBIOS name of the Active Directory domain in which the object is defined.

PROPERTY	DESCRIPTION
ObjectClass Syntax: <string>	Identifies the type of the directory object (such as User or Group).

AttributeChangeInfo Properties

An object of the AttributeChangeInfo object type contains information about changes to a single attribute in the directory.

This object type exposes the following properties:

PROPERTY	DESCRIPTION
Name Syntax: <string>	Identifies the attribute by name.
Type Syntax: <Enumeration>	Data type (syntax) of the attribute. Can be one of the following: <ul style="list-style-type: none"> • Unspecified • String • Boolean • Integer • LargeInteger • DateTime • OctetString
Operation Syntax: <Enumeration>	Category of changes to the attribute. Can be one of the following: <ul style="list-style-type: none"> • Add • Delete • Replace
SerializedValues Syntax: <string[]>	Array of strings representing the new attribute values in a serialized form. The attribute is changed by applying these values in accord with the category of changes (see Operation property).

Examples

Example 1

List the user accounts from a particular container that were changed on the current date:

```
C:\PS> Get-QARSOOperation -CompletedOn 'Today' -ParentContainer
'test.domain.com/container' -TargetObjectType 'user' -OperationType 'Modify' |
%{$_.TargetObjectInfo.DN} | Group-Object | %{$_.Name}
```

Example 2

List the groups that were created in a particular container on September 1, 2008:

```
C:\PS> Get-QARSOOperation -CompletedOn (get-date -year 2008 -month 9 -day 1)
-ParentContainer 'test.domain.com/container' -TargetObjectType 'Group' -OperationType
'Create' | %{$_.TargetObjectInfo.DN} | Group-Object | %{$_.Name}
```

Example 3

List the names of the security principals that added or removed members from a particular group during last month:

```
C:\PS> Get-QARSOOperation -CompletedRecently ([TimeSpan]::FromDays(30)) -TargetObject  
'domainName\groupName' -OperationType 'GroupMembershipChange' |  
%{$_.InitiatorInfo.NTAccountName} | Group-Object | %{$_.Name}
```

Example 4

List the names of the security principals that changed or reset the password of a particular user account:

```
C:\PS> Get-QARSOOperation -TargetObject 'domain\user' -OperationType 'Modify'  
-ChangedAttributes 'edsaPassword' | %{$_.InitiatorInfo.NTAccountName} | Group-Object |  
select Name
```

Example 5

List all user accounts from a particular container that were changed by the user 'MyDomain\JSmith' during last week:

```
C:\PS> Get-QARSOOperation -CompletedRecently ([TimeSpan]::FromDays(7)) -TargetObjectType  
user -ParentContainer 'test.domain.com/container' -InitiatedBy 'MyDomain\JSmith' |  
%{$_.TargetObjectInfo.DN}
```

Example 6

List the names of the security principals that changed the City (l) or Street Address (streetAddress) attribute on the account of a particular user account yesterday:

```
C:\PS> Get-QARSOOperation -TargetObject 'domain\user' -ChangedAttributes l,streetAddress  
-CompletedOn ((get-date).AddDays(-1)) | %{$_.InitiatorInfo.NTAccountName} | Group-Object  
| select Name
```

Example 7

List the groups from a particular container that had the membership list (Members attribute) changed during the time period from September 15, 2008 to September 30, 2008:

```
C:\PS> Get-QARSOOperation -ParentContainer test.domain.com/container -TargetObjectType  
group -OperationType 'GroupMembershipChange' -CompletedAfter (get-date -year 2008  
-month 9 -day 15 -hour 0 -minute 0 -second 0) -CompletedBefore (get-date -year 2008  
-month 9 -day 30 -hour 23 -minute 59 -second 59) | %{$_.TargetObjectInfo.DN} |  
Group-Object | %{$_.Name}
```

Get-QARSLastOperation

Retrieve information about the most recent operation request submitted to Quest One ActiveRoles from the current Management Shell session.

Syntax

```
Get-QARSLastOperation [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Detailed Description

Use this cmdlet to retrieve information about the last operation request that was made within the context of the current Management Shell session. The information includes the identifier of the operation request. You can pass the identifier to cmdlets such as *Get-QARSOOperation* or *Get-QARSAApprovalTask* in order to retrieve additional information about the request and the approval tasks associated with the request.

Return Type

Type of object returned by the cmdlet: **Quest.ActiveRoles.ArsPowershellSnapIn.UI.OperationInfo**

This object type exposes the following properties:

PROPERTY	DESCRIPTION
OperationID Syntax: <string>	An identifier of the operation record. You can pass this identifier to the <i>Get-QARSOOperation</i> through the <i>OperationID</i> parameter, to retrieve more information about the operation request.

PROPERTY	DESCRIPTION
OperationStatus Syntax: <Enumeration>	Indicates the status of the operation request, such as whether the requested changes are applied or pending (awaiting approval). Property value can be one of the following: <ul style="list-style-type: none">• Canceled• Completed• Denied• Failed• InProgress• Pending

Examples

Example 1

Connect to any available Quest One ActiveRoles Administration Service, submit a request to create a new user, and then retrieve information on that request:

```
C:\PS> Connect-QADService -Proxy
C:\PS> New-QADUser -ParentContainer 'labdomain.local/Users' -Name 'dummy'
C:\PS> Get-QARSLastOperation
```

Get-QARSWorkflowDefinition

Retrieve workflow definition objects from Quest One ActiveRoles. Each workflow definition object contains the configuration data for a certain workflow.

Syntax

```
Get-QARSWorkflowDefinition [[-Identity] <IdentityParameter>] [-SearchRoot
<IdentityParameter[]>] [-SearchScope <SearchScope>] [-AttributeScopeQuery <String>]
[-PageSize <Int32>] [-SizeLimit <Int32>] [-LdapFilter <String>] [-WildcardMode
<WildcardMode>] [-SearchAttributes <Object>] [-Description <String[]>] [-DisplayName
<String[]>] [-Name <String[]>] [-Anr <String>] [-Control <Hashtable>] [-CreatedOn
<DateTime>] [-CreatedAfter <DateTime>] [-CreatedBefore <DateTime>] [-LastChangedOn
<DateTime>] [-LastChangedAfter <DateTime>] [-LastChangedBefore <DateTime>]
[-IncludeAllProperties] [-DontConvertValuesToFriendlyRepresentation] [-SerializeValues]
[-ReturnPropertyNamesOnly] [-DontUseDefaultIncludedProperties]
[-UseDefaultExcludedProperties] [-ExcludedProperties <String[]>] [-IncludedProperties
<String[]>] [-UseDefaultExcludedPropertiesExcept <String[]>] [-ShowProgress] [-Activity
<String>] [-ProgressThreshold <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Activity

Use this parameter to specify the line of text above the progress bar which the cmdlet displays to depict the status of the running command in case of a lengthy operation. This text describes the activity whose progress is being reported (see also ShowProgress and ProgressThreshold). If this parameter is omitted, the name of the cmdlet is displayed above the progress bar.

Anr

Specify a value to be resolved using ambiguous name resolution (ANR). Which attributes are included in an ANR search depends upon the Active Directory schema. Thus, in Windows Server 2003 based Active Directory, the following attributes are set for ANR by default:

- Display-Name (displayName)
- Given-Name (givenName)
- Legacy-Exchange-DN (legacyExchangeDN)

- ms-DS-Additional-Sam-Account-Name (msDS-AdditionalSamAccountName)
- Physical-Delivery-Office-Name (physicalDeliveryOfficeName)
- Proxy-Addresses (proxyAddresses)
- RDN (name)
- SAM-Account-Name (sAMAccountName)
- Surname (sn)

For instance, when you supply 'ann*' as the value of this parameter, the cmdlet searches for objects that have ann at the beginning of the value of at least one of the attributes listed above.

AttributeScopeQuery

Specify the LDAP display name of an attribute that has DN syntax (for example, "member" or "memberOf"). The cmdlet enumerates the Distinguished Name values of the attribute on the object specified by the SearchRoot parameter, and performs the search on the objects represented by the Distinguished Names. The SearchScope parameter has no effect in this case. The object to search must be specified by using the SearchRoot parameter rather than the Identity parameter.

For instance, with the value of this parameter set to "memberOf", the cmdlet searches the collection of the groups to which the SearchRoot object belongs.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect on the operations that are performed through Quest One ActiveRoles (connection established using the Proxy parameter); otherwise, this parameter causes an error condition in ActiveRoles Management Shell.

CreatedAfter

Specify the lower boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created after the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedBefore

Specify the upper boundary of the object creation date and time by which to filter objects found. The cmdlet returns only the objects that were created before the date and time specified. Supplying both CreatedAfter and CreatedBefore bounds a time interval for the objects' creation. If you supply only CreatedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

CreatedOn

Specify the object creation date by which to filter objects found, searching for objects created within the date specified. This parameter is mutually exclusive with the CreatedAfter and CreatedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

Description

Search by the 'description' attribute.

DisplayName

Search by the 'displayName' attribute.

DontConvertValuesToFriendlyRepresentation

This parameter causes the cmdlet to represent the Integer8 and OctetString attribute values "as is," without converting them to a user-friendly, human-readable form. If this parameter is omitted, the cmdlet performs the following data conversions:

- The values of the Integer8 attributes listed in the Integer8AttributesThatContainDateTimes array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to DateTime.
- The values of the Integer8 attributes listed in the Integer8AttributesThatContainNegativeTimeSpans array (see parameter descriptions for the Get- and Set-QADPSSnapinSettings cmdlets) are converted from IADsLargeInteger to TimeSpan.
- The values of the other Integer8 attributes are converted from IADsLargeInteger to Int64.
- The values of the OctetString attributes are converted from byte[] to BinHex strings.

Note: This parameter has an effect only on the properties of the output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

DontUseDefaultIncludedProperties

This parameter causes the cmdlet to load only a small set of attributes from the directory to the local memory cache (normally, this set is limited to objectClass and ADsPath). Other attributes are retrieved from the directory as needed when you use the cmdlet output objects to read attribute values. Thus, if you want only to count the objects that meet certain conditions (rather than examine values of particular attributes), then you can use this parameter to increase performance of your search. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

ExcludedProperties

Use this parameter to specify the attributes that you do not want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the ExcludedProperties parameter you can change this default behavior on an ad-hoc basis, in order to prevent certain attributes from being loaded. Another scenario involves the use of this parameter in conjunction with IncludeAllProperties in order to restrict the set of the cached attributes.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

Identity

Specify the Distinguished Name (DN) of the workflow definition object you want the cmdlet to find. The cmdlet attempts to retrieve the object identified by this parameter value, disregarding the other search parameters. If you want other search parameters to have effect, do not supply this parameter.

IncludeAllProperties

With this parameter, the cmdlet retrieves all attributes of the respective directory object (such as a User object), and stores the attribute values in the memory cache on the local computer. Attribute values can be read from the memory cache by using properties of the object returned by the cmdlet. Thus, when used in conjunction with the SerializeValues parameter, it allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

IncludedProperties

Use this parameter to specify the attributes that you want the cmdlet to retrieve from the directory and store in the memory cache on the local computer. Supply a list of the attribute LDAP display names as the parameter value. By default, the cmdlet caches a certain pre-defined set of attributes, which you can view or modify by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Using the IncludedProperty parameter you can direct the cmdlet to cache some attributes in addition to the default set.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet.

LastChangedAfter

Specify the lower boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed after the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedAfter, there is no upper boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedBefore

Specify the upper boundary of the object modification date and time by which to filter objects found. The cmdlet returns only the objects that have last changed before the date and time specified. Supplying both LastChangedAfter and LastChangedBefore bounds a time interval for the objects' last change. If you supply only LastChangedBefore, there is no lower boundary on the date. Parameter value is a DateTime object that specifies the date and time you want.

LastChangedOn

Specify the object modification date by which to filter objects found, searching for objects that have last changed within the date specified. This parameter is mutually exclusive with the LastChangedAfter and LastChangedBefore parameters. Parameter value is a DateTime object that specifies the date you want.

LdapFilter

Specify the LDAP search filter that defines your search criteria. Note that the search filter string is case-sensitive.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply an Identity parameter value.

If you supply the LdapFilter parameter along with attribute-specific parameters, then your search returns objects that meet the conditions defined by the LDAP filter and have the specified attributes set to the specified values.

Name

Search by the 'name' attribute.

PageSize

Set the maximum number of items in each page of the search results that will be returned by the cmdlet. After the directory server has found the number of objects that are specified by this parameter, it will stop searching and return the results to the cmdlet. When the cmdlet requests more data, the server will restart the search where it left off. You can use this setting to adjust the number of requests (network calls) to the directory server issued by the cmdlet during a search.

Normally, the default page size is 50. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

ProgressThreshold

Use this parameter to specify a delay, in seconds, before the cmdlet displays a progress bar that depicts the status of the running command in case of a lengthy operation. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. The default threshold time setting can be configured by using the Set-QADProgressPolicy cmdlet.

ReturnPropertyNamesOnly

This parameter causes the cmdlet to list the names of the object attributes whose values the cmdlet retrieves from the directory and stores in the memory cache on the local computer. Thus, when used in conjunction with the IncludeAllProperties parameter, it lists the names of all attributes of the respective directory object (such as a User object). For examples of how to use this parameter, see documentation on the Get-QADUser or Get-QADObject cmdlet.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

SearchAttributes

Specify an associative array that defines the object attributes and values you want. The cmdlet searches for objects that have the specified attributes set to the specified values. Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name and the value of an attribute to search. A value may include an asterisk character - a wildcard representing any group of characters. For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

SearchRoot

Specify the Distinguished Name, Canonical Name, or GUID of the container to search. By default, the cmdlet searches the entire sub-tree of which SearchRoot is the topmost object (sub-tree search). This default behavior can be altered by using the SearchScope parameter.

The search criteria are defined by the LdapFilter parameter value and the values of attribute-specific parameters.

The cmdlet disregards this parameter if an Identity value is supplied. If you want this parameter to have effect, do not supply any Identity parameter value.

SearchScope

Specify one of these parameter values:

- 'Base' - Limits the search to the base (SearchRoot) object. The result contains a maximum of one object.
- 'OneLevel' - Searches the immediate descendant (child) objects of the base (SearchRoot) object, excluding the base object.
- 'Subtree' - Searches the whole sub-tree, including the base (SearchRoot) object and all its descendant objects.

Normally, if this parameter is not supplied, the cmdlet performs a Subtree search. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

SerializeValues

This parameter causes the cmdlet to output an object whose properties store the attribute values of the respective directory object that are loaded to the local memory cache. The value returned by each property of the output object is represented as a string (serialized) so as to facilitate the export of the attribute values to a text file. Thus, when used in conjunction with the IncludeAllProperties parameter, this parameter allows an entire object to be exported from the directory to a text file. For examples of how to use this parameter, see documentation on the Get-QADUser cmdlet.

ShowProgress

Supply this parameter if you want the cmdlet to display a progress bar that depicts the status of the running command in case of a lengthy operation. If this parameter is omitted, whether the cmdlet displays a progress bar depends upon the ShowProgress setting configured by using the Set-QADProgressPolicy cmdlet.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

Type

Specify the type of directory objects to find. The cmdlet searches for objects that have one of the 'objectClass' attribute values set to the Type parameter value.

UseDefaultExcludedProperties

When set to 'true', this parameter causes the cmdlet not to load a certain pre-defined set of attributes from the directory to the local memory cache. This pre-defined set of attributes (referred to as "default excluded properties") can be viewed or modified by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. Normally, this parameter is used in conjunction with IncludeAllProperties to avoid retrieval of unnecessary data from the directory server, thereby increasing performance of the search operation performed by the cmdlet.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

UseDefaultExcludedPropertiesExcept

This parameter is deprecated, and has no effect.

WildcardMode

Specify either 'PowerShell' or 'LDAP' as the parameter value. Normally, if this parameter is not supplied, the cmdlet assumes that WildcardMode is set to 'LDAP'. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively. The 'PowerShell' value causes the cmdlet to use PowerShell wildcards and quoting rules. Wildcards are processed on the client side, which may result in slow search performance.

For information about PowerShell wildcards and quoting rules, type the following commands at the PowerShell command-prompt:

```
help about_wildcard
help about_quoting_rule
```

The 'LDAP' value causes the cmdlet to use LDAP wildcards (asterisks only) and LDAP quoting rules (backslash as the escape character). Wildcards are processed on the server side, which enables faster search results.

Detailed Description

Use this cmdlet to search for, and retrieve information from, workflow definitions in Quest One ActiveRoles. Each workflow definition represents the structure of a certain workflow, is stored as a single object in the Quest One ActiveRoles configuration data store, and can be structured as an XML document defining the workflow start conditions, the activities, the parameters for the activities, and the order in which the activities should run. For more information, see "Workflows" in the Quest One ActiveRoles Administrator Guide.

The cmdlet returns a collection of objects, each of which represents one of the workflow definition objects that meet the search conditions specified. You can pass returned objects to the `Get-QARSWorkflowInstance` cmdlet through the `Workflow` parameter, in order to get information about workflow instances originated in Quest One ActiveRoles based on the corresponding workflow definitions.

Examples

Example 1

List all the pre-defined workflow definitions:

```
C:\PS> Get-QARSWorkflowDefinition -SearchRoot  
'CN=Builtin,CN=Workflow,CN=Policies,CN=Configuration'
```

Get-QARSWorkflowInstance

Retrieve workflow instance records from Quest One ActiveRoles. Each record contains information about a certain workflow, whether pending or completed, that was originated by a particular operation request in Quest One ActiveRoles.

Syntax

```
Get-QARSWorkflowInstance [-InstanceID <String[]>] [-Operation
<OperationIdentityParameter[]>] [-Workflow <IdentityParameter[]>] [-CreatedAfter
<DateTimeParameter>] [-CreatedBefore <DateTimeParameter>] [-CreatedOn <DayParameter>]
[-CreatedRecently <RelativeDateTimeParameter>] [-CompletedAfter <DateTimeParameter>]
[-CompletedBefore <DateTimeParameter>] [-CompletedOn <DayParameter>]
[-CompletedRecently <RelativeDateTimeParameter>] [-OperationType <OperationType[]>]
[-TaskStatus <TaskStatus[]>] [-AttributesChanges <Hashtable>] [-ChangedAttributes
<String[]>] [-SizeLimit <Int32>] [-Proxy] [-UseGlobalCatalog] [-Service <String>]
[-ConnectionAccount <String>] [-ConnectionPassword <SecureString>] [-Credential
<PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

AttributeChanges

Retrieve records relating to requests for changing the specified attributes to the specified values. You can use this parameter to limit your search based on particular changes requested or made to particular attributes. Parameter value is an associative array that specifies the attributes and values you want.

Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name of an attribute and the value set on the attribute in accord with the change request. Examples:

```
@{description="Text"}
```

Search for requests that change Description to the text specified.

```
@{description="Text*"}
```

Search for requests that change Description to any value that begins with the text specified.

```
@{description="$null"}
```

Search for requests that clear Description.

When multiple attributes are specified, the search returns the records that involve changes to all those attributes (the search conditions are combined using a logical AND operator).

For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ChangedAttributes

Retrieve records relating to change requests targeted at the attribute specified. Parameter value can be an array of strings, with each string representing the LDAP display name of an attribute (such as 'description' or 'SAMAccountName'). You can use this parameter to limit your search based on any changes requested or made to particular attributes. When multiple attributes are specified, the search returns the records that involve changes to any of those attributes (the search conditions are combined using a logical OR operator).

CompletedAfter

Retrieve records for the workflow instances that were finished after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedBefore

Retrieve records for the workflow instances that were finished before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedOn

Retrieve records for the workflow instances that were finished within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedRecently

Retrieve records for the workflow instances that were finished during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the workflows that were finished during the last 3 days.

CreatedAfter

Retrieve workflow instance records associated with the change requests that occurred after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedBefore

Retrieve workflow instance records associated with the change requests that occurred before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedOn

Retrieve workflow instance records associated the change requests that occurred within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedRecently

Retrieve workflow instance records associated with the change requests that occurred during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the workflow instances associated with the change requests that occurred during the last 3 days.

InstanceID

Retrieve workflow instance records by ID. Parameter value is an array of strings, with each string representing the ID of the workflow instance to retrieve.

Operation

Retrieve workflow instance records that are associated with the operation requests specified. Parameter value is an object or a collection of objects returned by the Get-QARSOOperation cmdlet, or a string array of operation IDs. You can use this parameter to get information about the workflow instances originated by a particular operation request in Quest One ActiveRoles.

OperationType

Retrieve records for the change requests of the category specified. Valid parameter values are:

- Create
- Delete
- Copy
- Modify
- GroupMembershipChange
- Move
- Rename
- Deprovision
- UndoDeprovision

Parameter value can be any combination of the listed values, separated by commas. For example, 'Create,Modify' limits the search to the requests for changing attributes of existing objects or creation of new objects. If this parameter is omitted, records are retrieved regardless of the change request category.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

TaskStatus

Retrieve records for the workflow instances that have a certain status, such as finished (completed) or waiting for a certain activity to be completed (pending). Valid parameter values are:

- Pending
- Completed
- Canceled

Workflow

Retrieve workflow instance records that are based on the workflow definitions specified. Parameter value is an object or a collection of objects returned by the Get-QARSWorkflowDefinition cmdlet, or an array of strings each of which represents the Distinguished Name of a workflow definition object. You can use this parameter to get information about the instances of a particular workflow definition in Quest One ActiveRoles.

Detailed Description

Use this cmdlet to retrieve information about workflow instances. When an operation request starts a workflow in Quest One ActiveRoles, it creates a workflow instance based on the settings found in the workflow definition. Each workflow instance stores the data indicating the current state of a single workflow that is in progress (pending) or finished (completed). For more information, see "Workflows" in the Quest One ActiveRoles Administrator Guide.

The objects returned by this cmdlet represent workflow instance records that meet the search conditions specified, allowing you to access information about the corresponding workflow instances. You can analyze the returned object to discover what operation started the workflow, when the workflow was started, and whether the workflow is completed. You can also pass the returned object to the Get-QARSApprovalTask cmdlet through the WorkflowInstance parameter in order to get information about the approval tasks, if any, that were originated within the workflow.

Return Type

Type of object returned by the cmdlet:

Quest.ActiveRoles.ArsPowerShellSnapIn.UI.WorkflowInstanceUI

Quest.ActiveRoles.ArsPowerShellSnapIn.UI.WorkflowInstanceUI Properties

Objects of the UI.WorkflowInstanceUI object type contain information about Quest One ActiveRoles workflow instance records, representing workflows that are running or completed in Quest One ActiveRoles.

This object type exposes the following properties:

PROPERTY	DESCRIPTION
ID Syntax: <string>	An identifier of the workflow instance.
WorkflowInstanceID Syntax: <Guid>	Globally unique identifier of the workflow instance.

PROPERTY	DESCRIPTION
Status Syntax: <Enumeration>	Indicates the status of the workflow instance, such as whether the workflow is finished (completed) or waiting for a certain activity to be completed (pending). Property value can be one of the following: <ul style="list-style-type: none"> • Pending • Completed • Canceled
Workflow Syntax: <string>	Distinguished name of the workflow definition object based on which the workflow instance was created.
Created Syntax: <DateTime>	Date and time that the workflow instance was originated.
LastChanged Syntax: <DateTime>	Date and time that the information in the workflow instance was last updated.
Completed Syntax: <DateTime>	Date and time that the workflow instance was completed.
Operation Syntax: <Output.Operation>	Object containing information about the operation request that the workflow instance is associated with. This is the operation request that caused the workflow to start. For a list of properties exposed by this object type, see the "Return Type" topic for the Get-QARSOOperation cmdlet.

Examples

Example 1

List the workflow instances that were created more than 30 days ago and have not been completed:

```
C:\PS> Get-QARSWorkflowInstance -CreatedBefore ((get-date).AddDays(-30))
-TaskStatus Pending
```

Example 2

List all workflow instances that were created based on a particular workflow definition and have not been completed:

```
C:\PS> Get-QARSWorkflowInstance -Workflow 'CN=Approval by Primary Owner
(Manager),CN=Builtin,CN=Workflow,CN=Policies,CN=Configuration' -TaskStatus Pending
```

Get-QARSApprovalTask

Retrieve approval task records from Quest One ActiveRoles. Each approval task record represents a task, whether pending or completed, to approve or reject a certain change request in Quest One ActiveRoles.

Syntax

```
Get-QARSApprovalTask [-TaskID <String[]>] [-Operation <OperationIdentityParameter[]>]
[-WorkflowInstance <WorkflowInstanceParameter[]>] [-CreatedAfter <DateTimeParameter>]
[-CreatedBefore <DateTimeParameter>] [-CreatedOn <DayParameter>] [-CreatedRecently
<RelativeDateTimeParameter>] [-CompletedAfter <DateTimeParameter>] [-CompletedBefore
<DateTimeParameter>] [-CompletedOn <DayParameter>] [-CompletedRecently
<RelativeDateTimeParameter>] [-OperationType <OperationType[]>] [-TaskStatus
<TaskStatus[]>] [-Approver <IdentityParameter[]>] [-ApproverIsMe] [-CompletedBy
<IdentityParameter[]>] [-CompletedByMe] [-AttributesChanges <Hashtable>]
[-ChangedAttributes <String[]>] [-SizeLimit <Int32>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Approver

Retrieve records for the approval tasks that have the specified security principal (such as a user or group) designated as an approver (a person authorized to approve or reject the respective change request). Parameter value can be an array of strings, with each string representing the Distinguished Name (DN), Canonical Name, Domain\Name, User Principal Name, SID or GUID of a security principal object in Active Directory. You can use this parameter to retrieve information about approval tasks assigned to particular persons.

ApproverIsMe

Retrieve records for the approval tasks that are assigned to the security principal (such as a user) in whose security context the cmdlet is currently running.

AttributeChanges

Retrieve records relating to requests for changing the specified attributes to the specified values. You can use this parameter to limit your search based on particular changes requested or made to particular attributes. Parameter value is an associative array that specifies the attributes and values you want.

Array syntax:

```
@{attr1='val1';attr2='val2';...}
```

In this syntax, each of the key-value pairs is the LDAP display name of an attribute and the value set on the attribute in accord with the change request. Examples:

```
@{description="Text"}
```

Search for requests that change Description to the text specified.

```
@{description="Text*"}
```

Search for requests that change Description to any value that begins with the text specified.

```
@{description="$null"}
```

Search for requests that clear Description.

When multiple attributes are specified, the search returns the records that involve changes to all those attributes (the search conditions are combined using a logical AND operator).

For information about associative arrays, type the following command at the PowerShell command-prompt:

```
help about_associative_array
```

ChangedAttributes

Retrieve records relating to change requests targeted at the attribute specified. Parameter value can be an array of strings, with each string representing the LDAP display name of an attribute (such as 'description' or 'sAMAccountName'). You can use this parameter to limit your search based on any changes requested or made to particular attributes. When multiple attributes are specified, the search returns the records that involve changes to any of those attributes (the search conditions are combined using a logical OR operator).

CompletedAfter

Retrieve records for the approval tasks that were completed (that is, received the Approve or Reject resolution) after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedBefore

Retrieve records for the approval tasks that were completed (that is, received the Approve or Reject resolution) before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedBy

Retrieve records for the approval tasks that were completed by the security principal (such as a user) specified. Parameter value can be an array of strings, with each string representing the Distinguished Name (DN), Canonical Name, Domain\Name, User Principal Name, SID or GUID of a security principal object in Active Directory. You can use this parameter to retrieve information about approval tasks to which a particular person has applied the Approve or Reject resolution.

CompletedByMe

Retrieve records for the approval tasks that were completed by the security principal (such as a user) in whose security context the cmdlet is currently running.

CompletedOn

Retrieve records for the approval tasks that were completed (that is, received the Approve or Reject resolution) within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CompletedRecently

Retrieve records for the approval tasks that were completed (that is, received the Approve or Reject resolution) during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the records regarding the approval tasks completed during the last 3 days.

CreatedAfter

Retrieve approval task records for the change requests that occurred after the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedBefore

Retrieve approval task records for the change requests that occurred before the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedOn

Retrieve approval task records for the change requests that occurred within the date specified. Parameter value can be a DateTime object or a string that specifies the date you want. Thus, you can supply "Today" as a parameter value.

CreatedRecently

Retrieve approval task records for the change requests that occurred during the recent time period specified. Parameter value is a TimeSpan object that specifies the time period you want. For example, if you supply a time span of 3 days, the cmdlet searches for the task records specific to the change requests that occurred during the last 3 days.

Operation

Retrieve approval task records that are associated with the operation records specified. Parameter value is an object or a collection of objects returned by the Get-QARSOOperation cmdlet, or an array of operation string IDs. You can use this parameter to access information about approval tasks specific to a particular change request that requires approval in Quest One ActiveRoles.

OperationType

Retrieve records for the change requests of the category specified. Valid parameter values are:

- Create
- Delete

- Copy
- Modify
- GroupMembershipChange
- Move
- Rename
- Deprovision
- UndoDeprovision

Parameter value can be any combination of the listed values, separated by commas. For example, 'Create,Modify' limits the search to the requests for changing attributes of existing objects or creation of new objects. If this parameter is omitted, records are retrieved regardless of the change request category.

SizeLimit

Set the maximum number of items to be returned by the cmdlet. Normally, the default size limit is 1000. You can view or modify this default setting by using the Get- or Set-QADPSSnapinSettings cmdlet, respectively.

TaskID

Retrieve approval task records by ID. Parameter value is an array of strings, with each string representing the ID of the task to retrieve (you can view task ID in the Approval section of the Quest One ActiveRoles Web Interface).

TaskStatus

Retrieve records for the approval tasks that are completed with a certain resolution (Approve or Reject) or pending (awaiting resolution). Valid parameter values are:

- Pending
- Canceled
- CompletedApproved
- CompletedRejected

Parameter value can be any combination of the listed values, separated by commas. For example, 'CompletedApproved,CompletedRejected' limits the search to the tasks that have been completed with any resolution. If this parameter is omitted, records are retrieved regardless of the tasks status (a search is performed against all tasks).

WorkflowInstance

Retrieve approval task records that are associated with the workflow instance records specified. By using this parameter, you can get information about the approval tasks originated by a particular workflow instance. Parameter value is an object or a collection of objects returned by the Get-QARWorkflowInstance cmdlet, or an array of workflow instance string IDs.

Detailed Description

Use this cmdlet to retrieve information about tasks of approving changes to directory data that were requested using Quest One ActiveRoles.

When changes are attempted that require approval, one or more approval task records are created in the Quest One ActiveRoles Management History data store, and associated with the operation record representing the change request.

Once all approval tasks associated with an operation are completed, the operation record is marked completed and the requested changes are either applied or denied depending the approvers' resolutions on the approval tasks (Approve or Reject). When any task associated with an operation receives the Reject resolution, the entire operation is denied and the requested changes are disregarded. When all tasks associated with the operation receive the Approve resolution, the operation is allowed and the requested changes are applied. For more information on the change approval function, see "Approval Workflow" in the Quest One ActiveRoles Administrator Guide.

The objects returned by this cmdlet represent approval task records that meet the search condition specified, allowing you to access information about the approval tasks and the operation they govern. You can discover who is expected to perform approval tasks, when and by whom the requested changes were approved or rejected, as well as get identities of approval task records for further processing in order to perform approval tasks.

Return Type

Type of object returned by the cmdlet:

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.ApprovalTask

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.ApprovalTask Properties

Objects of the Output.ApprovalTask object type contain information about Quest One ActiveRoles approval task records, representing tasks to approve or reject change requests in Quest One ActiveRoles.

This object type exposes the following properties:

PROPERTY	DESCRIPTION
ID Syntax: <string>	An identifier of the approval task record.
TaskGuid Syntax: <Guid>	Globally unique identifier of the approval task record.
Status Syntax: <Enumeration>	Indicates the status of the approval task, such as whether the task is completed (the approver has applied the Approve or Reject resolution on the task) or pending (awaiting an approver's action). Property value can be one of the following. <ul style="list-style-type: none">• Pending• Canceled• Completed• CompletedApproved• CompletedRejected• Reviewed
ApprovalRuleGuid Syntax: <Guid>	Identifies the Quest One ActiveRoles approval rule based on which the approval task was created.

PROPERTY	DESCRIPTION
Created Syntax: <DateTime>	Date and time that the approval task record was created.
LastChanged Syntax: <DateTime>	Date and time that the approval task was last changed.
Completed Syntax: <DateTime>	Date and time that the approval task was completed.
CompletedBy Syntax: <Output.PrincipalInformation>	Object containing information about the identity (such as a user) who completed the task (applied the Approve or Reject resolution on the task). The properties exposed by this object type are listed in the section that discusses the Get-QARSOOperation cmdlet.
CompletionReason Syntax: <string>	Comment text that was entered by the approver upon completion of the task.
ApproversInfo Syntax: <Output.ObjectInformation[]>	Collection of objects containing information about the approvers assigned to the task (the identities that are expected to perform the approval task). The properties exposed by this object type are listed in the section that discusses the Get-QARSOOperation cmdlet.
Operation Syntax: <Output.Operation>	Object containing information about the operation (change request) with which the approval task is associated. For a list of properties exposed by this object type, see the "Return Type" topic for the Get-QARSOOperation cmdlet.

Examples

Example 1

List all approval tasks awaiting response from the user 'MyDomain\JSmith':

```
C:\PS> Get-QARSAprovalTask -Approver 'MyDomain\JSmith'
```

Example 2

List all change requests (operations) awaiting approval by the current user:

```
C:\PS> Get-QARSAprovalTask -ApproverIsMe -TaskStatus Pending | %{$_.Operation}
```

Example 3

List all change requests (operations) awaiting approval by the user 'MyDomain\JSmith':

```
C:\PS> Get-QARSAprovalTask -Approver 'MyDomain\JSmith' -TaskStatus Pending | %{$_.Operation}
```

Example 4

List all change requests (operations) that were approved by the user 'MyDomain\JSmith' during last week:

```
C:\PS> Get-QARSAprovalTask -CompletedRecently ([TimeSpan]::FromDays(7)) -CompletedBy 'MyDomain\JSmith' -TaskStatus CompletedApproved | %{$_.Operation}
```

Example 5

List all approval tasks that the user 'MyDomain\JSmith' completed with the Reject resolution during last week:

```
C:\PS> Get-QARSApprovalTask -CompletedRecently ([TimeSpan]::FromDays(7)) -CompletedBy  
'MyDomain\JSmith' -TaskStatus CompletedRejected
```

Approve-QARSAprovalTask

Apply the Approve resolution on approval tasks for which you are assigned to the Approver role in Quest One ActiveRoles.

Syntax

```
Approve-QARSAprovalTask [-Task] <TaskIdentityParameter> [-Reason <String>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Reason

Supply any text you want to comment your decision. This text is saved by Quest One ActiveRoles for reporting and audit purposes, and can be accessed through the respective property of the ApprovalTask object.

Task

Pass an object returned by the Get-QARSAprovalTask cmdlet to this parameter. This parameter also accepts the numeric ID of an approval task (you can view task ID in the Approval section of the Quest One ActiveRoles Web Interface).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to approve change requests that require your approval in Quest One ActiveRoles. You can take an object returned by the Get-QARSAprovalTask cmdlet and pipe that object into this cmdlet in order to complete the respective approval task with the Approve resolution.

Every change request that requires your approval has an approval task for you to allow or deny the requested changes. When you complete your approval task with the Approve resolution, you allow the changes. Note that the changes may also require approval by other persons. In this case, the changes are applied only after all approval tasks (including yours) are completed with the Approve resolution. For more information about approval tasks, see description of the Get-QARSApprovalTask cmdlet.

Return Type

Type of object returned by the cmdlet:

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.ApprovalTask

The properties exposed by this object type are listed in the section that discusses the Get-QARSApprovalTask cmdlet.

Examples

Example 1

Approve all changes that are awaiting approval by the current user:

```
C:\PS> Get-QARSApprovalTask -ApproverIsMe -TaskStatus Pending | Approve-QARSApprovalTask
```

Example 2

Approve all changes to a particular user account that are awaiting approval by the current user:

```
C:\PS> Get-QARSOperation -TargetObject 'domainName\userName' -OperationStatus Pending |  
%{Get-QARSApprovalTask -Operation $_ -ApproverIsMe} | Approve-QARSApprovalTask
```

Reject-QARSApprovalTask

Apply the Reject resolution on approval tasks for which you are assigned to the Approver role in Quest One ActiveRoles.

Syntax

```
Reject-QARSApprovalTask [-Task] <TaskIdentityParameter> -Reason <String> [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Note that this cmdlet requires a connection to the ActiveRoles Administration Service, so the *Proxy* parameter must be used to establish a connection.

Parameters

Confirm

Prompts you for confirmation before executing the command.

Reason

Supply any text you want to comment your decision. This text is saved by Quest One ActiveRoles for reporting and audit purposes, and can be accessed through the respective property of the ApprovalTask object.

Task

Pass an object returned by the Get-QARSApprovalTask cmdlet to this parameter. This parameter also accepts the numeric ID of an approval task (you can view task ID in the Approval section of the Quest One ActiveRoles Web Interface).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to reject change requests that require your approval in Quest One ActiveRoles. You can take an object returned by the Get-QARSApprovalTask cmdlet and pipe that object into this cmdlet in order to complete the respective approval task with the Reject resolution.

Every change request that requires your approval has an approval task for you to allow or deny the requested changes. When you complete your approval task with the Reject resolution, you effectively deny the changes, preventing them from being applied. Note that the changes may also require approval by other persons. In this case, the changes are not applied if any one of the approval tasks is completed with the Reject resolution. For more information about approval tasks, see description of the Get-QARSApprovalTask cmdlet.

Return Type

Type of object returned by the cmdlet:

Quest.ActiveRoles.ArsPowerShellSnapIn.Output.ApprovalTask

The properties exposed by this object type are listed in the section that discusses the Get-QARSApprovalTask cmdlet.

Examples

Example 1

Reject all changes to a particular user account that were requested on the current date and are awaiting approval by the current user:

```
C:\PS> Get-QARSOOperation -TargetObject 'domainName\userName' -InitiatedOn 'Today'  
-OperationStatus Pending | %{Get-QARSApprovalTask -Operation $_ -ApproverIsMe} |  
Reject-QARSApprovalTask
```

Example 2

Reject all changes to a particular user account that were requested by the user 'MyDomain\JohnSmith' and are awaiting approval by the current user:

```
C:\PS> Get-QARSOOperation -TargetObject 'domainName\userName' -InitiatedBy  
'MyDomain\JohnSmith' -OperationStatus Pending | %{Get-QARSApprovalTask  
-Operation $_ -ApproverIsMe} | Reject-QARSApprovalTask
```

Cmdlet Reference - X.509 Certificate Management

Here you can find information about command-line tools (cmdlets) that are provided by ActiveRoles Management Shell.

This section covers the cmdlets for managing X.509 certificates. ActiveRoles Management Shell provides a number of cmdlets that facilitate the management tasks specific to public key certificates of X.509 format, which is the standard certificate format used by Windows certificate-based processes.

Get-QADLocalCertificateStore

Retrieve X.509 certificate stores held in a store location such as CurrentUser or LocalMachine.

Syntax

```
Get-QADLocalCertificateStore [[-StoreName] <String[]>] [[-StoreLocation]
<StoreLocation>]
```

Parameters

StoreLocation

Specify the location from which to retrieve a certificate store. The possible parameter values are:

- **CurrentUser** The store holds certificates for the current user. This is the default value.
- **LocalMachine** The store holds certificates for the local machine (all users).

StoreName

Specify the name of the certificate store to retrieve. Use a string array of names to retrieve more than one store at a time.

Detailed Description

Use this cmdlet to retrieve a certificate store by name. The output object can be used to identify the certificate store to search for certificates.

The cmdlet retrieves certificate stores from a specified store location. Certificate stores are used to save certificates on local computers. The certificate stores include:

- Physical stores, where certificates are physically stored on the local computer, in the system registry
- Logical stores, to group certificates together in functional categories, by using pointers to the physical stores

The cmdlet supports both physical and logical certificate stores, allowing a store to be retrieved from a particular location. Store locations are high-level containers that hold the certificate stores for the current user and for all users. Each system has two store locations: CurrentUser and LocalMachine (all users).

Examples

Example 1

Retrieve all the certificate stores that hold certificates for the current user:

```
C:\PS> Get-QADLocalCertificateStore
```

Example 2

Retrieve all the certificate stores that hold certificates for the local machine:

```
C:\PS> Get-QADLocalCertificateStore -StoreLocation LocalMachine
```

Example 3

Retrieve the certificate store for trusted root certification authorities (CAs) held in the LocalMachine store location:

```
C:\PS> Get-QADLocalCertificateStore Root -StoreLocation LocalMachine
```

Example 4

Retrieve two certificate stores with the specified names that hold certificates for the current user:

```
C:\PS> Get-QADLocalCertificateStore 'MyTrustedStore','MyUntrustedStore'
```

New-QADLocalCertificateStore

Create an X.509 certificate store in a store location such as CurrentUser or LocalMachine.

Syntax

```
New-QADLocalCertificateStore [-StoreName] <String> [[-StoreLocation] <StoreLocation>]  
[-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

StoreLocation

Specify the location in which to create a certificate store. Acceptable parameter values are:

- **CurrentUser** The store to hold certificates for the current user. This is the default value.
- **LocalMachine** The store to hold certificates for the local machine (all users).

StoreName

Specify the name to assign to the new certificate store.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to create a new certificate store. The output object can be used to identify the new store to which to add (import) certificates.

The cmdlet creates a certificate store in a specified store location. Certificate stores are physical stores in which certificates are saved and managed. Store locations are high-level containers that group the certificate stores for the current user and for all users. Each system has the CurrentUser store location and the LocalMachine (all users) store location.

Examples

Example 1

Create a certificate store called MyCert to hold certificates for the local machine (all users):

```
C:\PS> New-QADLocalCertificateStore "MyCert" -StoreLocation LocalMachine
```

Example 2

Create a certificate store called MyCert to hold certificates for the current user:

```
C:\PS> New-QADLocalCertificateStore "MyCert"
```

Remove-QADLocalCertificateStore

Delete X.509 certificate stores held in a store location such as CurrentUser or LocalMachine.

Syntax

```
Remove-QADLocalCertificateStore [-Store] <X509CertificateStoreUI> [-WhatIf] [-Confirm]
```

```
Remove-QADLocalCertificateStore [-StoreName] <String[]> [[-StoreLocation]  
<StoreLocation>] [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

Store

Use this parameter to supply an object representing the certificate store to remove. This could be an object output by the Get-QADLocalCertificateStore cmdlet (see examples).

StoreLocation

Specify the location from which to remove a certificate store. The possible parameter values are:

- **CurrentUser** The store holds certificates for the current user. This is the default value.
- **LocalMachine** The store holds certificates for the local machine (all users).

StoreName

Specify the name of the certificate store to remove. Use a string array of names to remove more than one store at a time.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to delete a certificate store. You can identify the certificate store either by name and location or by using an output object of the Get-QADLocalCertificateStore cmdlet (see examples).

The cmdlet allows you to specify a store location from which to remove certificate stores. Certificate stores are physical stores in which certificates are saved and managed. Store locations are high-level containers that group the certificate stores for the current user and for all users. Each system has the CurrentUser store location and the LocalMachine (all users) store location.

Examples

Example 1

Remove all certificate stores whose name begins with MyStore, from the CurrentUser store location:

```
C:\PS> Get-QADLocalCertificateStore MyStore* | Remove-QADLocalCertificateStore
```

In this command, the target certificate stores are retrieved by Get-QADLocalCertificateStore. Since the StoreLocation parameter is not supplied, the cmdlet retrieves certificate stores from the CurrentUser store location. The output objects are passed to Remove-QADLocalCertificateStore through the Store parameter, which is designed to accept parameter values from the pipeline.

Example 2

Remove all certificate stores whose name begins with MyStore, from the CurrentUser store location:

```
C:\PS> Remove-QADLocalCertificateStore MyStore*
```

In this command, the target certificate stores are identified by name and location through the StoreName and StoreLocation parameters. Since the StoreLocation parameter is not supplied, the cmdlet looks for certificate stores in the CurrentUser store location.

Get-QADCertificate

Retrieve X.509 certificates that match the desired conditions.

Syntax

```
Get-QADCertificate [-Store] <X509CertificateStoreUI> [-HasPrivateKey] [-FriendlyName
<String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>]
[-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey
<String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

```
Get-QADCertificate [-DirObj] <IGenericDirectoryObject> [-HasPrivateKey] [-FriendlyName
<String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>]
[-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey
<String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

```
Get-QADCertificate [-NativeCertificate] <X509Certificate2> [-HasPrivateKey]
[-FriendlyName <String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN
<String[]>] [-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>]
[-PublicKey <String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

```
Get-QADCertificate [-Signature] <Signature> [-HasPrivateKey] [-FriendlyName <String[]>]
[-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>] [-SubjectDN
<String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey <String[]>]
[-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>] [-SignatureAlgorithm
<String[]>] [-SubjectKeyIdentifier <String[]>] [-Template <String[]>] [-Version
<Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage <X509KeyUsageFlags>]
[-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage <String[]>] [-Expired] [-Valid]
[-CertificateAuthority] [-Revoked] [-PrivateKeyProtected] [-PrivateKeyExportable]
```

Parameters

AllEnhancedKeyUsages

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to retrieve, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to retrieve a certificate if the intended purposes of the certificate's key match all of the OIDs specified.

AllKeyUsages

Use this parameter to specify the key usage purpose for the certificates you want to retrieve. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusage.flags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet retrieves a certificate if the certificate's key is intended for each of the purposes defined by the members you specified.

AnyEnhancedKeyUsage

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to retrieve, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to retrieve a certificate if the intended purposes of the certificate's key match any of the OIDs specified.

AnyKeyUsage

Use this parameter to specify the key usage purpose for the certificates you want to retrieve. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusage.flags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet retrieves a certificate if the certificate's key is intended for any of the purposes defined by the members you specified.

CertificateAuthority

Supply this parameter to retrieve only certification authority (CA) certificates. (CA certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two certification authorities.) If you want to retrieve only the certificates that are not CA certificates, use the following syntax: -CertificateAuthority:\$false.

DirObj

Parameter value is an object representing the directory object, such as a user account, from which to retrieve certificates. To retrieve certificates that are assigned to a particular user in Active Directory, retrieve the corresponding user account by using Get-QADUser and then pass the output object to this parameter (see examples).

Expired

Supply this parameter to retrieve only expired certificates (a certificate is considered expired after the certificate's expiration date). If you want to retrieve only the certificates that are not expired, use the following syntax: -Expired:\$false.

FriendlyName

Use this parameter to specify the friendly name associated with the certificate to retrieve. You can supply an array of strings each of which represents the friendly name of a single certificate, to retrieve the certificates that have any of the specified names.

Friendly name is an optional property of a certificate that can be set on an as-needed basis. It is possible to assign a friendly name to a certificate so the certificate can be easily identified.

HasPrivateKey

Supply this parameter to retrieve only certificates containing a private key. With this parameter, the cmdlet retrieves a certificate only if the certificate has a private key associated with it. Without this parameter, the cmdlet does not consider the presence of a private key. If you want to retrieve only the certificates that do not contain a private key, use the following syntax: `-HasPrivateKey:$false`.

IssuedBy

Use this parameter to specify the name of the certification authority (CA) that issued the certificate to retrieve. You can supply an array of strings each of which represents the name of a single CA, to retrieve the certificates that were issued by any of the certification authorities specified.

IssuedTo

Use this parameter to specify the name of the principal to which the sought-for certificate was issued. You can supply an array of strings each of which represents a single principal's name, to retrieve the certificates that were issued to any of the principals specified.

IssuerDN

Use this parameter to specify the issuer distinguished name of the certificate to retrieve. You can supply an array of strings each of which represents the distinguished name of a single certificate's issuer, to retrieve the certificates issued by any of the issuers specified.

The issuer distinguished name identifies the certification authority (CA) that issued the certificate. A distinguished name consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

KeyAlgorithm

Use this parameter to specify the key algorithm information, in string format, for the certificate you want to retrieve. Parameter value is the object identifier (OID) or OID's friendly name that identifies the algorithm. You can specify an array of strings each of which identifies a certificate's key algorithm, to retrieve the certificates that use any of the specified key algorithms.

KeyAlgorithmParameters

Use this parameter to specify the hexadecimal string representing the key algorithm parameters of the certificate to retrieve. You can supply an array of strings each of which represents the key algorithm parameters of a single certificate, to retrieve the certificates that have any of the specified key algorithm parameters.

NativeCertificate

Parameter value is a native object provided by .NET Framework to represent X.509 certificates, such as an object of the X509Certificate2 class. The cmdlet retrieves the certificate data from that object if the certificate matches the conditions specified by the cmdlet parameters, and represents the certificate in the form of an object that could be recognized by ActiveRoles Management Shell cmdlets for certificate management.

PrivateKeyExportable

Supply this parameter to retrieve certificates containing an exportable private key. With this parameter, the cmdlet retrieves a certificate if the private key associated with the certificate can be exported. Without this parameter, the cmdlet does not consider whether the private key can be exported. If you want to retrieve certificates whose private key cannot be exported, use the following syntax: `-PrivateKeyExportable:$false`.

PrivateKeyProtected

Supply this parameter to retrieve certificates containing a protected private key. With this parameter, the cmdlet retrieves a certificate if the private key associated with the certificate is protected. Without this parameter, the cmdlet does not consider whether the private key is protected. If you want to retrieve certificates whose private key is not protected, use the following syntax: `-PrivateKeyProtected:$false`.

PublicKey

Use this parameter to specify the hexadecimal string representing the public key of the certificate to retrieve. You can supply an array of strings each of which represents the public key associated with a single certificate, to retrieve the certificates that contain any of the keys specified.

Revoked

Supply this parameter to retrieve only revoked certificates. If you want to retrieve only the certificates that are not revoked, use the following syntax: `-Revoked:$false`.

SerialNumber

Use this parameter to specify the serial number of the certificate to retrieve. You can supply an array of strings each of which represents the serial number of a single certificate, to retrieve the certificates that have any of the specified serial numbers.

The serial number of a certificate is a unique number assigned to the certificate by the certification authority (CA) that issued the certificate.

Signature

This parameter is intended to receive output objects of the Get-AuthenticodeSignature cmdlet. You can use this parameter, in conjunction with Get-AuthenticodeSignature, to retrieve certificates that were used to sign particular files: get information about the Authenticode signature in a file and then pass the corresponding object to the Signature parameter, thereby identifying the certificate to find.

SignatureAlgorithm

Use this parameter to specify the object identifier (OID) or OID's friendly name that identifies the type of the encryption algorithm used to create the signature of the certificate to retrieve. You can supply an array of strings each of which identifies a single certificate's signature algorithm, to retrieve the certificates that use any of the algorithms specified.

Store

Parameter value is an object that identifies the certificate store from which to retrieve certificates. Normally, this is an output object of the `Get-QADLocalCertificateStore` cmdlet.

SubjectDN

Use this parameter to specify the subject distinguished name of the certificate to retrieve. You can supply an array of strings each of which represents the distinguished name of a single certificate's subject, to retrieve the certificates issued to any of the subjects specified.

The subject distinguished name is a textual representation of the certificate's subject. This representation consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

SubjectKeyIdentifier

Use this parameter to specify the subject key identifier (SKI) of the certificate to retrieve. You can supply an array of strings each of which represents a single certificate's SKI encoded in hexadecimal format.

The subject key identifier can be used to differentiate between multiple public keys held by the certificate subject. The SKI value is typically an SHA-1 hash of the key.

Template

Use this parameter to specify the certificate template of the certificate to retrieve. Parameter value is the name of a certificate template. You can supply an array of strings each of which represents the name of a certificate template, to retrieve the certificates that are based on any of the templates specified.

Thumbprint

Use this parameter to specify the thumbprint of the certificate to retrieve. You can supply an array of strings each of which represents the thumbprint of a single certificate, to retrieve multiple certificates at a time.

The thumbprint is a hash value generated using the SHA-1 algorithm that uniquely identifies the certificate. As such, the thumbprint of a certificate is commonly used to find the certificate in a certificate store.

Valid

Supply this parameter to retrieve only valid certificates. If you want to retrieve only the certificates that are not valid, use the following syntax: `-Valid:$false`.

Version

Parameter value is the X.509 format version of the certificates to retrieve. For example, to search for X.509 version 3 certificates, supply the parameter value of 3. An array of numbers causes the cmdlet to retrieve certificates whose X.509 format version matches any of the numbers specified.

Detailed Description

Use this cmdlet to retrieve X.509 certificates from a certificate store or an Active Directory object. The cmdlet retrieves the certificates that satisfy the conditions you configure using the cmdlet parameters. Each of the output objects represents a certificate retrieved by this cmdlet, and can be passed to other cmdlets such as `Export-QADCertificate`, `Edit-QADCertificate`, or `Show-QADCertificate`.

Examples

Example 1

Retrieve all certificates from the Root certificate store in the CurrentUser store location:

```
C:\PS> Get-QADLocalCertificateStore Root | Get-QADCertificate
```

Example 2

Retrieve all certificates that are mapped to the specified user account in Active Directory:

```
C:\PS> Get-QADUser domainName\userName | Get-QADCertificate
```

Example 3

Retrieve all certificates issued by Microsoft or VeriSign that are mapped to user accounts in your Active Directory domain:

```
C:\PS> Get-QADUser | Get-QADCertificate -IssuerDN '*Microsoft*', '*VeriSign*'
```

Where-QADCertificate

Create a filter that determines which objects to pass along a command pipeline depending upon properties of X.509 certificates held in, or mapped to, input objects.

Syntax

```
Where-QADCertificate [-DirObj] <IGenericDirectoryObject> [-HasPrivateKey]
[-FriendlyName <String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN
<String[]>] [-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>]
[-PublicKey <String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

```
Where-QADCertificate [-Certificate] <X509CertificateUI> [-HasPrivateKey] [-FriendlyName
<String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>]
[-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey
<String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

```
Where-QADCertificate [-Store] <X509CertificateStoreUI> [-HasPrivateKey] [-FriendlyName
<String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>]
[-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey
<String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>] [-Template
<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage
<X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage
<String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable]
```

Parameters

AllEnhancedKeyUsages

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to match, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to take account of a certificate when filtering input objects if the intended purposes of the certificate's key match all of the OIDs specified.

AllKeyUsages

Use this parameter to specify the key usage purpose for the certificates you want the cmdlet to take account of when filtering input objects. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusageflags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet takes account of a certificate if the certificate's key is intended for each of the purposes defined by the members you specified.

AnyEnhancedKeyUsage

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to match, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to take account of a certificate when filtering input objects if the intended purposes of the certificate's key match any of the OIDs specified.

AnyKeyUsage

Use this parameter to specify the key usage purpose for the certificates you want the cmdlet to take account of when filtering input objects. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusageflags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet takes account of a certificate if the certificate's key is intended for any of the purposes defined by the members you specified.

Certificate

Use this parameter to supply the certificate objects representing the certificates to filter. This could be output objects of the Get-QADCertificate or Import-QADCertificate cmdlet (see examples). If the certificate represented by a given object matches the conditions specified by the cmdlet parameters, the object is allowed to pass; otherwise, the object is filtered out.

CertificateAuthority

Supply this parameter for the cmdlet to take account of only certification authority (CA) certificates. (CA certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two certification authorities.) If you want the cmdlet to take account of only the certificates that are not CA certificates, use the following syntax: -CertificateAuthority:\$false.

DirObj

Use this parameter to supply the Active Directory objects, such as user accounts, to filter. This could be output objects of the Get-QADUser cmdlet (see examples). If a given Active Directory object is associated with a certificate that matches the conditions specified by the cmdlet parameters, the object is allowed to pass; otherwise, the object is filtered out.

Expired

Supply this parameter for the cmdlet to take account of only expired certificates (a certificate is considered expired after the certificate's expiration date). If you want the cmdlet to take account of only the certificates that are not expired, use the following syntax: -Expired:\$false.

FriendlyName

Use this parameter to specify the friendly name associated with the certificate to match. You can supply an array of strings each of which represents the friendly name of a single certificate, for the cmdlet to take account of the certificates that have any of the specified names.

Friendly name is an optional property of a certificate that can be set on an as-needed basis. It is possible to assign a friendly name to a certificate so the certificate can be easily identified.

HasPrivateKey

Supply this parameter for the cmdlet to take account of only certificates containing a private key. With this parameter, the cmdlet takes account of a certificate only if the certificate has a private key associated with it. Without this parameter, the cmdlet does not consider the presence of a private key. If you want the cmdlet to take account of only the certificates that do not contain a private key, use the following syntax: -HasPrivateKey:\$false.

IssuedBy

Use this parameter to specify the name of the certification authority (CA) that issued the certificate to match. You can supply an array of strings each of which represents the name of a single CA, for the cmdlet to take account of the certificates that were issued by any of the certification authorities specified.

IssuedTo

Use this parameter to specify the name of the principal to which the sought-for certificate was issued. You can supply an array of strings each of which represents a single principal's name, for the cmdlet to take account of the certificates that were issued to any of the principals specified.

IssuerDN

Use this parameter to specify the issuer distinguished name of the certificate to match. You can supply an array of strings each of which represents the distinguished name of a single certificate's issuer, for the cmdlet to take account of the certificates issued by any of the issuers specified.

The issuer distinguished name identifies the certification authority (CA) that issued the certificate. A distinguished name consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

KeyAlgorithm

Use this parameter to specify the key algorithm information, in string format, for the certificate you want the cmdlet to take account of when filtering input objects. Parameter value is the object identifier (OID) or OID's friendly name that identifies the algorithm. You can specify an array of strings each of which identifies a certificate's key algorithm, for the cmdlet to take account of the certificates that use any of the specified key algorithms.

KeyAlgorithmParameters

Use this parameter to specify the hexadecimal string representing the key algorithm parameters of the certificate to match. You can supply an array of strings each of which represents the key algorithm parameters of a single certificate, for the cmdlet to take account of the certificates that have any of the specified key algorithm parameters.

PrivateKeyExportable

Supply this parameter for the cmdlet to take account of certificates containing an exportable private key. With this parameter, the cmdlet takes account of a certificate if the private key associated with the certificate can be exported. Without this parameter, the cmdlet does not consider whether the private key can be exported. If you want the cmdlet to take account of certificates whose private key cannot be exported, use the following syntax: `-PrivateKeyExportable:$false`.

PrivateKeyProtected

Supply this parameter for the cmdlet to take account of certificates containing a protected private key. With this parameter, the cmdlet takes account of a certificate if the private key associated with the certificate is protected. Without this parameter, the cmdlet does not consider whether the private key is protected. If you want the cmdlet to take account of certificates whose private key is not protected, use the following syntax: `-PrivateKeyProtected:$false`.

PublicKey

Use this parameter to specify the hexadecimal string representing the public key of the certificate to match. You can supply an array of strings each of which represents the public key associated with a single certificate, for the cmdlet to take account of the certificates that contain any of the keys specified.

Revoked

Supply this parameter for the cmdlet to take account of only revoked certificates. If you want the cmdlet to take account of only the certificates that are not revoked, use the following syntax: `-Revoked:$false`.

SerialNumber

Use this parameter to specify the serial number of the certificate to match. You can supply an array of strings each of which represents the serial number of a single certificate, for the cmdlet to take account of the certificates that have any of the specified serial numbers.

The serial number of a certificate is a unique number assigned to the certificate by the certification authority (CA) that issued the certificate.

SignatureAlgorithm

Use this parameter to specify the object identifier (OID) or OID's friendly name that identifies the type of the encryption algorithm used to create the signature of the certificate to match. You can supply an array of strings each of which identifies a single certificate's signature algorithm, for the cmdlet to take account of the certificates that use any of the algorithms specified.

Store

Use this parameter to supply the certificate store objects to filter. This could be output objects of the `Get-QADLocalCertificateStore` cmdlet (see examples). If the certificate store represented by a given object contains a certificate that matches the conditions specified by the cmdlet parameters, the object is allowed to pass; otherwise, the object is filtered out.

SubjectDN

Use this parameter to specify the subject distinguished name of the certificate to match. You can supply an array of strings each of which represents the distinguished name of a single certificate's subject, for the cmdlet to take account of the certificates issued to any of the subjects specified.

The subject distinguished name is a textual representation of the certificate's subject. This representation consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

SubjectKeyIdentifier

Use this parameter to specify the subject key identifier (SKI) of the certificate you want the cmdlet to take account of when filtering input objects. You can supply an array of strings each of which represents a single certificate's SKI encoded in hexadecimal format, for the cmdlet to take account of the certificates that have any of the specified subject key identifiers.

The subject key identifier can be used to differentiate between multiple public keys held by the certificate subject. The SKI value is typically an SHA-1 hash of the key.

Template

Use this parameter to specify the certificate template of the certificate to match. Parameter value is the name of a certificate template. You can supply an array of strings each of which represents the name of a certificate template, for the cmdlet to take account of the certificates that are based on any of the templates specified.

Thumbprint

Use this parameter to specify the thumbprint of the certificate to match. You can supply an array of strings each of which represents the thumbprint of a single certificate, for the cmdlet to take account of the certificates that have any of the specified thumbprints.

The thumbprint is a hash value generated using the SHA-1 algorithm that uniquely identifies the certificate. As such, the thumbprint of a certificate is commonly used to find the certificate in a certificate store.

Valid

Supply this parameter for the cmdlet to take account of only valid certificates. If you want the cmdlet to take account of only the certificates that are not valid, use the following syntax: `-Valid:$false`.

Version

Parameter value is the X.509 format version of the certificates to match. For example, if you want the cmdlet to take account of only X.509 version 3 certificates, supply the parameter value of 3. An array of numbers causes the cmdlet to take account of certificates whose X.509 format version matches any of the numbers specified.

Detailed Description

This cmdlet selects objects from the set of objects passed to it, based on properties of X.509 certificates held in, or mapped to, input objects. When the cmdlet receives an object, it checks to see whether the object contains, or is associated with, an X.509 certificate that matches the filter conditions specified by cmdlet parameters. If such a certificate exists, and is held in, or mapped to, an input object, the object is returned; otherwise, the object is ignored (filtered out).

An input object may represent a certificate store, an Active Directory object (for example, a user account), or a certificate. In case of a certificate store object, the cmdlet allows the object to pass if the certificate store contains a certificate that matches the filter conditions. In case of an Active Directory object, the cmdlet allows the object to pass if a certificate that matches the filter conditions is mapped to that object in Active Directory. In case of a certificate object, the cmdlet allows the object to pass if the certificate represented by that object matches the filter conditions.

Examples

Example 1

Retrieve all the certificate stores from the CurrentUser store location that contain any expired certificates:

```
C:\PS> Get-QADLocalCertificateStore | Where-QADCertificate -Expired
```

The output of this command is a set of objects each of which represents one of the certificate stores containing expired certificates.

Example 2

Retrieve all user accounts from your Active Directory domain that are associated with certificates issued by Microsoft or VeriSign:

```
C:\PS> Get-QADUser | Where-QADCertificate -IssuerDN '*Microsoft*', '*VeriSign*'
```

The output of this command is a set of objects each of which represents one of the user accounts to which the sought-for certificates are mapped in Active Directory.

Example 3

Create a collection of objects (\$cert) representing all the non-expired certificates found in the certificate files that are located in the specified folder (c:\cert); then, pass those objects to the Add-QADCertificate cmdlet to map the corresponding certificates to the specified user account in Active Directory:

```
C:\PS> $cert = dir c:\cert | Import-QADCertificate | Where-QADCertificate -Expired:$false
```

```
C:\PS> Get-QADUser domainName\userName | Add-QADCertificate $cert
```

Example 4

Retrieve users whose certificates in Active Directory have no key usage purpose specified:

```
C:\PS> Get-QADUser | Where-QADCertificate -AllKeyUsages None
```

Example 5

Retrieve users whose certificates in Active Directory have no key intended purpose specified:

```
C:\PS> Get-QADUser | Where-QADCertificate -AllEnhancedKeyUsages ''
```

Add-QADCertificate

Add X.509 certificates to a certificate store, or map X.509 certificates to a user account in Active Directory.

Syntax

```
Add-QADCertificate [-DirObj] <IGenericDirectoryObject> [-Certificate]
<X509CertificateUI[]> [-Control <Hashtable>] [-WhatIf] [-Confirm]
```

```
Add-QADCertificate [-Store] <X509CertificateStoreUI> [-Certificate]
<X509CertificateUI[]> [-WhatIf] [-Confirm]
```

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to add. This could be output objects of the Get-QADCertificate or Import-QADCertificate cmdlet (see examples).

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect if an Active Directory object passed to the DirObj parameter is retrieved through Quest One ActiveRoles. For example, you could retrieve the object by using Get-QADUser with the Proxy connection parameter. In this case, the request to update the corresponding user account is processed by Quest One ActiveRoles, so the Control parameter passes the request controls to Quest One ActiveRoles. If the input object is retrieved through a direct connection to the directory (the Proxy connection parameter was not used), the Control parameter has no effect.

DirObj

Parameter value is an object representing the Active Directory object, such as a user account, to which to map the certificates identified by the Certificate parameter. To map certificates to a particular user in Active Directory, retrieve the corresponding user account by using Get-QADUser and then pass the output object to this parameter (see examples).

Store

Parameter value is an object that identifies the certificate store to which to add the certificates identified by the Certificate parameter. Normally, this is an output object of the Get-QADLocalCertificateStore cmdlet (see examples).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add X.509 certificates to a certificate store or map X.509 certificates to an Active Directory object, such as a user account. The cmdlet takes an output object of the Get-QADCertificate or Import-QADCertificate cmdlet, and updates the specified certificate store or Active Directory object with the certificate represented by that object.

Note: If the certificate data to add to a user account in Active Directory contains private key information (for example, the data being imported from a .pfx file), the private key information is disregarded since it cannot be stored in Active Directory.

Examples

Example 1

Create a collection of objects (\$cert) representing all the certificates found in the TrustedPublisher certificate store for the current user; then, pass those objects to the Add-QADCertificate cmdlet to map the certificates to the specified user account in Active Directory:

```
C:\PS> $cert = Get-QADLocalCertificateStore TrustedPublisher | Get-QADCertificate
```

```
C:\PS> Get-QADUser domainName\userName | Add-QADCertificate -Certificate $cert
```

As a result, the certificates listed in the \$cert variable are mapped to the user account identified by domainName\userName.

Example 2

Create a collection of objects (\$cert) representing the certificates found in the certificate files that are located in the specified folder (c:\cert); then, pass those objects to the Add-QADCertificate cmdlet to identify the certificates to map to the specified user account.

```
C:\PS> $cert = dir c:\cert | Import-QADCertificate
```

```
C:\PS> Get-QADUser domainName\userName | Add-QADCertificate -Certificate $cert
```

As a result, the certificates listed in the \$cert variable are mapped to the user account identified by domainName\userName.

Example 3

Create a collection of objects (\$cert) representing the certificates found in the certificate files that are located in the specified folder (c:\cert), apply a filter using Where-QADCertificate to ensure that the collection lists only the certificates issued by Microsoft or VeriSign, and then, pass the certificate objects to the Add-QADCertificate cmdlet to identify the certificates to map to the specified user account.

```
C:\PS> $cert = dir c:\cert | Import-QADCertificate | Where-QADCertificate -IssuerDN *Microsoft*,*VeriSign*
```

```
C:\PS> Get-QADUser domainName\userName | Add-QADCertificate -Certificate $cert
```

As a result, the certificates listed in the \$cert variable are mapped to the user account identified by domainName\userName.

Example 4

Add the certificate from file c:\cert.cer to the Trusted Root Certification Authorities certificate store in the CurrentUser store location:

```
C:\PS> $store = Get-QADLocalCertificateStore Root
```

```
C:\PS> Import-QADCertificate c:\cert.cer | Add-QADCertificate -Store $store
```

The first command uses Get-QADLocalCertificateStore to populate the \$store variable with the object representing the Trusted Root Certification Authorities certificate store. In the second command, Import-QADCertificate creates a certificate object based on the certificate data found in file c:\cert.cer and passes that object to Add-QADCertificate, causing the certificate to be added to the certificate store identified by the object held in the \$store variable.

Import-QADCertificate

Create a certificate object and populate the object with the X.509 certificate data from a byte array or a file.

Syntax

```
Import-QADCertificate [-FileName] <String> [-ImportFlags <X509KeyStorageFlags>]
[-Password <SecureString>]
```

```
Import-QADCertificate -File <FileInfo> [-ImportFlags <X509KeyStorageFlags>] [-Password
<SecureString>]
```

```
Import-QADCertificate -RawData <Object> [-ImportFlags <X509KeyStorageFlags>] [-Password
<SecureString>]
```

Parameters

File

This parameter is intended to receive a FileInfo object that identifies the file containing the X.509 certificate data to import. If you need to supply the path and name of the file, use the FileName parameter.

With this parameter, the cmdlet takes a certificate file that represents an X.509 certificate, and populates the output object with the certificate the file contains. The cmdlet supports all the certificate file types that could be created by Export-QADCertificate, including DER- or Base64-encoded X.509 certificate files (.CER), PKCS7 (.P7B) and PKCS12 (.PFX) certificate files, and Microsoft serialized certificate store (.SST) files.

FileName

Use this parameter to supply the path and name of the file containing the X.509 certificate data to import. The path can be an absolute path, such as C:\MyCertificates\Cert.cer, or a relative path. If the path or file name includes spaces, enclose the parameter value in quotation marks.

With this parameter, the cmdlet takes a certificate file that represents an X.509 certificate, and populates the output object with the certificate the file contains. The cmdlet supports all the certificate file types that could be created by Export-QADCertificate, including DER- or Base64-encoded X.509 certificate files (.CER), PKCS7 (.P7B) and PKCS12 (.PFX) certificate files, and Microsoft serialized certificate store (.SST) files.

ImportFlags

Use this parameter to specify where and how to import the private key associated with the certificate. Parameter value can be a member of the X509KeyStorageFlags enumeration, such as UserKeySet, MachineKeySet, Exportable or UserProtected. For a complete list of the enumeration members, see the "X509KeyStorageFlags Enumeration" article in Microsoft's .NET Framework Class Library at msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keystorageflags.aspx

Password

Use this parameter to specify the password required to access the X.509 certificate data.

RawData

Use this parameter to specify the object that contains the X.509 certificate data to import. This could be, for example, an output object of the Get-Content cmdlet.

Detailed Description

Use this cmdlet to create a certificate object populated with the X.509 certificate data found in a byte array or a file. The cmdlet can take an output object of the Get-Content cmdlet containing the data found in a certificate file, and return a certificate object that represents the corresponding certificate. Another option is to have the cmdlet import the certificate data directly from a certificate file specified (see examples).

Examples

Example 1

Create a certificate object that represents a certificate found in the specified X.509 certificate file (c:\cert.cer):

```
C:\PS> Import-QADCertificate c:\cert.cer
```

Example 2

Create a certificate object that represents a certificate found in the specified password-protected certificate file (c:\cert.pfx) containing the certificate's private key data:

```
C:\PS> Import-QADCertificate c:\cert.pfx -Password (ConvertTo-SecureString 'P@ssw0rd' -asplaintext -force)
```

Example 3

Create a set of certificate objects each of which represents one of the certificates found in the X.509 certificate files that are located in the specified folder (c:\cert):

```
C:\PS> dir c:\cert | Import-QADCertificate
```

Example 4

Create a certificate object that represents a certificate found in the specified password-protected certificate file (c:\cert.pfx) containing the certificate's private key data, and configure the certificate in the certificate object to enable strong protection for the private key and mark the private key as exportable:

```
C:\PS> Import-QADCertificate c:\cert.pfx -Password (ConvertTo-SecureString 'P@ssw0rd' -asplaintext -force) -ImportFlags UserProtected,Exportable
```

Show-QADCertificate

Display a textual representation of an X.509 certificate.

Syntax

```
Show-QADCertificate [[-Certificate] <X509CertificateUI[]>]
```

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to display. This could be output objects of the Get-QADCertificate or Import-QADCertificate cmdlet (see examples).

Detailed Description

Use this cmdlet to view X.509 certificates in a text format. The cmdlet takes an output object of the Get-QADCertificate or Import-QADCertificate cmdlet, and displays a textual representation of the certificate represented by that object.

Examples

Example 1

Retrieve the certificates that are mapped to the specified user account in Active Directory, and view a textual representation of each certificate:

```
C:\PS> Get-QADUser domainName\userName | Get-QADcertificate | Show-QADCertificate
```

Edit-QADCertificate

Open a dialog box that contains the properties of an X.509 certificate.

Syntax

```
Edit-QADCertificate [-Certificate] <X509CertificateUI[]>
```

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to edit. This could be output objects of the Get-QADCertificate or Import-QADCertificate cmdlet (see examples).

Detailed Description

Use this cmdlet to display a dialog box with detailed information about the specified X.509 certificate. The cmdlet takes an output object of the Get-QADCertificate or Import-QADCertificate cmdlet, and opens a dialog box that contains the properties of the certificate.

Examples

Example 1

Retrieve the certificates that are mapped to the specified user account in Active Directory, and then, for each certificate, open a dialog box that contains the properties of the certificate:

```
C:\PS> Get-QADUser domainName\userName | Get-QADcertificate | Edit-QADCertificate
```

Export-QADCertificate

Export an X.509 certificate to a byte array or a file.

Syntax

```
Export-QADCertificate [-Certificate] <X509CertificateUI[]> [-Format <ExportFormat>]  
[-Password <SecureString>] [-Mode <ExportMode>] [-Encoding <CertificateEncoding>] [-File  
<String>]
```

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to export. This could be output objects of the Get-QADCertificate cmdlet (see examples).

Encoding

Use this parameter to specify how to encode the export data. The possible parameter values are:

- **Binary** The export data is a pure binary sequence
- **Base64** The export data is base64-encoded

The default parameter value is Binary for the Pkcs12 (Pfx) export format and Base64 for any other export format.

File

Use this parameter to specify the path and name of a file to which you want to export the certificates. The path can be an absolute path, such as C:\MyCertificates\Cert.cer, or a relative path. If the path or file name includes spaces, enclose the parameter value in quotation marks.

Format

Use this parameter to specify how to format the export data. The possible parameter values are the following members of the X509ContentType enumeration:

- **Cert**
- **SerializedCert**
- **Pfx**
- **Pkcs12** (same as **Pfx**)
- **SerializedStore**
- **Pkcs7**

If this parameter is omitted, the Cert format is used. For descriptions of these enumeration members, see the "X509ContentType Enumeration" article in Microsoft's .NET Framework Class Library at msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509contenttype.aspx

Mode

Use this parameter to specify whether you want the export data to include a single certificate, the certificate chain (that is, all certificates in the certification path for a given certificate), or a collection of certificates (see examples). The corresponding parameter values are:

- **Single** Exports each input certificate separately
- **Chain** Exports the entire certificate chain for each input certificate; not supported for the Cert and SerializedCert export formats
- **Collection** Exports all input certificates as a single bulk; not supported for the Cert and SerializedCert export formats

Password

Use this parameter to specify the password required to access the X.509 certificate export data. A password is required to export a certificate with both the public and private keys. In this case the certificate should be exported using the Pkcs12 (Pfx) export format.

Detailed Description

Use this cmdlet to export the X.509 certificate represented by a given certificate object, to a byte array or a file using the specified format. The cmdlet can take an output object of the Get-QADCertificate cmdlet, and return a byte array containing the certificate data found in that object. Another option is to have the cmdlet export the certificate directly to a file specified (see examples).

Examples

Example 1

```
C:\PS> Get-QADUser domainName\userName | Get-QADCertificate | %{Export-QADCertificate $_
-File "c:\cert\$($_.IssuedTo)-$($_.Thumbprint).cer"} | Out-Null
```

In this command: Get-QADCertificate generates certificate objects representing the X.509 certificates mapped to the specified user in Active Directory; Export-QADCertificate exports the certificate objects to byte arrays; and Select-Object passes the export data along with the file names to the Set-Content cmdlet which saves the export data into the files. As a result, each of the certificates is exported to a separate file in the c:\cert folder, with the file name set to the thumbprint value of the certificate.

Example 2

```
C:\PS> Get-QADLocalCertificateStore MyStore | Get-QADCertificate |
%{Export-QADCertificate $_ -Format Pkcs7 -Mode Chain -File
"c:\cert\$($_.SerialNumber).p7b"}
```

In this command: Get-QADLocalCertificateStore retrieves a certain certificate store by name from the CurrentUser store location and passes the corresponding object to Get-QADCertificate; Get-QADCertificate retrieves the certificates from that store and passes the certificate objects to Export-QADCertificate; for each certificate object, Export-QADCertificate exports the entire certificate chain of the corresponding certificate to a separate file using the Pkcs7 export format, with the file name composed of the certificate's serial number.

Example 3

```
C:\PS> Get-QADLocalCertificateStore MyStore | Get-QADCertificate | Export-QADCertificate
-Mode Collection -Format Pfx -Password (ConvertTo-SecureString <Password> -asplaintext
-force) -File c:\MyCerts.pfx
```

In this command: Get-QADLocalCertificateStore retrieves a certain certificate store by name from the CurrentUser store location and passes the corresponding object to Get-QADCertificate; Get-QADCertificate retrieves the certificates from that store and passes the certificate objects to Export-QADCertificate; Export-QADCertificate exports all the certificates, along with their private keys, to a single file using the Pfx export format. This export operation requires the export data to be protected by a password, so the Password parameter is used to set a password.

Remove-QADCertificate

Remove X.509 certificates that match the desired conditions.

Syntax

```
Remove-QADCertificate [-DirObj] <IGenericDirectoryObject> [[-Certificate]
<X509CertificateUI[]>] [-Control <Hashtable>] [-HasPrivateKey] [-FriendlyName
<String[]>] [-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>]
[-SubjectDN <String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey
<String[]>] [-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>]
[-SignatureAlgorithm <String[]>] [-SubjectKeyIdentifier <String[]>]
[-Template<String[]>] [-Version <Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>]
[-AnyKeyUsage <X509KeyUsageFlags>] [-AllEnhancedKeyUsages <String[]>]
[-AnyEnhancedKeyUsage <String[]>] [-Expired] [-Valid] [-CertificateAuthority] [-Revoked]
[-PrivateKeyProtected] [-PrivateKeyExportable] [-WhatIf] [-Confirm]
```

```
Remove-QADCertificate [-Store] <X509CertificateStoreUI> [[-Certificate]
<X509CertificateUI[]>] [-HasPrivateKey] [-FriendlyName <String[]>] [-IssuedTo
<String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>] [-SubjectDN <String[]>]
[-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey <String[]>]
[-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>] [-SignatureAlgorithm
<String[]>] [-SubjectKeyIdentifier <String[]>] [-Template <String[]>] [-Version
<Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage <X509KeyUsageFlags>]
[-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage <String[]>] [-Expired] [-Valid]
[-CertificateAuthority] [-Revoked] [-PrivateKeyProtected] [-PrivateKeyExportable]
[-WhatIf] [-Confirm]
```

Parameters

AllEnhancedKeyUsages

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to remove, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to remove a certificate if the intended purposes of the certificate's key match all of the OIDs specified.

AllKeyUsages

Use this parameter to specify the key usage purpose for the certificates you want to remove. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusage.flags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet removes a certificate if the certificate's key is intended for each of the purposes defined by the members you specified.

AnyEnhancedKeyUsage

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate to remove, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article "IX509ExtensionEnhancedKeyUsage Interface" at <http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to remove a certificate if the intended purposes of the certificate's key match any of the OIDs specified.

AnyKeyUsage

Use this parameter to specify the key usage purpose for the certificates you want to remove. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusage.flags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet removes a certificate if the certificate's key is intended for any of the purposes defined by the members you specified.

Certificate

Use this parameter to specify the certificate objects representing the certificates to remove. This could be output objects of the Get-QADCertificate or Import-QADCertificate cmdlet (see examples).

CertificateAuthority

Supply this parameter to remove only certification authority (CA) certificates. (CA certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two certification authorities.) If you want to remove only the certificates that are not CA certificates, use the following syntax: -CertificateAuthority:\$false.

Confirm

Prompts you for confirmation before executing the command.

Control

Use this parameter to pass request controls (in-controls) to Quest One ActiveRoles as part of an operation request. In Quest One ActiveRoles, request controls are used to send extra information along with an operation request, to control how Quest One ActiveRoles performs the request.

The parameter value is a hash table that defines the names and values of the request controls to be passed to Quest One ActiveRoles. The parameter syntax is as follows:

```
-Control @{<name> = <value>; [<name> = <value>] ...}
```

In this syntax, each of the name-value pairs is the name and the value of a single control. For instructions on how to create and use hash tables, see topic "about_associative_array" or "about_hash_tables" in Windows PowerShell Help. For information about Quest One ActiveRoles request controls, refer to Quest One ActiveRoles SDK documentation.

Note that this parameter only has an effect if an Active Directory object passed to the DirObj parameter is retrieved through Quest One ActiveRoles. For example, you could retrieve the object by using Get-QADUser with the Proxy connection parameter. In this case, the request to update the corresponding user account is processed by Quest One ActiveRoles, so the Control parameter passes the request controls to Quest One ActiveRoles. If the input object is retrieved through a direct connection to the directory (the Proxy connection parameter was not used), the Control parameter has no effect.

DirObj

Parameter value is an object representing the directory object, such as a user account, from which to remove certificates. To remove certificates that are assigned to a particular user in Active Directory, retrieve the corresponding user account by using Get-QADUser and then pass the output object to this parameter (see examples).

Expired

Supply this parameter to remove only expired certificates (a certificate is considered expired after the certificate's expiration date). If you want to remove only the certificates that are not expired, use the following syntax: -Expired:\$false.

FriendlyName

Use this parameter to specify the friendly name associated with the certificate to remove. You can supply an array of strings each of which represents the friendly name of a single certificate, to remove the certificates that have any of the specified names.

Friendly name is an optional property of a certificate that can be set on an as-needed basis. It is possible to assign a friendly name to a certificate so the certificate can be easily identified.

HasPrivateKey

Supply this parameter to remove only certificates containing a private key. With this parameter, the cmdlet removes a certificate only if the certificate has a private key associated with it. Without this parameter, the cmdlet does not consider the presence of a private key. If you want to remove only the certificates that do not contain a private key, use the following syntax: -HasPrivateKey:\$false.

IssuedBy

Use this parameter to specify the name of the certification authority (CA) that issued the certificate to remove. You can supply an array of strings each of which represents the name of a single CA, to remove the certificates that were issued by any of the certification authorities specified.

IssuedTo

Use this parameter to specify the name of the principal to which the sought-for certificate was issued. You can supply an array of strings each of which represents a single principal's name, to remove the certificates that were issued to any of the principals specified.

IssuerDN

Use this parameter to specify the issuer distinguished name of the certificate to remove. You can supply an array of strings each of which represents the distinguished name of a single certificate's issuer, to remove the certificates issued by any of the issuers specified.

The issuer distinguished name identifies the certification authority (CA) that issued the certificate. A distinguished name consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

KeyAlgorithm

Use this parameter to specify the key algorithm information, in string format, for the certificate you want to remove. Parameter value is the object identifier (OID) or OID's friendly name that identifies the algorithm. You can specify an array of strings each of which identifies a certificate's key algorithm, to remove the certificates that use any of the specified key algorithms.

KeyAlgorithmParameters

Use this parameter to specify the hexadecimal string representing the key algorithm parameters of the certificate to remove. You can supply an array of strings each of which represents the key algorithm parameters of a single certificate, to remove the certificates that have any of the specified key algorithm parameters.

PrivateKeyExportable

Supply this parameter to remove certificates containing an exportable private key. With this parameter, the cmdlet removes a certificate if the private key associated with the certificate can be exported. Without this parameter, the cmdlet does not consider whether the private key can be exported. If you want to remove certificates whose private key cannot be exported, use the following syntax: `-PrivateKeyExportable:$false`.

PrivateKeyProtected

Supply this parameter to remove certificates containing a protected private key. With this parameter, the cmdlet removes a certificate if the private key associated with the certificate is protected. Without this parameter, the cmdlet does not consider whether the private key is protected. If you want to remove certificates whose private key is not protected, use the following syntax: `-PrivateKeyProtected:$false`.

PublicKey

Use this parameter to specify the hexadecimal string representing the public key of the certificate to remove. You can supply an array of strings each of which represents the public key associated with a single certificate, to remove the certificates that contain any of the keys specified.

Revoked

Supply this parameter to remove only revoked certificates. If you want to remove only the certificates that are not revoked, use the following syntax: `-Revoked:$false`.

SerialNumber

Use this parameter to specify the serial number of the certificate to remove. You can supply an array of strings each of which represents the serial number of a single certificate, to remove the certificates that have any of the specified serial numbers.

The serial number of a certificate is a unique number assigned to the certificate by the certification authority (CA) that issued the certificate.

SignatureAlgorithm

Use this parameter to specify the object identifier (OID) or OID's friendly name that identifies the type of the encryption algorithm used to create the signature of the certificate to remove. You can supply an array of strings each of which identifies a single certificate's signature algorithm, to remove the certificates that use any of the algorithms specified.

Store

Parameter value is an object that identifies the certificate store from which to remove certificates. Normally, this is an output object of the Get-QADLocalCertificateStore cmdlet.

SubjectDN

Use this parameter to specify the subject distinguished name of the certificate to remove. You can supply an array of strings each of which represents the distinguished name of a single certificate's subject, to remove the certificates issued to any of the subjects specified.

The subject distinguished name is a textual representation of the certificate's subject. This representation consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

SubjectKeyIdentifier

Use this parameter to specify the subject key identifier (SKI) of the certificate to remove. You can supply an array of strings each of which represents a single certificate's SKI encoded in hexadecimal format.

The subject key identifier can be used to differentiate between multiple public keys held by the certificate subject. The SKI value is typically an SHA-1 hash of the key.

Template

Use this parameter to specify the certificate template of the certificate to remove. Parameter value is the name of a certificate template. You can supply an array of strings each of which represents the name of a certificate template, to remove the certificates that are based on any of the templates specified.

Thumbprint

Use this parameter to specify the thumbprint of the certificate to remove. You can supply an array of strings each of which represents the thumbprint of a single certificate, to remove multiple certificates at a time.

The thumbprint is a hash value generated using the SHA-1 algorithm that uniquely identifies the certificate. As such, the thumbprint of a certificate is commonly used to find the certificate in a certificate store.

Valid

Supply this parameter to remove only valid certificates. If you want to remove only the certificates that are not valid, use the following syntax: -Valid:\$false.

Version

Parameter value is the X.509 format version of the certificates to remove. For example, to remove X.509 version 3 certificates, supply the parameter value of 3. An array of numbers causes the cmdlet to remove certificates whose X.509 format version matches any of the numbers specified.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove X.509 certificates from a certificate store or an Active Directory object. The cmdlet removes the certificates that satisfy the conditions you configure using the cmdlet parameters.

Examples

Example 1

Remove all certificates from the specified user account in Active Directory:

```
C:\PS> Get-QADUser domainName\userName | Remove-QADCertificate
```

Example 2

Remove all expired certificates from all the certificate stores held in the CurrentUser store location:

```
C:\PS> Get-QADLocalCertificateStore | Remove-QADCertificate -Expired
```

Example 3

Remove all certificates issued by Microsoft or VeriSign from all user accounts in your Active Directory domain:

```
C:\PS> Get-QADUser | Remove-QADCertificate -IssuerDN '*Microsoft*', '*VeriSign*'
```

Example 4

Create a collection of objects (\$cert) representing the certificates found in the X.509 certificate files that are located in the specified folder (c:\cert); then, pass those objects to the Remove-QADCertificate cmdlet to remove the corresponding certificates from the specified user account:

```
C:\PS> $cert = dir c:\cert | Import-QADCertificate
```

```
C:\PS> Get-QADUser domainName\userName | Remove-QADCertificate $cert
```

Remove-QADPrivateKey

Delete the private keys that correspond to the X.509 certificates that match the desired conditions.

Syntax

```
Remove-QADPrivateKey [-Store] <X509CertificateStoreUI> [-FriendlyName <String[]>]
[-IssuedTo <String[]>] [-IssuedBy <String[]>] [-IssuerDN <String[]>] [-SubjectDN
<String[]>] [-SerialNumber <String[]>] [-Thumbprint <String[]>] [-PublicKey <String[]>]
[-KeyAlgorithm <String[]>] [-KeyAlgorithmParameters <String[]>] [-SignatureAlgorithm
<String[]>] [-SubjectKeyIdentifier <String[]>] [-Template <String[]>] [-Version
<Int32[]>] [-AllKeyUsages <X509KeyUsageFlags>] [-AnyKeyUsage <X509KeyUsageFlags>]
[-AllEnhancedKeyUsages <String[]>] [-AnyEnhancedKeyUsage <String[]>] [-Expired] [-Valid]
[-CertificateAuthority] [-Revoked] [-PrivateKeyProtected] [-PrivateKeyExportable]
[-WhatIf] [-Confirm]
```

Parameters

AllEnhancedKeyUsages

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate whose private key you want to delete, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article

"IX509ExtensionEnhancedKeyUsage Interface" at
<http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to delete the private key associated with a certificate if the intended purposes of the certificate's key match all of the OIDs specified.

AllKeyUsages

Use this parameter to specify the key usage purpose for the certificates whose private keys you want to delete. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at
<http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusage.flags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet deletes the private key associated with a certificate if the certificate's key is intended for each of the purposes defined by the members you specified.

AnyEnhancedKeyUsage

Use this parameter to specify the object identifiers (OIDs) that indicate the intended purposes of the public key contained in the certificate whose private key you want to delete, in addition to or in place of the key usage setting. Parameter value can be one or more OIDs or OID friendly names separated by commas. A list of some enhanced key usage OIDs can be found in Microsoft's article

"IX509ExtensionEnhancedKeyUsage Interface" at
<http://msdn.microsoft.com/en-us/library/aa378132.aspx>

This parameter causes the cmdlet to delete the private key associated with a certificate if the intended purposes of the certificate's key match any of the OIDs specified.

AnyKeyUsage

Use this parameter to specify the key usage purpose for the certificates whose private keys you want to delete. Parameter value can be any member of the X509KeyUsageFlags enumeration, such as EncipherOnly or DigitalSignature. For a complete list of the enumeration members, see the "X509KeyUsageFlags Enumeration" article in Microsoft's .NET Framework Class Library at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keyusageflags.aspx>

You can supply multiple members as an attribute value, separating them by commas. In this case, the cmdlet deletes the private key associated with a certificate if the certificate's key is intended for any of the purposes defined by the members you specified.

CertificateAuthority

Supply this parameter to delete only the private keys that are associated with certification authority (CA) certificates. (CA certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two certification authorities.) If you want to delete only the private keys associated with the certificates that are not CA certificates, use the following syntax: -CertificateAuthority:\$false.

Confirm

Prompts you for confirmation before executing the command.

Expired

Supply this parameter to delete only the private keys that are associated with expired certificates (a certificate is considered expired after the certificate's expiration date). If you want to delete only the private keys that are associated with the certificates that are not expired, use the following syntax: -Expired:\$false.

FriendlyName

Use this parameter to specify the friendly name associated with the certificate whose private key you want to delete. You can supply an array of strings each of which represents the friendly name of a single certificate, to delete the private keys corresponding to the certificates that have any of the specified names.

Friendly name is an optional property of a certificate that can be set on an as-needed basis. It is possible to assign a friendly name to a certificate so the certificate can be easily identified.

IssuedBy

Use this parameter to specify the name of the certification authority (CA) that issued the certificate whose private key you want to delete. You can supply an array of strings each of which represents the name of a single CA, to delete the private keys corresponding to the certificates that were issued by any of the certification authorities specified.

IssuedTo

Use this parameter to specify the name of the principal to which the sought-for private key was issued. You can supply an array of strings each of which represents a single principal's name, to delete the private keys corresponding to the certificates that were issued to any of the principals specified.

IssuerDN

Use this parameter to specify the issuer distinguished name of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the distinguished name of a single certificate's issuer, to delete the private keys that correspond to the certificates issued by any of the issuers specified.

The issuer distinguished name identifies the certification authority (CA) that issued the certificate. A distinguished name consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

KeyAlgorithm

Use this parameter to specify the key algorithm information, in string format, for the certificate whose private key you want to delete. Parameter value is the object identifier (OID) or OID's friendly name that identifies the algorithm. You can specify an array of strings each of which identifies a certificate's key algorithm, to delete the private keys corresponding to the certificates that use any of the specified key algorithms.

KeyAlgorithmParameters

Use this parameter to specify the hexadecimal string representing the key algorithm parameters of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the key algorithm parameters of a single certificate, to delete the private keys corresponding to the certificates that have any of the specified key algorithm parameters.

PrivateKeyExportable

Supply this parameter to delete the private keys associated with the certificates that have their private key marked as exportable. With this parameter, the cmdlet only the private keys that can be exported. Without this parameter, the cmdlet does not consider whether a private key can be exported. If you want to delete only private keys that cannot be exported, use the following syntax:
-PrivateKeyExportable:\$false.

PrivateKeyProtected

Supply this parameter to delete the private keys associated with the certificates that have their private key marked as protected. With this parameter, the cmdlet deletes only the protected private keys. Without this parameter, the cmdlet does not consider whether a private key is protected. If you want to delete only private keys that are not protected, use the following syntax: -PrivateKeyProtected:\$false.

PublicKey

Use this parameter to specify the hexadecimal string representing the public key of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the public key associated with a single certificate, to remove the private keys corresponding to the certificates that contain any of the public keys specified.

Revoked

Supply this parameter to delete only private keys corresponding to revoked certificates. If you want to remove only private keys corresponding to the certificates that are not revoked, use the following syntax: -Revoked:\$false.

SerialNumber

Use this parameter to specify the serial number of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the serial number of a single certificate, to delete the private keys corresponding to the certificates that have any of the specified serial numbers.

The serial number of a certificate is a unique number assigned to the certificate by the certification authority (CA) that issued the certificate.

SignatureAlgorithm

Use this parameter to specify the object identifier (OID) or OID's friendly name that identifies the type of the encryption algorithm used to create the signature of the certificate whose private key you want to delete. You can supply an array of strings each of which identifies a single certificate's signature algorithm, to delete the private keys corresponding to the certificates that use any of the algorithms specified.

Store

Parameter value is an object that identifies the certificate store that holds the certificate whose private key you want to delete. Normally, this is an output object of the `Get-QADLocalCertificateStore` cmdlet.

SubjectDN

Use this parameter to specify the subject distinguished name of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the distinguished name of a single certificate's subject, to delete the private keys corresponding to the certificates issued to any of the subjects specified.

The subject distinguished name is a textual representation of the certificate's subject. This representation consists of name attributes, for example, "CN=Name,OU=OrgUnit,C=US".

SubjectKeyIdentifier

Use this parameter to specify the subject key identifier (SKI) of the certificate whose private key you want to delete. You can supply an array of strings each of which represents a single certificate's SKI encoded in hexadecimal format, to delete the private keys that correspond to the certificates with the specified subject key identifiers.

The subject key identifier can be used to differentiate between multiple public keys held by the certificate subject. The SKI value is typically an SHA-1 hash of the key.

Template

Use this parameter to specify the certificate template of the certificate whose private key you want to delete. Parameter value is the name of a certificate template. You can supply an array of strings each of which represents the name of a certificate template, to delete the private keys for the certificates that are based on any of the templates specified.

Thumbprint

Use this parameter to specify the thumbprint of the certificate whose private key you want to delete. You can supply an array of strings each of which represents the thumbprint of a single certificate, to delete the private keys for multiple certificates at a time.

The thumbprint is a hash value generated using the SHA-1 algorithm that uniquely identifies the certificate. As such, the thumbprint of a certificate is commonly used to find the certificate in a certificate store.

Valid

Supply this parameter to delete only private keys associated with valid certificates. If you want to delete private keys for only the certificates that are not valid, use the following syntax: `-Valid:$false`.

Version

Parameter value is the X.509 format version of the certificate whose private key you want to delete. For example, to delete the private key for an X.509 version 3 certificate, supply the parameter value of 3. An array of numbers causes the cmdlet to remove private keys for certificates whose X.509 format version matches any of the numbers specified.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

For a given certificate held in a local certificate store, you can use this cmdlet to delete the private key that corresponds to the certificate, from the local computer. The cmdlet deletes the private key for each of the certificates that satisfy the conditions you configure using the cmdlet parameters. A typical use of this cmdlet is to delete the certificate's private key after exporting a certificate (see examples).

Examples

Example 1

Export certificates and then delete the private keys that correspond to those certificates:

```
C:\PS> $store = Get-QADLocalCertificateStore MyStore
C:\PS> $store | Get-QADCertificate -IssuedTo 'John Smith' | Export-QADCertificate -Format
Pfx -Mode Collection -Password (ConvertTo-SecureString <Password> -asplaintext -force)
-File 'John Smith.pfx' | Out-Null
C:\PS> $store | Remove-QADPrivateKey -IssuedTo 'John Smith'
```

In this example: `Get-QADLocalCertificateStore` retrieves the certificate store named `MyStore`, from the `CurrentUser` store location; `Get-QADCertificate` retrieves the certificates from that store that are issued to John Smith, and passes the certificate objects to `Export-QADCertificate`, which exports the certificates, along with their private keys, to a single file using the Pfx export format; then, `Remove-QADPrivateKey` deletes the private keys that correspond to the exported certificates. Note that the export operation requires the export data to be protected by a password, so the `Password` parameter of the `Export-QADCertificate` cmdlet is used to set a password.

Get-QADCertificateRevocationList

Retrieve certificate revocation lists from a certificate store or Active Directory.

Syntax

```
Get-QADCertificateRevocationList [-Store] <X509CertificateStoreUI>
```

```
Get-QADCertificateRevocationList [-DirObj] <IGenericDirectoryObject>
```

Parameters

DirObj

Parameter value is an object representing the CRL distribution point (cRLDistributionPoint) object from which to retrieve certificate revocation list. To retrieve the certificate revocation lists that are published to a particular CRL distribution point in Active Directory, you could first retrieve the corresponding cRLDistributionPoint objects by using Get-QADPKIObject and then pass the output objects to this parameter (see examples).

Store

Parameter value is an object that identifies the certificate store from which to retrieve certificate revocation lists. Normally, this is an output object of the Get-QADLocalCertificateStore cmdlet (see examples).

Detailed Description

Use this cmdlet to retrieve certificate revocation lists from a particular location, which could be either a certificate store on the local computer or a CRL distribution point (cRLDistributionPoint) object in Active Directory. Each of the objects output by this cmdlet represents a certificate revocation list found in the specified location, and can be passed to other *-QADCertificateRevocationList cmdlets intended to manage certificate revocation lists.

Examples

Example 1

Retrieve the certificate revocation lists from the Intermediate Certification Authorities certificate store in the CurrentUser store location:

```
C:\PS> Get-QADLocalCertificateStore CertificateAuthority |  
Get-QADCertificateRevocationList
```

Example 2

Retrieve the certificate revocation lists from the CRL distribution point (CDP) container in Active Directory:

```
C:\PS> Get-QADPKIObject CDP | Get-QADCertificateRevocationList
```

Add-QADCertificateRevocationList

Add certificate revocation lists to a local certificate store.

Syntax

```
Add-QADCertificateRevocationList [-Store] <X509CertificateStoreUI> [-CRL]
<CertificateRevocationListUI[]> [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

CRL

Use this parameter to specify the CRL objects representing the certificate revocation lists to add. This could be output objects of the Import-QADCertificateRevocationList cmdlet (see examples).

Store

Parameter value is an object that identifies the certificate store to which to add certificate revocation lists. Normally, this is an output object of the Get-QADLocalCertificateStore cmdlet (see examples).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to add the certificate revocation list (CRL) represented by a given CRL object to a local certificate store. The cmdlet takes an output object of the Get-QADCertificateRevocationList or Import-QADCertificateRevocationList cmdlet, and updates the specified certificate store with the certificate revocation list represented by that object.

Examples

Example 1

Import the certificate revocation lists from the files located in a certain folder to a particular local certificate store:

```
C:\PS> $crl = dir c:\crl | Import-QADCertificateRevocationList
```

```
C:\PS> Get-QADLocalCertificateStore MyStore | Add-QADCertificateRevocationList -CRL $crl
```

The first command populates the `$crl` variable with the objects representing the certificate revocation lists found in the files that are located in the specified folder (`c:\crl`). In the second command, `Get-QADLocalCertificateStore MyStore` retrieves the certificate store and passes the output object to `Add-QADCertificateRevocationList` whose CRL parameter takes the `$crl` variable, thereby causing the certificate revocation lists from the `$crl` variable to be added to the certificate store identified by the output object of `Get-QADLocalCertificateStore MyStore`.

Import-QADCertificateRevocationList

Create a certificate revocation list (CRL) object and populate the object with the CRL data from a byte array or a file.

Syntax

```
Import-QADCertificateRevocationList -File <FileInfo>
```

```
Import-QADCertificateRevocationList [-FileName] <String>
```

```
Import-QADCertificateRevocationList -RawData <Object>
```

Parameters

File

This parameter is intended to receive a FileInfo object that identifies the file containing the certificate revocation list (CRL) data to import. If you need to supply the path and name of the file, use the FileName parameter.

With this parameter, the cmdlet takes a CRL file that represents a certificate revocation list, and populates the output object with the CRL the file contains. This could be, for example, a file created by using the Export-QADCertificateRevocationList cmdlet.

FileName

Use this parameter to supply the path and name of the file containing the certificate revocation list (CRL) data to import. The path can be an absolute path, such as c:\export.crl, or a relative path. If the path or file name includes spaces, enclose the parameter value in quotation marks.

With this parameter, the cmdlet takes a CRL file that represents a certificate revocation list, and populates the output object with the CRL the file contains. This could be, for example, a file created by using the Export-QADCertificateRevocationList cmdlet.

RawData

Use this parameter to specify the object that contains the certificate revocation list (CRL) data to import. This could be, for example, an output object of the Get-Content cmdlet.

Detailed Description

Use this cmdlet to create a certificate revocation list (CRL) object populated with the CRL data found in a byte array or a file. The cmdlet can take an output object of the Get-Content cmdlet containing the data found in a CRL file, and return a CRL object that represents the corresponding certificate revocation list. Another option is to have the cmdlet import the certificate revocation list directly from a CRL file specified (see examples).

Examples

Example 1

Create a CRL object that represents a certificate revocation list found in the specified CRL file (c:\export.crl):

```
C:\PS> Import-QADCertificateRevocationList c:\export.crl
```

Example 2

Create a set of CRL objects each of which represents one of the certificate revocation lists found in the CRL files that are located in the specified folder (c:\crl):

```
C:\PS> dir c:\crl | Import-QADCertificateRevocationList
```

Export-QADCertificateRevocationList

Export a certificate revocation list (CRL) to a byte array or a file.

Syntax

```
Export-QADCertificateRevocationList [-CRL] <CertificateRevocationListUI[]> [-Encoding  
<CertificateEncoding>] [-File <String>]
```

Parameters

CRL

Use this parameter to specify the CRL objects representing the certificate revocation lists to export. This could be output objects of the Get-QADCertificateRevocationList cmdlet (see examples).

Encoding

Use this parameter to specify how to encode the export data. The possible parameter values are:

- **Binary** The export data is a pure binary sequence
- **Base64** The export data is base64-encoded (default value)

File

Use this parameter to specify the path and name of a file to which you want to export the certificate revocation lists. The path can be an absolute path, such as c:\export.crl, or a relative path. If the path or file name includes spaces, enclose the parameter value in quotation marks.

Detailed Description

Use this cmdlet to export the certificate revocation list (CRL) represented by a given CRL object, to a byte array or a file. The cmdlet can take an output object of the Get-QADCertificateRevocationList cmdlet, and return a byte array containing the CRL data found in that object. Another option is to have the cmdlet export the certificate revocation list directly to a file specified (see examples).

Examples

Example 1

```
C:\PS> Get-QADLocalCertificateStore CertificateAuthority |  
Get-QADCertificateRevocationList | %{$count++; Export-QADCertificateRevocationList $_  
-File "C:\crl\local-${count}.crl"} | Out-Null
```

In this command, the `Get-QADLocalCertificateStore CertificateAuthority | Get-QADCertificateRevocationList` pipeline retrieves the certificate revocation lists from the Intermediate Certification Authorities certificate store in the CurrentUser store location and passes the CRL objects to a script block where `Export-QADCertificateRevocationList` exports each CRL to a separate file. The export files are base64-encoded since the `Encoding` parameter is omitted.

Example 2

```
C:\PS> Get-QADPKIOBJECT CDP | Get-QADCertificateRevocationList | %{$count++;  
Export-QADCertificateRevocationList $_ -File "C:\crl\ad-${count}.crl" -Encoding Binary}  
| Out-Null
```

In this command, the `Get-QADPKIOBJECT CDP | Get-QADCertificateRevocationList` pipeline retrieves the certificate revocation lists from the CRL distribution point (CDP) container in Active Directory and passes the CRL objects to a script block where `Export-QADCertificateRevocationList` exports each CRL to a separate file. The export files are binary-encoded, which is due to the `Encoding` parameter value of `Binary`.

Remove-QADCertificateRevocationList

Remove certificate revocation lists from a local certificate store.

Syntax

```
Remove-QADCertificateRevocationList [-Store] <X509CertificateStoreUI> [[-CRL]  
<CertificateRevocationListUI[]>] [-WhatIf] [-Confirm]
```

Parameters

Confirm

Prompts you for confirmation before executing the command.

CRL

Use this parameter to specify the CRL objects representing the certificate revocation lists to remove. This could be output objects of the Import-QADCertificateRevocationList cmdlet. If this parameter is omitted, the cmdlet removes all certificate revocation lists from the certificate store specified by the Store parameter (see examples).

Store

Parameter value is an object that identifies the certificate store from which to remove certificate revocation lists. Normally, this is an output object of the Get-QADLocalCertificateStore cmdlet (see examples).

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove the certificate revocation list (CRL) represented by a given CRL object from a local certificate store. The cmdlet takes an output object of the Get-QADCertificateRevocationList or Import-QADCertificateRevocationList cmdlet, and updates the specified certificate store to remove the certificate revocation list represented by that object.

Examples

Example 1

Import the certificate revocation lists from the files located in a certain folder to a particular local certificate store:

```
C:\PS> $crl = dir c:\crl | Import-QADCertificateRevocationList
```

```
C:\PS> Get-QADLocalCertificateStore MyStore | Remove-QADCertificateRevocationList  
-CRL $crl
```

The first command populates the `$crl` variable with the objects representing the certificate revocation lists found in the files that are located in the specified folder (`c:\crl`). In the second command, `Get-QADLocalCertificateStore MyStore` retrieves the certificate store and passes the output object to `Remove-QADCertificateRevocationList` whose `CRL` parameter takes the `$crl` variable, thereby causing the certificate revocation lists found in the `$crl` variable to be removed from the certificate store identified by the output object of `Get-QADLocalCertificateStore MyStore`.

Example 2

Remove all certificate revocation lists held in the specified local certificate store:

```
C:\PS> Get-QADLocalCertificateStore MyStore | Remove-QADCertificateRevocationList
```

Get-QADPKIObject

Retrieve objects from PKI-related containers in Active Directory, such as the Certification Authorities, AIA or CDP container, or NTAuthCertificates object.

Syntax

```
Get-QADPKIObject [-Container] <PKIContainerType[]> [-Forest <String>] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnetion>]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

Container

Use this parameter to specify the PKI-related containers from which you want to retrieve data. The possible parameter values are:

- **RootCA** Identifies the Certification Authorities (trusted root CA) container
- **AIA** Identifies the authority information access container
- **SubCA** Same as AIA
- **CDP** Identifies the CRL distribution point container
- **NTAuthCA** Identifies the NTAuthCertificates object

Forest

Use this parameter to identify the Active Directory forest of the PKI-related containers from which you want to retrieve data. Parameter value is the fully qualified distinguished name of the forest root domain. This parameter only has an effect on the operations being performed through Quest One ActiveRoles (connection established using the Proxy parameter). In case of a proxy connection, the Forest parameter is required to identify the forest of the target PKI-related containers, since Quest One ActiveRoles could be configured to manage domains from more than one forest.

Detailed Description

Use this cmdlet to retrieve objects from PKI-related containers that are used by certification authorities (CAs) to publish certificates, certificate revocation lists (CRLs), and other data to Active Directory. This cmdlet is intended to retrieve objects held in those containers. Output objects of this cmdlet could be passed, for example, to cmdlets for retrieving certificates or certificate revocation lists published in Active Directory.

The cmdlet can retrieve data from the following Active Directory containers:

- **Certification Authorities (RootCA)** This is the publication point for the trusted root certification authorities' (CA) certificates. Publishing a root CA's certificate to the Certification Authorities container causes all domain members to import the root CA's certificate into their own trusted root CA stores.

Objects in this container hold certificates for trusted root CAs in the forest. Root CA certificates are added automatically when an enterprise admin sets up an enterprise root CA or stand-alone root CA that is joined to the domain. Root CA certificates can also be added or removed from this container by using the Publish- or Unpublish-QADCertificate cmdlet.

- **Authority information access (AIA)** This is the publication point for the most currently published CA certificates for root and intermediate certification authorities. Publishing CA certificates to the AIA container helps clients find CA certificates dynamically during certificate chain building. The CA certificates that are available in the AIA container are also deployed with group policies into every client computer's Intermediate Certification Authorities store.

Objects in this container hold CA certificates that can be retrieved by clients using the authority information access (AIA) certificate extension to build a valid certificate chain and to retrieve any cross-certificates issued by the CA. The Publish- or Unpublish-QADCertificate cmdlet can be used to add or remove certificates from this container.

- **NTAuthCertificates (NTAuthCA)** Publishing CA certificates to the NTAuthCertificates object indicates that these CAs are trusted to both (1) issue authentication (logon) certificates for any user in the forest and (2) enable logon for smart cards, IIS mapping, and Extensible Authentication Protocol-Transport Layer Security (EAP-TLS). The CA certificates that are available in the NTAuthCertificates object are also deployed with group policies into every client computer's Intermediate Certification Authorities store.

Normally, this object contains all of the CA certificates in the current forest. Certificates are added automatically when a new CA is installed by an enterprise admin. Certificates can also be added or removed from this object by using the Publish- or Unpublish-QADCertificate cmdlet.

- **CRL distribution point (CDP)** This is the publication point for the certification authorities' (CA) certificate revocation lists (CRL). Publishing a CA's certificate revocation list to the CDP container enables all domain members to verify the revocation status of certificates issued by the CA.

Objects in this container hold all base CRLs and delta CRLs published in the forest. Certificate revocation lists can be added or removed from this container by using the Publish- or Unpublish-QADCertificateRevocationList cmdlet.

For every Active Directory forest, these containers are located in the forest's Configuration naming context under Services/Public Key Services, and are therefore replicated to every domain controller in the forest.

Examples

Example 1

Retrieve all the certificates that are published in the Certification Authorities (RootCA) or AIA container:

```
C:\PS> Get-QADPKIObject RootCA,AIA | Get-QADCertificate
```

Example 2

Retrieve all the certificate revocation lists that are published in the forest:

```
C:\PS> Get-QADPKIObject CDP | Get-QADCertificateRevocationList
```

Publish-QADCertificate

Publish X.509 certificates to PKI-related containers in Active Directory.

Syntax

```
Publish-QADCertificate [-Container] <CAContainerType[]> [-Certificate]
<X509CertificateUI[]> [-Forest <String>] [-CrossCertificate] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to publish. This could be output objects of the *Get-QADCertificate* or *Import-QADCertificate* cmdlet (see examples).

Confirm

Prompts you for confirmation before executing the command.

Container

Use this parameter to specify the PKI containers where you want to publish certificates. The possible parameter values are:

- **RootCA** Identifies the Certification Authorities (trusted root CA) container
- **AIA** Identifies the authority information access container
- **SubCA** Same as AIA
- **NTAuthCA** Identifies the NTAuthCertificates object

CrossCertificate

Supply this parameter when publishing cross-certificates.

A cross-certificate is a certificate issued by one Certification Authority (CA) that signs the public key for the root certificate of another Certification Authority. Cross-certificates provide a means to create a chain of trust from a single, trusted, root CA to multiple other CAs.

Forest

Use this parameter to identify the Active Directory forest where you want to publish certificates. Parameter value is the fully qualified distinguished name of the forest root domain. This parameter only has an effect on the operations being performed through Quest One ActiveRoles (connection established using the Proxy parameter). In case of a proxy connection, the Forest parameter is required to identify the forest of the PKI containers to act upon, since Quest One ActiveRoles could be configured to manage domains from more than one forest.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to publish X.509 certificates to PKI-related containers in the Active Directory configuration naming context. The following containers are supported:

- **Certification Authorities** (RootCA) This is the publication point for the trusted root certification authorities' (CA) certificates. Publishing a root CA's certificate to the Certification Authorities container causes all domain members to import the root CA's certificate into their own trusted root CA stores.
- **Authority information access** (AIA) This is the publication point for the most currently published CA certificates for root and intermediate certification authorities. Publishing CA certificates to the AIA container helps clients find CA certificates dynamically during certificate chain building. The CA certificates that are available in the AIA container are also deployed with group policies into every client computer's Intermediate Certification Authorities store.
- **NTAuthCertificates** (NTAuthCA) Publishing CA certificates to the NTAuthCertificates object indicates that these CAs are trusted to both (1) issue authentication (logon) certificates for any user in the forest and (2) enable logon for smart cards, IIS mapping, and Extensible Authentication Protocol-Transport Layer Security (EAP-TLS). The CA certificates that are available in the NTAuthCertificates object are also deployed with group policies into every client computer's Intermediate Certification Authorities store.

For every Active Directory forest, these containers are located in the forest's Configuration naming context under Services/Public Key Services, and are therefore replicated to every domain controller in the forest.

Examples

Example 1

Publish the certificates from the files located in the c:\cert folder to the authority information access (AIA) container:

```
C:\PS> dir c:\cert | Import-QADCertificate | Publish-QADCertificate -Container AIA
```

Unpublish-QADCertificate

Remove X.509 certificates from PKI-related containers in Active Directory.

Syntax

```
Unpublish-QADCertificate [-Container] <CAContainerType[]> [-Certificate]
<X509CertificateUI[]> [-Forest <String>] [-Force] [-CrossCertificate] [-Proxy]
[-UseGlobalCatalog] [-Service <String>] [-ConnectionAccount <String>]
[-ConnectionPassword <SecureString>] [-Credential <PSCredential>] [-Connection
<ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the “Connect-QADService” section earlier in this document.

Parameters

Certificate

Use this parameter to specify the certificate objects representing the certificates to remove. This could be output objects of the *Get-QADCertificate* or *Import-QADCertificate* cmdlet (see examples).

Confirm

Prompts you for confirmation before executing the command.

Container

Use this parameter to specify the PKI containers from which you want to remove certificates. The possible parameter values are:

- **RootCA** Identifies the Certification Authorities (trusted root CA) container
- **AIA** Identifies the authority information access container
- **SubCA** Same as AIA
- **NTAuthCA** Identifies the NTAuthCertificates object

CrossCertificate

Supply this parameter when removing cross-certificates.

A cross-certificate is a certificate issued by one Certification Authority (CA) that signs the public key for the root certificate of another Certification Authority. Cross-certificates provide a means to create a chain of trust from a single, trusted, root CA to multiple other CAs.

Force

Supply this parameter to delete the Certification Authority object from which all certificates have been removed by the unpublish operation. Without this parameter, the cmdlet does not delete the Certification Authority object, even though all certificates are removed from that object.

Forest

Use this parameter to identify the Active Directory forest where you want to unpublish certificates. Parameter value is the fully qualified distinguished name of the forest root domain. This parameter only has an effect on the operations being performed through Quest One ActiveRoles (connection established using the Proxy parameter). In case of a proxy connection, the Forest parameter is required to identify the forest of the PKI containers to act upon, since Quest One ActiveRoles could be configured to manage domains from more than one forest.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to remove X.509 certificates from Certification Authority objects held in PKI-related containers in the Active Directory configuration naming context. The following containers are supported:

- **Certification Authorities** (RootCA) This is the publication point for the trusted root certification authorities' (CA) certificates. Publishing a root CA's certificate to the Certification Authorities container causes all domain members to import the root CA's certificate into their own trusted root CA stores.
- **Authority information access** (AIA) This is the publication point for the most currently published CA certificates for root and intermediate certification authorities. Publishing CA certificates to the AIA container helps clients find CA certificates dynamically during certificate chain building. The CA certificates that are available in the NTAuthCertificates object are also deployed with group policies into every client computer's Intermediate Certification Authorities store.
- **NTAuthCertificates** (NTAuthCA) Publishing CA certificates to the NTAuthCertificates object indicates that these CAs are trusted to both (1) issue authentication (logon) certificates for any user in the forest and (2) enable logon for smart cards, IIS mapping, and Extensible Authentication Protocol-Transport Layer Security (EAP-TLS). The CA certificates that are available in the NTAuthCertificates object are also deployed with group policies into every client computer's Intermediate Certification Authorities store.

For every Active Directory forest, these containers are located in the forest's Configuration naming context under Services/Public Key Services, and are therefore replicated to every domain controller in the forest.

Examples

Example 1

Remove the certificates found in the certificate files held in the c:\cert folder, from the authority information access (AIA) and trusted root CA (RootCA) containers:

```
C:\PS> dir c:\cert | Import-QADCertificate | Unpublish-QADCertificate AIA,RootCA
```

Publish-QADCertificateRevocationList

Publish certificate revocation lists to the CRL distribution point (CDP) container in Active Directory.

Syntax

```
Publish-QADCertificateRevocationList [-CAName] <String[]> [-CRL]
<CertificateRevocationListUI[]> [-Forest <String>] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

CAName

Use this parameter to specify the Certification Authority name for the publish operation. When publishing a certificate revocation list (CRL), the cmdlet adds the CRL to a certain CRL distribution point (cRLDistributionPoint) object in a sub-container of the CDP container, with the sub-container's name identified by the value of this parameter. A parameter value could be the NetBIOS name of the server running the Certification Authority for which you want to publish a certificate revocation list (see examples).

Confirm

Prompts you for confirmation before executing the command.

CRL

Use this parameter to specify the CRL objects representing the certificate revocation lists to publish. This could be output objects of the Import-QADCertificateRevocationList cmdlet (see examples).

Forest

Use this parameter to identify the Active Directory forest where you want to publish certificate revocation lists. Parameter value is the fully qualified distinguished name of the forest root domain. This parameter only has an effect on the operations being performed through Quest One ActiveRoles (connection established using the Proxy parameter). In case of a proxy connection, the Forest parameter is required to identify the forest of the target CDP container, since Quest One ActiveRoles could be configured to manage domains from more than one forest.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to publish certificate revocation lists to the CRL distribution point (CDP) container in the Active Directory configuration naming context. The CDP container is the publication point for the certification authorities' (CA) certificate revocation lists (CRL). Publishing a CA's certificate revocation list to the CDP container enables all domain members to verify the revocation status of certificates issued by the CA. For every Active Directory forest, the CDP container is located in the forest's Configuration naming context, and is therefore replicated to every domain controller in the forest. Note that the CRLs that are available in the CDP container are not automatically deployed into client computers, so CRLs in this container have an effect only on certificates whose CRL distribution points setting specifies the CDP container as a CRL location.

Publication of a CRL effectively adds the CRL to a certain CRL distribution point (cRLDistributionPoint) object held in a sub-container of the CDP container, with the sub-container normally identified by the NetBIOS name of a particular CA server computer. The cmdlet allows you to specify one or more CA names for a single publish operation.

Examples

Example 1

Publish the certificate revocation lists from the files located in the c:\crl folder to a CRL distribution point object for the CA server named CA3SRV:

```
C:\PS> dir c:\crl | Import-QADCertificateRevocationList |  
Publish-QADCertificateRevocationList -CAName CA3SRV
```

Unpublish-QADCertificateRevocationList

Remove certificate revocation lists from the CRL distribution point (CDP) container in Active Directory.

Syntax

```
Unpublish-QADCertificateRevocationList [-CAName] <String[]> [-CRL]
<CertificateRevocationListUI[]> [-Forest <String>] [-Force] [-Proxy] [-UseGlobalCatalog]
[-Service <String>] [-ConnectionAccount <String>] [-ConnectionPassword <SecureString>]
[-Credential <PSCredential>] [-Connection <ArsConnection>] [-WhatIf] [-Confirm]
```

The cmdlet has optional parameters that determine the server and the security context for the operation. The connection parameters could be omitted since a connection to a server is normally established prior to using this cmdlet. In this case, the server and the security context are determined by the **Connect-QADService** cmdlet. If you do not use **Connect-QADService** and have no connection established prior to using a cmdlet, then the connection settings, including the server and the security context, are determined by the connection parameters of the first cmdlet you use. Subsequent cmdlets will use those settings by default.

The connection parameters include: *Proxy*, *Service*, *ConnectionAccount*, *ConnectionPassword*, *Credential*, *Connection*, and *UseGlobalCatalog*. For parameter descriptions, see the "Connect-QADService" section earlier in this document.

Parameters

CAName

Use this parameter to specify the Certification Authority name for the unpublish operation. When unpublishing a certificate revocation list (CRL), the cmdlet removes the CRL to a certain CRL distribution point (cRLDistributionPoint) object in a sub-container of the CDP container, with the sub-container's name identified by the value of this parameter. A parameter value could be the NetBIOS name of the server running the Certification Authority for which you want to unpublish a certificate revocation list (see examples).

Confirm

Prompts you for confirmation before executing the command.

CRL

Use this parameter to specify the CRL objects representing the certificate revocation lists to remove. This could be output objects of the Import-QADCertificateRevocationList cmdlet (see examples).

Force

Supply this parameter to delete the CRL distribution point (cRLDistributionPoint) object from which all certificate revocation lists have been removed by the unpublish operation. Without this parameter, the cmdlet does not delete the cRLDistributionPoint object, even though all certificate revocation lists are removed from that object. This parameter also causes the cmdlet to delete the container that held the deleted cRLDistributionPoint object, if the container is empty.

Forest

Use this parameter to identify the Active Directory forest where you want to unpublish certificate revocation lists. Parameter value is the fully qualified distinguished name of the forest root domain. This parameter only has an effect on the operations being performed through Quest One ActiveRoles (connection established using the Proxy parameter). In case of a proxy connection, the Forest parameter is required to identify the forest of the target CDP container, since Quest One ActiveRoles could be configured to manage domains from more than one forest.

WhatIf

Describes what would happen if you executed the command, without actually executing the command.

Detailed Description

Use this cmdlet to unpublish certificate revocation lists from the CRL distribution point (CDP) container in the Active Directory configuration naming context. The CDP container is the publication point for the certification authorities' (CA) certificate revocation lists (CRL). Publishing a CA's certificate revocation list to the CDP container enables all domain members to verify the revocation status of certificates issued by the CA. For every Active Directory forest, the CDP container is located in the forest's Configuration naming context, and is therefore replicated to every domain controller in the forest. Note that the CRLs that are available in the CDP container are not automatically deployed into client computers, so CRLs in this container have an effect only on certificates whose CRL distribution points setting specifies the CDP container as a CRL location.

Unpublishing a CRL effectively removes the CRL from a certain CRL distribution point (cRLDistributionPoint) object held in a sub-container of the CDP container, with the sub-container normally identified by the NetBIOS name of a particular CA server computer. The cmdlet allows you to specify one or more CA names for a single unpublish operation.

Examples

Example 1

Remove the certificate revocation lists found in the files held in the c:\crl folder, from a CRL distribution point object for the CA server named CA3SRV:

```
C:\PS> dir c:\crl | Import-QADCertificateRevocationList |  
Unpublish-QADCertificateRevocationList CAName CA3SRV -Force
```

The Force parameter in this command causes the cmdlet to delete the CRL distribution point object from which all certificate revocation lists are removed.

Cmdlet Reference - Utility

Here you can find information about command-line tools (cmdlets) that are provided by ActiveRoles Management Shell.

This section covers the utility cmdlets, such as cmdlets for configuring the shell or converting data from one data type to another.

Set-QADInactiveAccountsPolicy

Set the current user preference on what accounts to consider inactive by default.

Syntax

```
Set-QADInactiveAccountsPolicy [-AccountExpiredPeriod <Int32>]  
[-PasswordNotChangedPeriod <Int32>] [-AccountNotLoggedOnPeriod <Int32>]
```

Parameters

AccountExpiredPeriod

Use this parameter to specify the number of days after which an expired account is considered inactive by default. Thus, an account is considered inactive if the account remains in the expired state for more days than specified by this parameter.

AccountNotLoggedOnPeriod

Use this parameter to specify the period, in days, that an account is not used to log on, after which the account is considered inactive by default. Thus, an account is considered inactive if no successful logons to that account occur for more days than specified by this parameter.

The last time that a given user or computer successfully logged on to the domain is retrieved from the lastLogonTimeStamp attribute of the user or computer object. This requires the domain functional level of Windows 2003 or higher. Also note that Active Directory updates this attribute only periodically, rather than every time that a user or computer logs on. The period of update is configurable, and defaults to 14 days. This means that lastLogonTimeStamp for any given user or computer could be off by as much as 14 days, so the true last logon time is later than lastLogonTimeStamp. Hence, it is advisable to choose AccountNotLoggedOnPeriod of more than 14 days.

PasswordNotChangedPeriod

Use this parameter to specify the password age, in days, after which an account is considered inactive by default. Thus, an account is considered inactive if the password of the account remains unchanged for more days than specified by this parameter.

Detailed Description

Use this cmdlet to specify the default conditions that must be met for a user or computer account to be considered inactive. The inactivity conditions are specific to the current user, and have an effect on the cmdlets that support the Inactive parameter (such as Get-QADUser or Get-QADComputer). If no account-inactivity related parameters other than Inactive are supplied, then the Inactive parameter retrieves the accounts that meet the conditions defined by this cmdlet. To view the inactivity conditions that are currently in effect, use the Get-QADInactiveAccountsPolicy cmdlet.

Examples

Example 1

Set the default inactivity conditions so that an account is considered inactive if any of the following is true:

- The account is expired
- The account was not used to log on for at least 30 days
- The password of the account remains unchanged for at least 120 days

```
C:\PS> Set-QADInactiveAccountsPolicy -AccountExpiredPeriod 0 -AccountNotLoggedInPeriod 30 -PasswordNotChangedPeriod 120
```

Get-QADInactiveAccountsPolicy

View the current user preference on what accounts to consider inactive by default.

Syntax

```
Get-QADInactiveAccountsPolicy
```

Detailed Description

Use this cmdlet to examine the settings that were specified by using Set-QADInactiveAccountsPolicy, and are in effect for the current user session. These settings specify the default conditions that must be met for a user or computer account to be considered inactive. The inactivity conditions are specific to the current user, and have an effect on the cmdlets that support the Inactive parameter (such as Get-QADUser or Get-QADComputer). If no account-inactivity related parameters other than Inactive are supplied, then the Inactive parameter retrieves the accounts that meet the conditions defined by the AccountExpiredPeriod, AccountNotLoggedInPeriod, and PasswordNotChangedPeriod settings that you can examine using this cmdlet. For details regarding each of these settings, see the corresponding parameter description for the Set-QADInactiveAccountsPolicy cmdlet.

Examples

Example 1

View the default account inactivity conditions that are in effect for the current user session:

```
C:\PS> Get-QADInactiveAccountsPolicy
```

Set-QADProgressPolicy

Set the user preference on whether to display a progress bar for long-running commands.

Syntax

```
Set-QADProgressPolicy [-ShowProgress] [-Threshold <Int32>]
```

Parameters

ShowProgress

Use this parameter to specify whether you want QAD and QARS cmdlets to display a progress bar that depicts the status of the running command in case of a lengthy operation. This setting has an effect only on the cmdlets that support the ShowProgress and ProgressThreshold parameters to control the appearance of a progress bar.

The ShowProgress setting of TRUE causes a cmdlet that supports the ShowProgress parameter to display a progress bar, even if the ShowProgress parameter is omitted. The ShowProgress setting of FALSE suppresses a progress bar unless the ShowProgress parameter is supplied.

Threshold

Use this parameter to set the default delay, in seconds, before a cmdlet that performs a lengthy operation displays a progress bar to depict the status of the running command. If the running command finishes before the threshold time has elapsed, a progress bar does not appear. This threshold time setting is used as the default threshold time setting by the QAD and QARS cmdlets that support the ShowProgress and ProgressThreshold parameters to control the appearance of a progress bar.

Detailed Description

Use this cmdlet to specify whether you want the QAD and QARS cmdlets that support the ShowProgress parameter, to display a progress bar by default. When performing a lengthy operation (that is, an operation that lasts longer than a certain threshold time), such a cmdlet may display a progress bar to depict the status of the operation. Whether a progress bar is displayed, depends upon the ShowProgress setting. A threshold time can be set by using the Threshold parameter. These settings only affect the current user, and can be overridden on a per-cmdlet basis.

Note that the progress bar feature is based on the Write-Progress function that was first introduced in version 2.0 of Windows PowerShell. With Windows PowerShell 1.0 this feature is not available.

Examples

Example 1

Set the user preference for the progress bar appearance policy, to cause a progress bar to appear by default when a command takes longer than 2 seconds to finish:

```
C:\PS> Set-QADProgressPolicy -ShowProgress $true -ProgressThreshold 2 | Out-Null
```

Get-QADProgressPolicy

View the user preference on whether to display a progress bar for long-running commands.

Syntax

```
Get-QADProgressPolicy
```

Detailed Description

Use this cmdlet to examine the settings that were specified by using Set-QADProgressPolicy, and are in effect for the current user session. These settings control whether the QAD and QARS cmdlets that support the ShowProgress parameter display a progress bar by default. When performing a lengthy operation (that is, an operation that lasts longer than a certain threshold time), such a cmdlet may display a progress bar to depict the status of the operation. Whether a progress bar is displayed, depends upon the ShowProgress setting. The threshold time is controlled by the Threshold setting. These settings only affect the current user, and can be overridden on a per-cmdlet basis. For details regarding the ShowProgress or Threshold setting, see the corresponding parameter description for the Set-QADProgressPolicy cmdlet.

Note that the progress bar feature is based on the Write-Progress function that was first introduced in version 2.0 of Windows PowerShell. With Windows PowerShell 1.0 this feature is not available.

Examples

Example 1

View the ShowProgress and Threshold settings that are in effect for the current user session:

```
C:\PS> Get-QADProgressPolicy
```

Convert-QADAttributeValue

Convert attribute values of a directory object to the specified .NET type.

Syntax

```
Convert-QADAttributeValue -Input <Object> -OutputTypeName <String>
```

Parameters

Input

Specify the object representing the attribute value to convert. This parameter accepts pipeline input, and can be omitted on the command line if you pipe into this cmdlet an object returned by a Get- cmdlet (see examples).

OutputTypeName

Specify the fully qualified name of the .NET type to convert the attribute value to. The assembly name and namespace indication can be omitted if the type is from the System namespace (see examples).

Detailed Description

Use this cmdlet to convert attribute values of directory objects returned by a cmdlet (for example, by a **Get-QADUser** cmdlet). You can convert:

- Values of the byte[] type to the SecurityIdentifier or Guid type
- Values of the IADsLargeInteger type to the Int64, DateTime, or TimeSpan type

Examples

Example 1

Convert the value of the objectGuid attribute to the Guid type, and display the value in the console window:

```
C:\PS> get-QADuser 'MyDomain\JSmith' | %{$_.DirectoryEntry.objectGuid} |  
convert-QADAttributeValue -outputTypeName 'Guid' | Write-Host
```

Example 2

Convert the value of the objectSid attribute to the SecurityIdentifier type, and display the value in the console window:

```
C:\PS> get-QADuser 'MyDomain\JSmith' | %{$_.DirectoryEntry.objectSid} |  
convert-QADAttributeValue -outputTypeName 'Security.Principal.SecurityIdentifier' |  
Write-Host
```

Example 3

Convert the value of the lastLogon attribute to the DateTime type, and display the value in the console window:

```
C:\PS> get-QADuser 'MyDomain\JSmith' | %{$_.DirectoryEntry.lastLogon} |  
convert-QADAttributeValue -outputTypeName 'DateTime' | Write-Host
```

Example 4

For each domain controller, retrieve the time that the user JSmith last logged on by using a particular domain controller, and display the results in the console window:

```
C:\PS> get-QADComputer -searchRoot 'mydomain.company.com/domain controllers' |  
Select-Object Name,@{Name="Last Logon"; Expression={%{ get-QADUser 'MyDomain\JSmith'  
-Service $_.Name} | %{$_.DirectoryEntry.lastLogon} | convert-QADAttributeValue  
-outputTypeName 'DateTime'}}
```

Get-QADPSSnapinSettings

View default settings that apply to all cmdlets of this PowerShell snap-in.

Syntax

```
Get-QADPSSnapinSettings [-DefaultExcludedProperties]
[-DefaultPropertiesExcludedFromNonBaseSearch] [-Integer8AttributesThatContainDateTimes]
[-Integer8AttributesThatContainNegativeTimeSpans] [-DefaultPageSize]
[-DefaultSizeLimit] [-DefaultSearchScope] [-DefaultWildcardMode]
[-DefaultOutputPropertiesForUserObject] [-DefaultOutputPropertiesForGroupObject]
[-DefaultOutputPropertiesForComputerObject] [-DefaultOutputPropertiesForAdObject]
[-DefaultOutputPropertiesForPasswordSettingsObject]
```

Parameters

DefaultExcludedProperties

This parameter causes the cmdlet to return a list of the attributes that are excluded from processing by the UseDefaultExcludedProperties parameter on any particular cmdlet.

DefaultOutputPropertiesForAdObject

This parameter causes the cmdlet to return the default list of the object attributes that are retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for objects other than a User, Group, Computer, or Password Settings object.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForComputerObject

This parameter causes the cmdlet to return the default list of the Computer object attributes that are retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Computer objects.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForGroupObject

This parameter causes the cmdlet to return the default list of the Group object attributes that are retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Group objects.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForPasswordSettingsObject

This parameter causes the cmdlet to return the default list of the Password Settings object attributes that are retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Password Settings objects.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForUserObject

This parameter causes the cmdlet to return the default list of the User object attributes that are retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for User objects.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultPageSize

Supply this parameter to view the default value of the PageSize parameter that is used by the Get- cmdlets. This page size value is used if the PageSize parameter is omitted.

DefaultPropertiesExcludedFromNonBaseSearch

This parameter causes the cmdlet to return a list of the attributes that are not retrieved from the directory and stored in the local memory cache by any particular Get- cmdlet during a search with the search scope other than 'Base'.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

DefaultSearchScope

Supply this parameter to view the default value of the SearchScope parameter that is used by the Get- cmdlets. This search scope setting is used if the SearchScope parameter is omitted.

DefaultSizeLimit

Supply this parameter to view the default value of the SizeLimit parameter that is used by the Get- cmdlets. This limitation on the size of the search result set is used if the SizeLimit parameter is omitted.

DefaultWildcardMode

Supply this parameter to view the default value of the WildcardMode parameter that is used by the Get- cmdlets. This wildcard mode setting is used if the WildcardMode parameter is omitted.

Integer8AttributesThatContainDateTimes

This parameter causes the cmdlet to return a list of the Integer8 attributes whose values are represented as DateTime objects in the output of the Get- cmdlets by default. Each attribute is identified by its LDAP display name.

Note: This setting applies only to the properties of a cmdlet's output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

Integer8AttributesThatContainNegativeTimeSpans

This parameter causes the cmdlet to return a list of the Integer8 attributes whose values are represented as TimeSpan objects in the output of the Get- cmdlets by default. Each attribute is identified by its LDAP display name.

The output TimeSpan objects represent the absolute values of the attributes, and thus indicate positive time intervals regardless of whether an actual attribute value is a negative or positive time interval.

Note: This setting applies only to the properties of a cmdlet's output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

Detailed Description

You can use this cmdlet to view some default settings that have effect within this PowerShell snap-in on any cmdlet where those settings are applicable. To change default settings, use the **Set-QADPSSnapinSettings** cmdlet.

Set-QADPSSnapinSettings

Modify default settings that apply to all cmdlets of this PowerShell snap-in.

Syntax

```
Set-QADPSSnapinSettings [-DefaultExcludedProperties <String[]>]
[-DefaultPropertiesExcludedFromNonBaseSearch <String[]>]
[-Integer8AttributesThatContainDateTimes <String[]>]
[-Integer8AttributesThatContainNegativeTimeSpans <String[]>] [-DefaultPageSize <Int32>]
[-DefaultSizeLimit <Int32>] [-DefaultSearchScope <SearchScope>] [-DefaultWildcardMode
<WildcardMode>] [-DefaultOutputPropertiesForUserObject <String[]>]
[-DefaultOutputPropertiesForGroupObject <String[]>]
[-DefaultOutputPropertiesForComputerObject <String[]>]
[-DefaultOutputPropertiesForAdObject <String[]>]
[-DefaultOutputPropertiesForPasswordSettingsObject <String[]>]
```

Parameters

DefaultExcludedProperties

Use this parameter to specify the attributes that are excluded from processing by the UseDefaultExcludedProperties parameter on any particular cmdlet. Supply a list of the attribute LDAP display names as the parameter value.

DefaultOutputPropertiesForAdObject

Use this parameter to specify the default list of the object attributes that are to be retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for objects other than a User, Group, Computer, or Password Settings object. Supply a list of the attribute LDAP display names as the parameter value.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForComputerObject

Use this parameter to specify the default list of the Computer object attributes that are to be retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Computer objects. Supply a list of the attribute LDAP display names as the parameter value.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForGroupObject

Use this parameter to specify the default list of the Group object attributes that are to be retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Group objects. Supply a list of the attribute LDAP display names as the parameter value.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForPasswordSettingsObject

Use this parameter to specify the default list of the Password Settings object attributes that are to be retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for Password Settings objects. Supply a list of the attribute LDAP display names as the parameter value.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultOutputPropertiesForUserObject

Use this parameter to specify the default list of the User object attributes that are to be retrieved from the directory and stored in the local memory cache by a Get- cmdlet during a search for User objects. Supply a list of the attribute LDAP display names as the parameter value.

Note: Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by a Get- cmdlet. If a particular attribute is not in the cache, the output object may not have a property that would provide access to the value of the attribute.

DefaultPageSize

Specify a new default value of the PageSize parameter for the Get- cmdlets. This page size value is used if the PageSize parameter is omitted. Initially, the default value of the PageSize parameter is set to 50.

DefaultPropertiesExcludedFromNonBaseSearch

Use this parameter to specify the attributes that are not to be retrieved from the directory and stored in the local memory cache by any particular Get- cmdlet during a search with the search scope other than 'Base'. Supply a list of the attribute LDAP display names as the parameter value.

Note: If a cmdlet does not cache a particular attribute, then the output object returned by the cmdlet may not have a property that would provide access to the value of the attribute.

DefaultSearchScope

Specify a new default value of the SearchScope parameter for the Get- cmdlets. Acceptable vales are:

- 'Base'
- 'OneLevel'
- 'Subtree'

This search scope setting is used if the SearchScope parameter is omitted. Initially, the default value of the SearchScope parameter is set to 'Subtree'.

DefaultSizeLimit

Specify a new default value of the SizeLimit parameter for the Get- cmdlets. This limitation on the size of the search result set is used if the SizeLimit parameter is omitted. Initially, the default value of the SizeLimit parameter is set to 1000.

DefaultWildcardMode

Specify a new default value of the WildcardMode parameter for the Get- cmdlets. Acceptable vales are:

- 'Ldap'
- 'PowerShell'

This wildcard mode setting is used if the WildcardMode parameter is omitted. Initially, the default value of the WildcardMode parameter is set to 'Ldap'.

Integer8AttributesThatContainDateTimes

Use this parameter to specify the Integer8 attributes whose values you want to be represented as DateTime objects in the output of the Get- cmdlets by default. Supply a list of the attribute LDAP display names as the parameter value.

Note: This setting applies only to the properties of a cmdlet's output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

Integer8AttributesThatContainNegativeTimeSpans

Use this parameter to specify the Integer8 attributes whose values you want to be represented as TimeSpan objects in the output of the Get- cmdlets by default. Supply a list of the attribute LDAP display names as the parameter value.

The output TimeSpan objects represent the absolute values of the attributes, and thus indicate positive time intervals regardless of whether an actual attribute value is a negative or positive time interval.

Note: This setting applies only to the properties of a cmdlet's output object that have the member type of NoteProperty. Such properties are normally added to the output object in order to provide access to the attribute values of the respective directory object that are loaded to the local memory cache but cannot be accessed by using properties of the base object (the object for which the output object serves as a wrapper).

Detailed Description

You can use this cmdlet to modify some default settings that have effect within this PowerShell snap-in on any cmdlet where those settings are applicable. The changes you make to default settings are in effect during the current PowerShell session, and are discarded once you close the PowerShell console window. To view the default settings that are currently in effect, use the **Get-QADPSSnapinSettings** cmdlet.

Examples

Example 1

Configure the **Get-** cmdlets to return all search results by default (rather than limit the search result set to a maximum of 1000 items):

```
C:\PS> Set-QADPSSnapinSettings -DefaultSizeLimit 0
```

Example 2

Configure the Get- cmdlets to cache the 'msDS-ReplAttributeMetaData' attribute when retrieving User objects from the directory, in addition to the other attributes that are cached by default:

```
C:\PS> $list = Get-QADPSSnapinSettings -DefaultOutputPropertiesForUserObject
C:\PS> $list += 'msDS-ReplAttributeMetaData'
C:\PS> Set-QADPSSnapinSettings -DefaultOutputPropertiesForUserObject $list
```

Caching an attribute guarantees that the value of the attribute can be read by using properties of the output object returned by the Get- cmdlet. Thus, after you have changed configuration so as to cache the 'msDS-ReplAttributeMetaData' attribute, you can view the value of that attribute on a user account by using the following command:

```
C:\PS> Get-QADUser 'MyDomain\JSmith' | Format-Table name, 'msDS-ReplAttributeMetaData'
```

Enable-QADDiagnosticLog

Turn on diagnostic logging in ActiveRoles Management Shell. Diagnostic logging is mainly intended to be used by support personnel for troubleshooting purposes.

Syntax

```
Enable-QADDiagnosticLog [-Path] <String> [-Global]
```

Parameters

Path

Specify the path and name of the log file to hold the diagnostic information.

Global

Use this parameter to enable diagnostic logging for the current and future sessions. Without this parameter, diagnostic logging is enabled for the current session only.

Note: The Global settings are stored separately for the 32-bit and 64-bit versions of Management Shell. Enabling global logging for the 64-bit version does not enable global logging for the 32-bit version, and vice versa.

Detailed Description

Use this cmdlet to turn on diagnostic logging so as to have the ActiveRoles Management Shell cmdlets record diagnostic information to the file specified. When you use this cmdlet without the Global parameter, diagnostic logging is enabled only during the current session. If you want to keep diagnostic logging after you close and reopen Management Shell, use the Global parameter.

Note that the Global parameter does not have an effect on the current session if you have already enabled logging without the Global parameter. Similarly, when global logging is enabled, specifying a new log file by using this cmdlet without the Global parameter causes the current session diagnostic information to be redirected to the file specified.

Examples

Example 1

Turn on diagnostic logging for the current Management Shell session, with the diagnostic information being recorded to the file c:\temp\arsps.log:

```
C:\PS> Enable-QADDiagnosticLog 'c:\temp\arsps.log'
```

Disable-QADDiagnosticLog

Turn off diagnostic logging in ActiveRoles Management Shell. Diagnostic logging is mainly intended to be used by support personnel for troubleshooting purposes.

Syntax

```
Disable-QADDiagnosticLog [-Global]
```

Parameters

Global

Use this parameter to stop diagnostic logging that was earlier enabled with the use of the Global parameter.

Note: The Global settings are stored separately for the 32-bit and 64-bit versions of Management Shell. Disabling global logging for the 64-bit version does not disable global logging for the 32-bit version, and vice versa.

Detailed Description

Use this cmdlet to turn off diagnostic logging that was turned on by using Enable-QADDiagnosticLog. Without the Global parameter, this cmdlet turns off diagnostic logging that was enabled only for the current session. Supply the Global parameter if you want to turn off diagnostic logging that was enabled for all sessions.

Examples

Example 1

Turn off diagnostic logging that was earlier enabled for the current Management Shell session:

```
C:\PS> Disable-QADDiagnosticLog
```

Example 2

Turn off diagnostic logging that was earlier enabled for all Management Shell sessions:

```
C:\PS> Disable-QADDiagnosticLog -Global
```

Get-QADDiagnosticLogStatus

Check to see if diagnostic logging is enabled or disabled in ActiveRoles Management Shell.

Syntax

```
Get-QADDiagnosticLogStatus
```

Detailed Description

Use this cmdlet to view status information regarding diagnostic logging. The cmdlet output includes the following items:

- **Local log: Enabled** Indicates that diagnostic logging for the current session was turned on by using Enable-QADDiagnosticLog without the Global parameter. The cmdlet also displays the path to the log file for the current session (local log file).
- **Local log: Disabled** Indicates that diagnostic logging for the current session was not turned on by using Enable-QADDiagnosticLog without the Global parameter.
- **Global log: Enabled** Indicates that diagnostic logging for the current and future sessions was turned on by using Enable-QADDiagnosticLog with the Global parameter. The cmdlet also displays the path to the global log file.
- **Global log: Disabled** Indicates that diagnostic logging for the future sessions was not turned on by using Enable-QADDiagnosticLog with the Global parameter.
- **Active log: Path** Displays the path and name of the file that is currently used to store diagnostic information.

If both local log and global log are enabled, the local log option takes precedence and diagnostic information is recorded to the local rather than global log file.

Examples

Example 1

Check whether diagnostic logging is turned on (enabled) or turned off (disabled):

```
C:\PS> Get-QADDiagnosticLogStatus
```

Using Certificate Management Cmdlets

With digital certificates security administrators can uniquely identify and trust hardware devices, servers and other entities. A certificate based infrastructure (public key infrastructure, PKI) is one of the most secure access solutions; however, it can be complicated to set up and manage. ActiveRoles Management Shell addresses the management challenge for PKI administrators by providing a broad range of certificate management cmdlets. This section reviews the security concepts surrounding digital certificate management and details how certificate management cmdlets can be used to simplify PKI management.

Understanding Digital Certificates

A *digital certificate* is an electronic representation of an entity, such as a user, device (usually a computer), or service and is associated with a certificate holder's public and private key pair. A digital certificate is usually issued by a certificate issuer called a *Certification Authority (CA)* and is digitally signed using the Certification Authority's private key. This digital signature prevents certificate content modification and any other method of certificate tampering. When a Certification Authority signs a certificate, it guarantees that all information in the certificate and the certificate holder's entity is correct.

The process of verifying the authenticity and validity of a certificate normally involves checking not only that certificate, but all of the certificates in the certificate chain. A *certificate chain* (also known as a *certification path*) is a sequence of certificates that establishes a chain of trust from a peer certificate to a trusted CA certificate. Each certificate in the chain is signed by the subsequent certificate, except for the last certificate in the chain, the *root CA* certificate, which is signed by the CA itself. A *certificate chaining engine (CCE)* is used to build and verify certificate chains.

Cryptography Fundamentals

Cryptography in general supports two types of encryption: symmetric encryption and asymmetric encryption.

Symmetric Encryption

Symmetric encryption is the simpler of the two types of encryption because only one key is used for both the encryption and decryption processes. When a user encrypts some data with a symmetric key, the encryption key is required to be delivered to another user who needs access to the encrypted data. Since data is encrypted (secured), it can be transmitted to the recipient over an unsecured or un-trusted network. However, the symmetric encryption key is not secured in any way, so this key should not be transmitted over an unsecured or un-trusted network.

Asymmetric Encryption

Asymmetric encryption uses two keys (a key pair): a private key and a public key. The private key is always held by the user and never exposed. The public key can be transmitted in plain text over an unsecured or un-trusted network. These keys are mathematically matched such that data encrypted with one key can only be decrypted using the other matching key. That is, data encrypted with a user's public key can only be decrypted with that user's private key, and vice-versa.

Best Practices for Symmetric and Asymmetric Encryption

Symmetric encryption is quite simple, but sharing the same key with many recipients is often impractical. Asymmetric encryption, on the other hand, consumes a lot of time and system resources. Therefore, many organizations use both types of encryption together: symmetric encryption is used for large data encryption, and asymmetric encryption is used to encrypt symmetric keys and transfer them to recipients across unsecure or un-trusted networks.

Types of Certificates

Three X.509 certificate versions are supported today, although only version 1 and version 3 are used in practice in Internet PKI. Each of these versions consists of several mandatory fields and optional extensions.

X.509 Certificate Version 1

The X.509 version 1 certificate format consists of the following mandatory fields:

- **Version** The X.509 certificate version.
- **Serial number** The unique serial number of the certificate. A given Certification Authority cannot issue two or more certificates with the same serial number.
- **Signature algorithm** The algorithm that was used by the Certification Authority in the signing process.
- **Signature hash algorithm** The algorithm used by the Certification Authority in computing the certificate signature hash.
- **Issuer** The X.500 distinguished name (DN) of the certificate issuer. This field is used by the certificate chaining engine.
- **Valid from** The start time when the public key can be used and the certificate is considered as valid. Internally, this field is also known as NotBefore.
- **Valid to** The time after which the public key cannot be used. Internally, this field is known as NotAfter. After the time specified in this field, the certificate is considered invalid (although there are some exceptions when the certificate can be considered valid after the **Valid to** time).
- **Subject** The X.500 distinguished name (DN) of the certificate's public and private key holder (user, computer, service, etc.). As specified in RFC5280, this field may contain RFC822 name formats (such as e-mail addresses).
- **Public key** The certificate holder's public key. This field is used for signature verification and/or encryption purposes.

Optionally X.509 version 1 certificates may contain the following fields:

- **Thumbprint** The value of the certificate content hash.
- **Thumbprint algorithm** The algorithm used by the issuing Certification Authority to calculate the certificate content hash.

X.509 Certificate Version 2

X.509 version 1 certificates do not provide a way for a Certification Authority to renew their own certificate. The problem is that when a CA's certificate was renewed, that CA would have two or more signing certificates, each containing the same value in the **Subject** field. Since that value appears in all certificates issued by the CA, there is no way to tell whether those certificates were issued before or after the CA's own certificate was renewed.

To work around this issue, the X.509 version 2 certificates provide two additional fields, both of which are optional:

- **Issuer unique ID** The issuer's unique ID, usually in hexadecimal format.
- **Subject unique ID** The subject unique ID, usually in hexadecimal format.

These two fields are related. When a Certification Authority renews its own certificate, a new ID is generated for that CA. That ID is included in the new CA certificate in the **Subject unique ID** field and in all certificates issued by the CA in the **Issuer unique ID** field. When either of these fields is present in a certificate, the **Version** field in the certificate must be 2 or 3.

For example, suppose that a Certification Authority was assigned the ID **0x1**. This value would appear in the **Subject unique ID** field in its own certificate, and the same value, **0x1**, would appear in the **Issuer unique ID** field in all certificates issued by that CA. When the CA's certificate is renewed, a new ID (for example **0x2**) is generated and included in the **Subject unique ID** field. All certificates that are signed using the renewed CA certificate will have **0x2** in the **Issuer unique ID** field. Therefore, certificate holders can select a proper CA certificate to build a correct chain.

The **Subject unique ID** and **Issuer unique ID** fields also enable the reuse of subject and/or issuer names over time. Since the IETF recommends against the reuse of subject and/or issuer names, X.509 version 2 certificates are not very popular and are not widely used in Internet PKI.

X.509 Certificate Version 3

Neither version 1 nor version 2 of the X.509 certificate format makes it possible to determine whether a certificate was issued to a Certification Authority or to an end entity (a user, computer, or service). Also, since the Certification Authority and its certificate holders can be in different locations, it is necessary to distribute all certificate chains with each issued certificate. Finally, a certificate may need to be decommissioned and considered as invalid prior to the **Valid to** date, such as when an employee leaves a company or the private key corresponding to the certificate is lost or stolen. With version 1 and version 2 certificates, it is hard to maintain certificate revocation information on the clients.

To address these and other issues, version 3 of X.509 certificate format introduces *certificate extensions*, which are optional fields that provide additional information about the certificate or its cryptographic capabilities.

Each extension is composed of three parts:

- **Extension Identifier** An Object Identifier (OID) that indicates the format and definition of the extension.
- **Criticality flag** A flag that indicates the importance of the extension. If an extension is flagged as critical, the extension value cannot be empty and it must be recognized by the application; otherwise (that is, if the extension is marked as critical and either the extension's value is empty or the application cannot recognize the extension's value), the application must

reject the certificate. Some extensions (such as Basic Constraints) must be marked as critical in all cases; some extensions (such as Certificate Policies) must never be marked as critical; and some extensions (such as Subject Alternative Name) may be marked as critical in some cases but not in others.

- **Extension value** The extension's value.

There are many extensions used in the Internet. Vendors can implement their own vendor-specific extensions; however, the vendor is responsible for supporting those extensions in the applications.

Common Certificate Extensions

The following are the most common certificate extensions:

- **Authority Key Identifier (AKI)** This extension provides a means of identifying the public key corresponding to the private key used to sign the certificate. It can contain either the key identifier (public key hash) or the issuer name and issuer certificate serial number. This extension must be the same as the **Subject Key Identifier** in the issuer's certificate. Clients use this extension to build a correct chain (certification path) to select a proper issuer certificate if the issuer has multiple signing certificates.
- **Subject Key Identifier (SKI)** This extension provides a means of identifying certificates that contain a particular public key. Like the **Authority Key Identifier** the **SKI** may contain either the key identifier (public key hash) or the current certificate subject name and subject certificate serial number. The **Subject Key Identifier** extension must appear in certificates issued to Certification Authorities. If the certificate is a CA certificate, the value of this extension must appear in all issued certificates in the **AKI** extension.
- **Key Usage** This extension defines the purpose of the key contained in the certificate. It can be marked as critical or not. RFC5280 defines the following nine key usages (multiple key usages can be asserted in one certificate):
 - **Digital Signature** This flag indicates that the certificate public key is used for validating digital signatures (except certificate and Certificate Revocation List signatures). It is also used for client authentication.
 - **Content Commitment** Formerly known as **Non-Repudiation**. This flag indicates that the certificate public key can be used to validate the signer's identity, preventing a signer from denying that he or she signed an object.
 - **Key Encipherment** This flag indicates that the certificate public key is used for key transport. For example, when two hosts negotiate a secure channel across an unsecured or un-trusted network, they usually generate symmetric encryption keys for data encryption. To transfer the symmetric key, one host encrypts its symmetric key material using a peer public key. This flag is used when an RSA key is used for key management.
 - **Data Encipherment** This flag indicates that the certificate public key is used for data encryption only and cannot be used for encrypting other cryptographic keys. Usually this flag is not used, because many applications use symmetric keys for data encryption for performance reasons. After completing the encryption process with symmetric keys, the application encrypts symmetric key material using the certificate public key.
 - **Key Agreement** This flag means the same as **Key Encipherment** flag with only one exception: this flag is used when the Diffie-Hellman key is used for key management.
 - **Key Cert Sign** This flag indicates that the certificate public key is used to validate the certificate signature. It is usually used for CA certificates only.

- **CRL Sign** This flag indicates that the certificate public key is used to validate the Certificate Revocation List (CRL) signature.
- **Encipher Only** This flag is used in conjunction with the **Key Agreement** extension. The resulting symmetric key can only be used for data encryption.
- **Decipher Only** This flag is used in conjunction with the **Key Agreement** extension. The resulting symmetric key can only be used for data decryption.
- **Private Key Usage Period** This extension limits the private key usage period. For example, suppose a document signing certificate is valid for five years and **Private Key Usage Period** is set to one year. In that case, the certificate holder can sign documents for one year only. For the other four years, the certificate can be used for signature validation only. Windows PKI does not use this extension.
- **Certificate Policies** This extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifiers. Optional qualifiers usually consist of **User Notice**, **Notice URL**, or both. **User Notice** contains a short description of the particular policy and **Notice URL** contains an URL to the certificate policy repository.
- **Policy Mappings** This extension allows certificate policy mapping translation between organizations. It is used for cross-certification between two or more organizations. For example, suppose one organization defines a custom digital signature policy called **Org1 Signing**, and another organization defines its own custom digital signature policy called **Org2 Signing**. Organizationally they are identical, but defined with different OIDs. To make these policies technically identical, policy mapping is used. In that case, the first organization trusts the **Org2 Signing** policy as well as its own **Org1 Signing** policy.
- **Subject Alternative Name (SAN)** This extension allows additional identities to be bound to the certificate subject. Defined options include e-mail address, User Principal Name (UPN), IP address, DNS name, and URI (Uniform Resource Identifier). In certain cases, this extension is the only one that identifies the certificate subject. In these cases, the **Subject** field must be empty, and **Subject Alternative Name** must not be empty and must be marked as critical. When the **Subject** field is not empty, **Subject Alternative Name** must not be marked as critical.
- **Issuer Alternative Name (IAN)** As with **Subject Alternative Name**, this extension is used to associate Internet style identities with the certificate issuer.
- **Subject Directory Attributes** This extension can include any attributes from an organization's X.500 or Lightweight Directory Access Protocol (LDAP) directory. The extension is defined as a sequence of one or more attributes. For each attribute, the OID and its corresponding value must be included.
- **Basic Constraints** This extension identifies whether the subject of the certificate is a Certification Authority or an end entity (such as a user, computer or service), and the maximum depth of valid certification paths that include this certificate. If this extension is omitted and the certificate version is 3, the subject type is an end entity. This extension must appear in CA certificates. The **PathLength** property is used to reduce deep certification paths and also can be used for qualified subordination. For example, suppose an organization doesn't want to make any additional Certification Authorities under a particular CA. Setting **PathLength=0** will prevent path validation for all CA certificates issued by a particular CA where this property is defined. If **PathLength=1**, only one subordinate CA is allowed under a particular CA where this property is defined. This property is included in CA certificates only. If no restrictions are set on path length, the value "None" is used.

- **Name Constraints** This extension is used for qualified subordination and must not appear in end entity certificates. For example, suppose an organization consists of a forest with two trees: Contoso.com and Adatum.com. The organization can restrict one CA to issue certificates for the Contoso.com tree only and another CA to issue certificates for Adatum.com only. Name constraints may consist of two properties: **PermittedSubtrees** and **ExcludedSubtrees**. In our example, the organization will configure Contoso.com in **PermittedSubtrees** and Adatum.com in **ExcludedSubtrees** for one CA. For another CA, Contoso.com will be added to **ExcludedSubtrees** and Adatum.com will be added to **PermittedSubtrees**.

This extension is also used for qualified subordination between two or more organizations. For example, if Contoso.com will subordinate CAs in Adatum, then Contoso will add Contoso.com to **ExcludedSubtrees** and Adatum.com to **PermittedSubtrees**. This will prevent Adatum's CAs from issuing rogue certificates for the Contoso namespace. Name constraints can be defined for X.500, RFC822, IP address, and DNS name forms.

- **Policy Constraints** The extension can prohibit policy mapping between CAs or require that each certificate in a certificate chain include an explicit certificate policy OID. It is usually is used for qualified subordination.
- **Enhanced Key Usages (EKU)** Also known as **Extended Key Usages** or **Application Policies**. While **Key Usage** extensions define basic public key usages, **EKU** allows a more granular determination of certificate public key usage purposes. There are several predefined application policies that can be used in the Internet, such as **Server Authentication** (1.3.6.1.5.5.7.3.1), **Client Authentication** (1.3.6.1.5.5.7.3.2), **Secure e-mail** (1.3.6.1.5.5.7.3.4), and **Code Signing** (1.3.6.1.5.5.7.3.3). Each policy must have its own object identifier (OID). Organizations can create their own custom application policies and are responsible for supporting the custom policies in applications. For example, Microsoft has defined a **Smart card logon** policy with OID of 1.3.6.1.4.1.311.20.2.2. For additional OIDs in the Microsoft namespace, see Microsoft's article at <http://support.microsoft.com/kb/287547>.
- **CRL Distribution Points (CDP)** Since certificates are used widely, it is impractical to distribute Certificate Revocation Lists (CRLs) with each certificate. Instead, a Certification Authority can define points where clients can obtain an actual CRL for a particular CA across an unsecured or un-trusted network. Each CRL is digitally signed and is protected from tampering and unauthorized content change. The **CDP** extension contains a sequence of **DistributionPoints**. Windows CAs may include one or more LDAP, HTTP, or FILE URLs (SSL-secured protocols, such as HTTPS, are not supported in any way). Other protocol types are not supported by Windows PKI. However, RFC5280 defines more extensible options for **DistributionPoints**. Before certificate usage, the client extracts the URL from the **CDP** extension and downloads the actual CRL and verifies that specified certificate is not revoked. If a client is unable to download a CRL using the first URL in the **CDP** extension due to network connectivity issues, name resolution issues, firewall restrictions, and so forth, then the client will try to use the other URLs, if any, in the **CDP** extension in the same order as they are published in the extension. If the client software is unable to retrieve the CRL using any URL, certificate revocation validation may fail, depending on software configuration. This extension must be populated in all certificates that are not presented in a self-signed form (usually these are certification path end points).
- **Inhibit Any-Policy** This extension indicates that **All issuance policies** (OID=2.5.29.32.0) in subordinate Certification Authorities certificates is not considered an explicit match for other certificate policies except when it appears in an intermediate, self-signed CA certificate. The value indicates the number of additional non-self-signed certificates that may appear in the

path before **All issuance policies** is no longer permitted. For example, if the extension is set to value=1, then **All issuance Policies** can be processed in a particular certificate where this extension is present and in certificates issued by this CA, but not in additional certificates in the chain.

- **Authority Information Access (AIA)** This extension may consist of two access methods: **Certification Authority Issuer** and **On-line Certificate Status Protocol**. The **Certification Authority Issuer** method is used to retrieve a particular certificate issuer certificate if it is not already cached on the client. As with the **CDP** extension, clients try to use URLs in the same order as they are published in the extension. **On-line Certificate Status Protocol** is an alternate method to perform certificate revocation validation. In the case of **CDP**, the client has to download a whole CRL, which can be several megabytes and therefore will consume a lot of time and disk space for caching. To resolve this issue, a Certification Authority can publish URLs to an OCSP Responder. In that case, a client sends a short request that contains a certificate serial number and the OCSP Responder will return the certificate revocation status: Valid or Revoked. OCSP Responder usage can dramatically reduce the time and memory required for the certificate revocation validation process. URLs are processed until the certificate status is determined or all URLs fail.
- **Subject Information Access** This extension contains information on how to access information and services for the subject of the particular certificate where this extension is populated. When the certificate is a CA certificate, the information can include certificate validation services and/or the CA policy. When the certificate is an end entity certificate, the extension may contain information about the services offered by the certificate subject and how to access those services.

As mentioned earlier, organizations can also define their own extensions. For example, Microsoft has introduced some vendor-specific extensions, such **CA Version**:

- **CA Version** This extension indicates the particular Certification Authority certificate and key index in the case in which the CA has more than one signing certificate and one or more public and private key pairs. This extension allows clients and tools to determine the correct file or entry name when a CA certificate or CRL is published to Active Directory by the CA or third-party tools. Windows CA also uses this extension to determine the correct file publishing paths and URLs that will be published in issued certificates.

Certificate Revocation List

Public key infrastructure uses digital certificates with public and private key pairs. The public key may be transferred across un-trusted or unsecured networks without any security concerns. The private key must be securely protected by the key holder (owner). If the private key is stolen, lost, or became known to an unauthorized person, it is necessary to stop private key usage and trust. For example, suppose a user held a private key and associated with it a signing certificate, and the user's laptop was stolen. A malicious user can use this stolen certificate and private key to create rogue digital signatures and impersonate the legitimate user.

To avoid misuse of a lost or stolen private key, the certificate and the associated private key must be revoked. When a certificate is revoked, it is considered un-trusted even if the issuer is trusted. Therefore, if a malicious user attempts to sign a document using a revoked private key, digital signature validation will fail because the private key and the certificate are un-trusted.

On a regular basis, Certificate issuers (Certification Authorities) issue digitally signed files that contain revoked (un-trusted) certificate serial numbers. These files are called Certificate Revocation Lists (CRLs). When a client application validates a digital signature, the application should check the digital signature certificate's serial number against the most recent CRL from the issuer. If the serial number is not listed in the CRL, the digital signature may be considered as valid; otherwise, the signature should be rejected. Most certificate-aware applications are written so that if certificate revocation status cannot be verified (for example, the client cannot obtain the most recent time-valid CRL), the certificate is rejected. Therefore it is very important that the most recent CRL for each CA be publicly available. Since CRLs are signed using the CA's private key, they can be distributed across unsecure or un-trusted networks without any additional security measures.

Certificate Revocation Lists (CRLs) can be separated by scope to reduce each CRL's size. For example, a CA can issue separate CRLs for CA certificates and for end entity certificates. In that case, in a CA certificate validation process, a client will obtain only the first CRL. When an end entity certificate is to be validated, the application will need to obtain a CRL with the end entity scope. CAs may implement other scope types based on, for example, an LDAP tree (the CA will issue a separate CRL for each LDAP tree). Windows CAs use the **All certificates** scope for each key pair, which means that each CRL contains all revoked certificates regardless of certificate type or LDAP path.

Internet PKI currently supports CRL version 1 and version 2, discussed in the sections that follow.

X.509 Certificate Revocation List Version 1

CRL version 1 contains several mandatory fields:

- **Version** CRL version number. Value can be 1 or 2.
- **Issuer** The X.500 distinguished name (DN) of the CRL issuer. This field must be the same as the issuer certificate's **Subject** field and is used by the certificate chaining engine (CCE).
- **Effective date** The date and time that the CRL becomes valid.
- **Next update** The date and time this CRL is valid through. If the current date and time is not between the **Effective date** and the **Next update**, then the CRL is not valid for certificate revocation checking.
- **Signature algorithm** The algorithm used by the Certification Authority in the signing process.
- **Signature hash algorithm** The algorithm used by the Certification Authority in computing the CRL signature hash.
- **Revocation list** The list of certificate serial numbers revoked by the CA and the date when each certificate was revoked. This list may also contain reason codes indicating why each certificate was revoked.

CRL publishing periods are determined by each CA separately. For example, high-load CAs may issue certificates every 2-3 days, while other CAs may issue CRLs every 3 months. It is recommended to issue CRLs as frequently as possible. However, if the CRL is large (that is, it contains a large number of revoked certificates) and the CRL publishing period is very short, the client applications have to download a new CRL each time one is issued, which would consume a lot of network bandwidth. Additionally, there is no way to renew a CA signing key pair when a CA issues a Version 1 CRL.

To address this and other issues, RFC5280 defines the X.509 Version 2 CRL profile with additional extensions.

X.509 Certificate Revocation List Version 2

The X.509 V2 CRL profile introduces additional extensions and delta (or incremental) CRL support for situations where it is impractical to issue full CRLs at short intervals. While the base CRL contains all certificates revoked since the CA key was introduced (CA keys are normally used for 5-20 years), the next delta CRL contains only the certificates that have been revoked since the last base CRL was published. For example, suppose a base CRL is issued every 7 days and a delta CRL is issued every day. When a new base CRL is published, it contains all revoked certificates for the CA. The next day, two certificates are revoked. Without delta CRLs, clients will not recognize those two certificates as revoked until seven days after revocation. With delta CRLs, clients will recognize those certificates as revoked within 24 hours after certificate revocation. When a new base CRL is published, the delta CRL clears its revocation information. Therefore delta CRLs provide the following advantages:

- It is possible to publish CRLs more frequently without adversely affecting network load.
- Revocation information is more accurate and up-to-date.
- While base CRLs are fairly large, delta CRLs are quite small.

Common CRL Extensions

The X.509 V2 CRL profile provides for the following common extensions:

- **Authority key identifier (AKI)** Identifies the public key corresponding to the private key used to sign the certificate. It may contain either the key identifier (public key hash) or the issuer name and issuer certificate serial number. This extension must be the same as the **Subject Key Identifier** in the issuer's certificate. Clients use this extension to build a correct chain (certification path) to select a proper issuer certificate if the issuer has multiple signing certificates.
- **CRL number** The number of the CRL. When a new CRL is issued, the CRL number is incremented by one. This number must persist in the delta CRL to indicate the base CRL that the delta CRL corresponds to. For example, when the current base CRL number is 10, delta CRLs that are issued must contain the number 10. When next base CRL (11) is issued, delta CRLs will contain the number 11.
- **Delta CRL indicator** This is a critical CRL extension that identifies a CRL as being a delta CRL. Delta CRLs contain updates to revocation information previously distributed, rather than all the information that would appear in a complete CRL. The **Delta CRL indicator** extension contains a single value of type 'CRL number'. The CRL number identifies the CRL, complete with a given scope that was used as the starting point in the creation of the delta CRL.
- **Issuing Distribution Point (IDP)** This is a critical CRL extension that identifies the CRL distribution point and scope for a particular CRL, and indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, attribute certificates only, or a limited set of reason codes. This extension is supported by Widows CA, but is not used by default.
- **Freshest CRL** This extension may exist in base CRLs only. Clients use this extension to locate the latest delta CRL. When a client application performs certificate validation, the application retrieves the base CRL using the **CDP** extension in the certificate. If the certificate's serial number is not listed in the base CRL, the application will retrieve the most recent delta CRL using the **Freshest CRL** extension. If the CA does not use delta CRLs, this extension is omitted.
- **Authority Information Access (AIA)** This extension is used to retrieve a particular CRL issuer certificate if it is not already cached on the client.

Certificate Stores and Containers

For certificate storage, Microsoft uses logical certificate stores, also referred to as certificate containers. A certificate store location can be one of the following:

- **LocalMachine** (computer account) The containers in this store location hold certificates that are used by the Local System account. All stored certificates (except those stored in the **Personal** and **Certificate Enrollment Request** containers) are propagated to the user store locations and affect all user accounts logged on to this computer.
- **CurrentUser** (current user account) A separate store location is created for each logged-on user account. For example, User1 has her own certificate store location, and User2 has his own certificate store location.

Each store location holds a number of logical containers specific to the purpose of the stored certificates:

- **Personal (My)** Contains certificates associated with a private key controlled by the user or computer. When a certificate private key is used, the application searches this container for the appropriate certificate along with the associated private key.
- **Trusted Root Certification Authorities (ROOT)** Contains self-signed certificates from implicitly trusted certification authorities (CAs). Each certificate chain must chain up to a certificate presented in the self-signed form. This self-signed certificate is the "root certificate" or "trusted anchor." However, not all root certificates can be considered as trusted. You should carefully choose which new certificates are considered as trusted.
- **Enterprise Trust (trust)** Contains Certificate Trust Lists (CTL) typically used to trust self-signed certificates from other organizations. For example, the Key Management Server adds its certificate to this container.
- **Intermediate Certification Authorities (CA)** Contains certificates issued to subordinate CAs in the certification hierarchy. These certificates are typically used by the certificate chaining engine to build certificate chains.
- **Trusted Publishers (TrustedPublisher)** Contains explicitly trusted signing certificates. Although digital signature certificates normally chain up to the trusted root certification authority, many applications (such as Microsoft Office and Windows PowerShell) require a particular signing certificate to be stored in this container in order to trust signatures from that particular signer. This means that a digital signature-aware application may trust one signing certificate but not trust another signing certificate even if both certificates are issued by the same certification authority.
- **Untrusted Certificates (Disallowed)** Contains certificates that have been explicitly identified as un-trusted. By default, this container already contains two certificates. It is strongly recommended NOT TO REMOVE them from the container. For details, see Microsoft's article at <http://support.microsoft.com/kb/293817>.
- **Third-Party Root Certification Authorities (AuthRoot)** Contains trusted root certificates from CAs outside the internal certificate hierarchy. Certificates from the Microsoft Root Certificate Program are stored in this container. For more information about the Microsoft Root Certificate program, see Microsoft's article at <http://support.microsoft.com/kb/931125>.
- **Trusted People (TrustedPeople)** Contains certificates issued to users or entities that have been explicitly trusted. Most often, these are self-signed certificates or certificates explicitly trusted by an application such as Microsoft Outlook. To share an EFS-encrypted file with other parties, you must have their certificate in this store.

- **Certificate Enrollment Requests (REQUEST)** Contains pending or rejected certificate requests. When a certification authority issues a certificate in response to a request, you need to install the certificate response to this container using a special utility, such as CertReq.exe.
- **Smart Card Trusted Roots (SmartCardRoot)** Contains trusted smart card certificates.
- **Other People (AddressBook)** Contains certificates issued to users or entities that have been implicitly trusted. Thus, this container holds certificates that have been added to Outlook contacts.
- **Active Directory User Object (UserdS)** Contains the user object certificate or certificates published in Active Directory.

Certain applications may also create their own custom containers. For example, the System Center Operations Manager agent creates the Operations Manager container in the local computer store location.

Active Directory Certificate Containers

In an Active Directory domain environment, Microsoft supports some additional forest-wide certificate containers, listed in the table that follows.

NAME	LDAP PATH	DESCRIPTION
NTAuthCA	CN=NTAuthCertificates, CN=Public Key Services, CN=Services, CN=Configuration, DC=ForestRootDomain	Contains all certification authorities' certificates that are trusted to issue smart card and other user authentication certificates. In order for users to authenticate to a domain using a smart card, the smart card certificate issuer certificate must be published in NTAuthCA . Certificates from this container are automatically downloaded by computers within the current forest.
SubCA	CN=AIA, CN=Public Key Services, CN=Services, CN=Configuration, DC=ForestRootDomain	Contains many different types of CA entries with their certificates. Certificates from this container are automatically downloaded by clients within the current forest to the Intermediate Certification Authorities container in the local store.
CDP	CN=CDP, CN=Public Key Services, CN=Services, CN=Configuration, DC=ForestRootDomain	Contains enterprise CAs' certificate revocation lists (CRLs). A separate subcontainer is created for each CA computer. CRLs from the CDP container are not automatically downloaded by clients. The only purpose of this container is to download particular a CA's CRL using the corresponding URL in the certificate's CRL Distribution Point extension.
RootCA	CN=Certification Authorities, CN=Public Key Services, CN=Services, CN=Configuration, DC=ForestRootDomain	Contains trusted, self-signed certificates without private keys. Certificates from this container are automatically downloaded by clients within the current forest to the Trusted Root Certification Authorities container, so you should carefully choose which certificates will be considered as trusted.

Certificate Management Cmdlets Overview

This section provides a short description of each of the certificate management cmdlets that are included in the ActiveRoles Management Shell for Active Directory. For detailed information, see [Cmdlet Reference - X.509 Certificate Management](#) earlier in this document.

- **Get-QADLocalCertificateStore** Retrieves a certificate store location and available stores in that location.
- **New-QADLocalCertificateStore** Creates a new certificate store (container) in the specified store location.
- **Remove-QADLocalCertificateStore** Removes a certificate store (container) along with its contents from the specified store location.
- **Get-QADCertificate** Retrieves X.509 certificates matching the criteria you specify.
- **Where-QADCertificate** Implements various filters to determine which certificate or certificate collections are passed along the pipeline.
- **Add-QADCertificate** Adds a certificate object to the specified certificate container.
- **Remove-QADCertificate** Removes a certificate from the certificate container.
- **Import-QADCertificate** Imports a certificate from a file or other applicable structure (for example, an ASN.1 DER encoded byte array) to the internal certificate object.
- **Export-QADCertificate** Exports a certificate or certificate chain from an internal certificate object to a file or other applicable structure (for example, an ASN.1 DER encoded byte array or Base64 string).
- **Publish-QADCertificate** Publishes a certificate to the specified Active Directory container (see Active Directory Certificate Containers).
- **Unpublish-QADCertificate** Removes a certificate from the specified Active Directory container (see Active Directory Certificate Containers).
- **Show-QADCertificate** Displays a textual representation of an X.509 certificate.
- **Edit-QADCertificate** Shows a certificate in a GUI similar to Windows Explorer, and allows you to edit certain certificate properties.
- **Remove-QADPrivateKey** Removes a private key from the certificate.
- **Get-QADPKIObject** Retrieves Active Directory containers that may contain published certificates.
- **Get-QADCertificateRevocationList** Retrieves certificate revocation lists from a certificate store or Active Directory.
- **Add-QADCertificateRevocationList** Adds a CRL to the specified certificate container.
- **Remove-QADCertificateRevocationList** Removes a CRL from the specified certificate container.
- **Import-QADCertificateRevocationList** Imports a CRL from a file or other applicable structure (for example, an ASN.1 DER encoded byte array).
- **Export-QADCertificateRevocationList** Exports a CRL to a file or other applicable structure (for example, an ASN.1 DER encoded byte array or Base64 string).
- **Publish-QADCertificateRevocationList** Publishes a CRL to the Active Directory CDP container (see Active Directory Certificate Containers).
- **Unpublish-QADCertificateRevocationList** Removes a CRL from an Active Directory CDP container (see Active Directory Certificate Containers).

Certificate Store, Certificate, and CRL Objects

This section covers the internal object structures of certificate stores, certificates, and CRLs.

Certificate Store

To view certificate stores you need to invoke the `Get-QADLocalCertificateStore` cmdlet:

```
PS C:\> Get-QADLocalCertificateStore | Format-Table -AutoSize
```

Name	Location
----	-----
ACRS	CurrentUser
AuthRoot	CurrentUser
CA	CurrentUser
Disallowed	CurrentUser
My	CurrentUser
REQUEST	CurrentUser
Root	CurrentUser
SmartCardRoot	CurrentUser
trust	CurrentUser
TrustedPeople	CurrentUser
TrustedPublisher	CurrentUser
UserDS	CurrentUser

The output lists all available stores (containers) within the current user store location.

Note that the `Format-Table` cmdlet is only used to format the output text for the purposes of this document. You don't need to use it in regular operations unless you want a specific tabular format.

Certificate

Here is an example of a certificate internal object output:

```
PS C:\> $cert = Get-QADLocalCertificateStore CA | Get-QADCertificate -SerialNumber '52c820137c85*'
```

```
PS C:\> $cert | Format-List
```

```
FriendlyName          :
IssuerDN              : OU=Class 2 Public Primary Certification Authority,
O="VeriSign, Inc.", C=US
SubjectDN             : CN=VeriSign Class 2 CA - Individual Subscriber,
OU="www.verisign.com/repository/RPA Incorp. By Ref., LIAB.LTD(c)98", OU=VeriSign Trust
Network, O="VeriSign, Inc."
IssuedTo              : VeriSign Class 2 CA - Individual Subscriber
IssuedBy              : Class 2 Public Primary Certification Authority
Expired              : True
Valid                 : False
CertTimeValid         : False
CertChainValid        : True
Revoked               : False
HasPrivateKey         : False
PrivateKeyProtected    : False
```

Quest One ActiveRoles Management Shell for Active Directory

```
PrivateKeyExportable      : False
ValidFrom                 : 1998.05.12. 3:00:00
ValidTo                   : 2004.01.07. 1:59:59
Version                   : 3
SerialNumber              : 52C820137C85A7EDF217CE82C8451673
Thumbprint                : 7B02312BACC59EC388FEAE12FD277F6A9FB4FAC1
PublicKey                 : {48, 129, 137, 2...}
PublicKeyString           :
0818902818100B5CB1A545E25B02C595F096BD0DAD64A4B119D1A0A3E7E2FB7655F176315E52CD020000CF
0BA6BAA5E49B168938325AC245FA2231C694DB83BDB7DDA8FC109CFA5583AB64BC4D4DBD8AE75FA8622992
2012860A5DBD530DF21705E4899AD215491D1DE5FFB3829531BE27A5358C50D5D1307B350C4064B39F854A
BB98B6912130203010001
KeyAlgorithm              : 1.2.840.113549.1.1.1 (RSA)
KeyAlgorithmParameters    : 0500
SignatureAlgorithm        : 1.2.840.113549.1.1.4 (md5RSA)
SubjectKeyIdentifier      :
KeyUsages                 : CrlSign, KeyCertSign
EnhancedKeyUsages         : {}
Policies                  : {2.16.840.1.113733.1.7.1.1 ()}
CDP                       : {URL=http://crl.verisign.com/pca2.1.1.crl}
SubjectAlternativeNames    : {}
IssuerAlternativeNames    : {}
AIA                       : {}
NameConstraints           :
Template                  :
CertificateAuthority       : True
CrossCertificate          : False
NativeObject              : [Subject]
                           CN=VeriSign Class 2 CA - Individual Subscriber,
OU="www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98", OU=VeriSign Trust
Network, O="VeriSign, Inc."
                           [Issuer]
                           OU=Class 2 Public Primary Certification Authority,
O="VeriSign, Inc.", C=US
                           [Serial Number]
                           52C820137C85A7EDF217CE82C8451673
                           [Not Before]
                           1998.05.12. 3:00:00
                           [Not After]
                           004.01.07. 1:59:59
                           [Thumbprint]
                           7B02312BACC59EC388FEAE12FD277F6A9FB4FAC1
```

The output is pretty detailed and many properties are self-explanatory. For example, two properties indicate that the certificate is expired: **Valid** and **CertTimeValid**. The **NativeObject** property contains the .NET Framework X509Certificate 2 object representation, so you might use this property for certificate operations not covered by the existing cmdlets. Also you can map the X509Certificate2 object representation to the native X509CertificateUI object.

Certificate Revocation List (CRL)

Here is an example of a Certificate Revocation List object output:

```
PS C:\> $CRL = Get-QADLocalCertificateStore CA | Get-QADCertificateRevocationList
PS C:\> $CRL[3] | Format-List

Version                : CRL_V2
SignatureAlgorithm     : 1.2.840.113549.1.1.2 (md2RSA)
Issuer                 : OU=VeriSign Commercial Software Publishers CA,
O="VeriSign, Inc.", L=Internet
IssuedBy               : VeriSign Commercial Software Publishers CA
Number                 : 0
BaseNumber              :
CertificateIndex        :
KeyIndex               :
Type                   : Base
EffectiveDate           : 2001.03.24. 2:00:00
NextUpdate              : 2004.01.08. 1:59:59
NextPublish             :
Entries                 : {Quest.ActiveRoles.ArsPowerShellSnapIn.Interop.CRLEntry,
Quest.ActiveRoles.ArsPowerShellSnapIn.Interop.CRLEntry,
Quest.ActiveRoles.ArsPowerShellSnapIn.Interop.CRLEntry}
RawData                 : {48, 130, 1, 177...}
```

Here we see an X.509 version 2 CRL object that contains three entries:

```
PS C:\> $CRL[3].Entries | Format-Table -AutoSize
```

SerialNumber	Reason	RevocationDate
-----	-----	-----
6A99374642CD54929C392437F790511B		2001.01.30. 2:01:24
FDABB14E08C756F5ED47F097FF400E75		2001.01.31. 2:00:49
22C2ADCD1A80757A5F5D9359435AE677		2000.08.31. 3:00:56

Each of these entries identifies the serial numbers of a revoked certificate. If someone presents a certificate with one of the serial number listed in the CRL and the **Issuer** field is the same as shown in the CRL object, the application will reject the certificate.

Working with Certificate Stores

There are three types of task in working with certificate stores: explore a certificate store, create a certificate store, and delete a certificate store.

Explore a Certificate Store (Container)

In order to access a particular store, you need to get the certificate store object using the `Get-QADLocalCertificateStore` cmdlet. For example, suppose we need to get the **Personal** store from the **LocalMachine** store location:

```
PS C:\> $Store = Get-QADLocalCertificateStore -StoreLocation LocalMachine -StoreName My
PS C:\> $Store | Format-Table -AutoSize
```

Name	Location
----	-----
MY	LocalMachine

This object can be passed to other cmdlets that will access the selected container. For security reasons, you cannot select certificate containers in multiple store locations, but you can select multiple certificate containers within a single store location.

If you don't specify any parameters for this cmdlet, all certificate containers within the **CurrentUser** store location will be selected. The following examples show how to select all containers in the **CurrentUser** and **LocalMachine** store locations, respectively:

```
PS C:\> $UserStore = Get-QADLocalCertificateStore
PS C:\> $UserStore | Format-Table -AutoSize
```

Name	Location
----	-----
ACRS	CurrentUser
AuthRoot	CurrentUser
CA	CurrentUser
Disallowed	CurrentUser
My	CurrentUser
REQUEST	CurrentUser
Root	CurrentUser
SmartCardRoot	CurrentUser
trust	CurrentUser
TrustedPeople	CurrentUser
TrustedPublisher	CurrentUser
UserDS	CurrentUser

```
PS C:\> $ComputerStore = Get-QADLocalCertificateStore -StoreLocation LocalMachine
PS C:\> $ComputerStore | Format-Table -AutoSize
```

Name	Location
-----	-----
AddressBook	LocalMachine
AuthRoot	LocalMachine
CA	LocalMachine
Disallowed	LocalMachine
KRA	LocalMachine
MY	LocalMachine
Remote Desktop	LocalMachine
REQUEST	LocalMachine
ROOT	LocalMachine
SmartCardRoot	LocalMachine
trust	LocalMachine
TrustedPeople	LocalMachine
TrustedPublisher	LocalMachine

You can use these commands to search for a particular certificate in all certificate containers. For example, to view all certificates within a particular certificate container, you need to pass the container name and location to the Get-QADCertificate cmdlet:

```
PS C:\> Get-QADLocalCertificateStore My LocalMachine | Get-QADCertificate | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Contoso CA	contoso-DC2-CA	2010.03.06.	2015.03.05.
contoso-DC2-CA	dc2.contoso.com	2010.01.06.	2011.01.06.
contoso-DC2-CA	dc2.contoso.com	2010.01.11.	2011.01.11.
contoso-DC2-CA	dc2.contoso.com	2010.04.24.	2010.05.08.
Contoso CA	contoso-DC2-CA	2009.03.30.	2011.03.30.
contoso-DC2-CA	dc2.contoso.com	2009.05.11.	2011.03.30.
contoso-DC2-CA	dc2.contoso.com	2010.04.12.	2011.04.12.
Contoso CA	contoso-DC2-CA	2010.03.06.	2015.03.05.

Note that in certain cases you may encounter time delays. Get-QADCertificate performs certificate validation by passing the certificate through the certificate chaining engine (CCE). If the URLs provided in the CDP/AIA extensions are not downloadable, CCE will wait up to 10 second until it reports timeout.

If you want to filter certificates using a custom filter, you can use filter parameters for Get-QADCertificate. The following example filters certificates by using the CertificationAuthority parameter:

```
PS C:\> Get-QADLocalCertificateStore My LocalMachine | Get-QADCertificate  
-CertificateAuthority Format-Table -AutoSize
```

IssuedBy -----	IssuedTo -----	ValidFrom -----	ValidTo -----
Contoso CA	contoso-DC2-CA	2010.03.06.	2015.03.05.
Contoso CA	contoso-DC2-CA	2009.03.30.	2011.03.30.
Contoso CA	contoso-DC2-CA	2010.03.06.	2015.03.05.

To filter certificates that are applicable for code signing, you can use the following command:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate -AnyEnhancedKeyUsage "Code  
Signing" | Format-Table -AutoSize
```

IssuedBy -----	IssuedTo -----	ValidFrom -----	ValidTo -----
contoso-DC2-CA	Administrator	2010.01.11.	2011.01.11.
contoso-DC2-CA	Administrator	2010.03.16.	2011.03.16.

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate -AnyEnhancedKeyUsage  
1.3.6.1.5.5.7.3.3 | Format-Table -AutoSize
```

IssuedBy -----	IssuedTo -----	ValidFrom -----	ValidTo -----
contoso-DC2-CA	Administrator	2010.01.11.	2011.01.11.
contoso-DC2-CA	Administrator	2010.03.16.	2011.03.16.

These two examples show that you can specify either the user-friendly name or the unique OID value. Not all OIDs have user-friendly names, however. A list of well-known OIDs is provided in the following article: <http://msdn.microsoft.com/library/aa378132.aspx>. The list of well-known OIDs is not limited to those in this article. In an Active Directory domain environment, you may define forest-wide custom OIDs and user-friendly names.

For instance, if you want to add (or map) a friendly name for a particular EnhancedKeyUsage OID and your forest has at least one Enterprise CA, then you can define a friendly name as follows:

1. Open the **Certificate Template** MMC snap-in (certtmpl.msc).
2. Open any version 2 certificate template (where the template major version is greater than 100).
3. Click the **Extension** tab.
4. On the **Extension** tab, select the **Application Policies** extension, click **Edit**, and then click **Add**.
5. In the **Add Application Policy** window, click **New**.
6. In the **Name** field, type a friendly name for the OID. In the **Object Identifier** field, remove the auto-generated OID and type your own custom OID.
7. Click **OK**. You don't need to save changes to the certificate template, so close the other windows by clicking the **Cancel** button.

The following example shows how that works:

```
PS C:\> Get-QADLocalCertificateStore My localmachine | Get-QADCertificate
-AnyEnhancedKeyUsage "Remote desktop*" | Select EnhancedKeyUsages
```

```
EnhancedKeyUsages
-----
{Remote Desktop Authentication}
```

Remote Desktop Authentication is not a well-known OID, and has the following OID value: 1.3.6.1.4.1.311.54.1.2. Using the steps above you can define custom names for various OIDs and they will be recognized within your Active Directory forest.

Create a Certificate Store (Container)

In certain scenarios, you may need to create a new certificate store (container). To create one, you need to specify a new container name with the `New-QADLocalCertificateStore` cmdlet:

```
PS C:\> New-QADLocalCertificateStore MyCustomUserContainer
```

Name	Location
-----	-----
MyCustomUserContainer	CurrentUser

```
PS C:\> New-QADLocalCertificateStore -StoreLocation LocalMachine
MyCustomComputerContainer
```

Name	Location
-----	-----
MyCustomComputerContainer	LocalMachine

These examples create new containers in the `CurrentUser` and `LocalMachine` store locations, respectively. However, applications usually create all required certificate containers, so you should not create new containers unless a particular application requires them.

Delete a Certificate Store (Container)

When a certificate container is no longer required, you can delete it. Certain applications create their own custom containers during the installation process. However, usually the containers are not removed when you uninstall the application. For example, when you install System Center Operations Manager Agent, it creates a custom certificate container. When you uninstall System Center Operations Manager Agent, the container is not removed.

The process for deleting a certificate container is the same as the process for creating a container except that you use Remove-QADLocalCertificateStore:

```
PS C:\> Remove-QADLocalCertificateStore MyCustomUserContainer
```

Confirm

Are you sure you want to perform this action?

Remove local certificate store: Name = MyCustomUserContainer, Location = CurrentUser

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): Y

Name	Location
----	-----
MyCustomUserContainer	CurrentUser

```
PS C:\> Remove-QADLocalCertificateStore -StoreLocation LocalMachine
```

```
>> MyCustomComputerContainer, "Operations Manager"
```

```
>>
```

Confirm

Are you sure you want to perform this action?

Remove local certificate store: Name = MyCustomComputerContainer, Location = LocalMachine

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): A

Name	Location
----	-----
MyCustomComputerContainer	LocalMachine
Operations Manager	LocalMachine

Note that the deletion of a certificate container is quite a sensitive operation, because this process also deletes all certificates in that container. Therefore you always will be prompted for confirmation of the operation and the -Force parameter is not supported for container removal.

Adding Certificates to a Certificate Store

The operation of adding a certificate to a certificate store consists of two steps:

1. Import the certificate from a file or other applicable data using Import-QADCertificate.
2. Add the certificate to the specified store using Add-QADCertificate.

About Certificate File Types

There are several types of certificate file. You can recognize them by the file name extension, as explained below.

Single certificate with a CER or DER file name extension

This is a simple certificate file type that may contain only one certificate per file, and files contain only the public certificate part (without the private key).

Personal Information Exchange (Pkcs12) with a PFX or P12 file name extension

This is a certificate container file and the only file type that may contain certificate private keys. Typically, Pkcs12 is protected by an external password. PFX or P12 files may contain multiple certificates with multiple private keys. Normally, PFX or P12 files are used to distribute certificates with private keys only. The file may contain a full (if possible) or partial certificate chain. The certificate chain contains all issuer certificates. These additional certificates help to build a certificate chain for a particular certificate.

Cryptographic Message Syntax Standard (Pkcs7) with a P7B file name extension

This is another container file type. The Pkcs7 format is generally used to export multiple certificates and/or certificate chains without private keys. For example, consider a root CA without network access, known as an offline CA. When you request a certificate from an offline CA, the CA may issue a certificate with a full certificate chain that simplifies certificate chain building when you install a CA response. Another case when this format is very useful is the situation where you deploy a multi-tier CA hierarchy. When two or more tiers are implemented, an offline CA is usually deployed in a workgroup environment. To deploy an enterprise subordinate CA (one that will be subordinate to the offline root) in an Active Directory environment, you have to install all certificates and CRLs from all parent CAs. The use of Pkcs7 files will simplify this process.

Serialized store with an SST file name extension

A serialized store file contains the encoded certificate and its extended properties, such as friendly name, cross-certificates, additional OCSP URLs, and Extended Validation OIDs. Extended properties are associated with a certificate and are not part of the certificate as issued by a certification authority. This can be very helpful when you need to back up or move a certificate store to another computer. Like Pkcs7 files, a serialized store file cannot contain certificates with their private keys.

Import a Single Certificate

To import certificates from an external data source, use the Import-QADCertificate cmdlet. The following examples show how to import a simple certificate to a certificate object from various data sources.

```
PS C:\> Import-QADCertificate C:\Certs\caxhg.cer | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
contoso-DC2-CA	contoso-DC2-CA-Xchg	2010.04.24.	2010.05.29.

```
PS C:\> $bytes = [System.IO.File]::ReadAllBytes("C:\Certs\caxhg.cer")
PS C:\> Import-QADCertificate -RawData $bytes | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
contoso-DC2-CA	contoso-DC2-CA-Xchg	2010.04.24.	2010.05.29.

```
PS C:\> dir cert:\CurrentUser\my | Import-QADCertificate : Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
contoso-DC2-CA	contoso-DC2-CA-Xchg	2010.04.24.	2010.05.29.

You may specify either the path to the file or the encoded raw data. In the first example, we specify the path to a certificate file. In the second example, we extract certificate data as an ASN.1 DER encoded byte array, and then, by using the `-RawData` parameter, we import that raw data to the certificate object. The last example demonstrates how to import a certificate from a native .NET Framework X509Certificate2 object: we pass the `Cert: PowerShell Provider` output to the `Import-QADCertificate cmdlet`.

If you don't want to see the output, you can use the `Out-Null` cmdlet. For example:

```
PS C:\> Import-QADCertificate C:\Certs\caxhg.cer | Out-Null
```

Import a Pkcs7 Certificate Container

In the same manner you can import certificates from Pkcs7 formatted files:

```
PS C:\> Import-QADCertificate C:\Certs\xchg.p7b | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Contoso CA	contoso-DC2-CA	2010.03.06.	2015.03.05.
Contoso CA	Contoso CA	2009.02.15.	2029.05.19.
contoso-DC2-CA	contoso-DC2-CA-Xchg	2010.04.24.	2010.05.29.

The `xchg.p7b` file contains a full chain for the CA Exchange certificate (the last certificate). The first certificate in the output is the CA Exchange certificate issuer and the second certificate is the chain root certificate.

Import a Serialized Store

While the `Cert` and `Pkcs7` formats do not support extended properties for certificates, a serialized store has this support. Here is an example of importing a serialized store with extended properties:

```
PS C:\> $cert = Import-QADCertificate C:\Certs\SerializedStore.sst
PS C:\> $cert | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
VeriSign Commercial Software Publishers CA	Microsoft Corporation	2001.01.31.	2002.02.01.
VeriSign Commercial Software Publishers CA	Microsoft Corporation	2001.01.30.	2002.01.31.

```
PS C:\> $cert[0].FriendlyName
```

```
Fraudulent, NOT Microsoft
```

As you can see, the imported certificates still hold their extended information, such as a friendly name.

Import a Pkcs12 Certificate with a Private Key

The only certificate file type that can hold certificate private keys is Pkcs12. With Pkcs12, some additional parameters are available:

- **Password** You must specify the password in SecureString format in order to import a certificate.
- **ImportFlags** The list of valid import flags is documented in Microsoft's article [X509KeyStorageFlags Enumeration](http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keystorageflags.aspx) at <http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.x509keystorageflags.aspx>.

The **Exportable** and **UserProtected** flags should be used very carefully. You need to consider the following requirements and recommendations:

- **Exportable** This flag marks private key as exportable. You may have to use this option in order to perform a backup of the key in the future. If this parameter is not asserted, the private key is marked as non-exportable by default.

For security reasons, you should NOT mark private key as exportable for signing and authentication certificates. This is because these certificate types are very sensitive and may fraudulently impersonate a legitimate user.
- **UserProtected** This flag causes a CryptoAPI dialog box to prompt you how to protect the particular private key. You can protect the private key by an external password or consent window (available for Windows Vista and higher). This will prevent applications from using the certificate private key without user approval. This flag is strongly recommended for signing certificates to ensure that the user, but not any malicious application, performs the signing operation.

You must NOT specify the **UserProtected** flag for the following certificate types:

- **Computer certificates** When a protected private key is used, the local security authority (LSA or lsass.exe) sends a confirmation dialog box to a user account session interactive desktop. When computer certificates are used by the Local System account and have no interactive desktop, the user will never receive this dialog box and certificate will become unusable.
- **Encrypting File System (EFS) certificates** When EFS certificates are used by users, the LSA should send a confirmation dialog box to the user's interactive desktop. However, this is not possible for EFS certificates. When a user tries to encrypt a file, the LSA generates a symmetric encryption key in the system kernel and encrypts the file. After that, the LSA extracts the EFS public key to encrypt the symmetric encryption key. While the symmetric key is generated in the kernel mode, the confirmation dialog box has access to that mode too. To prevent this security risk, exposure of the LSA to the desktop is not allowed. If you do implement strong private key protection for an EFS certificate, EFS will silently fail because the confirmation dialog box is never exposed to the user.

The following examples demonstrate Pkcs12 import techniques:

```
PS C:\> $password = Read-Host "Enter password" -AsSecureString
```

```
Enter password: *****
```

```
PS C:\> $cert = Import-QADCertificate C:\Certs\sample.pfx -Password $password  
-ImportFlags "Exportable"
```

```
PS C:\> $cert | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

```
PS C:\> $password = ConvertTo-SecureString "P@ssw0rd" -AsPlainText -Force
```

```
PS C:\> $password
```

```
System.Security.SecureString
```

```
PS C:\> $cert = Import-QADCertificate C:\Certs\sample.pfx -Password $password  
-ImportFlags "Exportable"
```

```
PS C:\> $cert | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

Since Pkcs12 requires a password in the SecureString format, you can use either of two common ways to retrieve it:

- For interactive operations, use the Read-Host cmdlet with the AsSecureString parameter.
- For automated operations (without user input), you can use the ConvertTo-SecureString cmdlet. When you use the ConvertTo-SecureString cmdlet, you must specify the -Force parameter.

Just like a serialized store, a Pkcs12 file can hold certificate extended information. Thus, you can use the following command to view the friendly name of the imported certificate:

```
PS C:\> $cert.FriendlyName
```

```
Encrypting File System
```

Add Imported Certificates to a Store

You can use the examples in the previous sections to import certificates from a file to a \$cert variable. This section demonstrates how to add the imported certificates to a certificate store. The following examples add an imported certificate to the current user's Personal store:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate
PS C:\> Get-QADLocalCertificateStore My | Add-QADCertificate -Certificate $cert |
Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

The first command in this example shows us an empty Personal container. The second command adds a certificate to the specified store, and the last command shows us that the Personal store is not empty now.



When you add a certificate with a private key, you should specify only the 'My' store container because many applications find certificate private keys only in the 'My' store container. Therefore it is uncommon to add certificates with their private keys to other store containers.

To add the specified certificate to multiple stores in a particular store location (CurrentUser or LocalMachine), you can specify multiple containers, as follows:

```
PS C:\> Get-QADLocalCertificateStore trust, TrustedPublisher | Add-QADCertificate
-Certificate $cert
```



When you add a certificate to the 'Trusted Root CAs (Root)' store, Windows presents you with a security warning dialog box, prompting you to confirm the origin of the certificate. The dialog box requires your confirmation in order to trust the certificate. There is no way to avoid that dialog box, which prevents malicious applications from silently installing an untrusted certificate.

To add certificate to a local computer store location, you need to specify that store location in the Get-QADLocalCertificateStore cmdlet:

```
PS C:\> Get-QADLocalCertificateStore My | Add-QADCertificate -Certificate $cert |
Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.



To maintain certificate stores in the local computer store location, you must be logged on as a local administrator. If User Account Control (UAC) is enabled, run PowerShell by clicking **Run as Administrator** on the shortcut menu.

Add an Imported Certificate to a User Account

You can also add a certificate to an Active Directory user account. For example, if someone wants to grant you access to encrypted data, he or she selects your published certificate from the Active Directory user account. Another scenario is where someone wants to send you encrypted e-mail. The e-mail sender selects your e-mail encryption certificate from Active Directory and encrypts the mail message using your published certificate public key.

The following example checks user account certificates and adds a new one:

```
PS C:\> Get-QADUser Administrator | Get-QADCertificate
PS C:\> Get-QADUser Administrator | Add-QADCertificate -Certificate $cert | Format-Table
-AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

```
PS C:\> Get-QADUser Administrator | Get-QADCertificate | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Administrator	Administrator	2010.04.05.	2110.03.12.

The first command in this example demonstrates that the Administrator user account has no certificates in Active Directory. The second command adds a certificate to the Administrator user account, and the last command shows us that now Administrator has a certificate published in Active Directory.

Advanced Techniques

It is common to install a certificate along with its certificate chain. You may also want to add the end certificate to the 'My' container, all intermediate CA certificates to the 'CA' container, and the root CA certificate to the 'Root' container in the LocalMachine store location. To achieve this, you need to distinguish those certificates:

- Certificates that are issued to end entities have the CertificateAuthority property set to False.
- Intermediate CA certificates have the CertificateAuthority property set to True, and they are not self-signed (the Issuer and Subject fields are different).
- Root CA certificates have the CertificateAuthority property set to True and are self-signed (the Issuer and Subject field are equal).

```
PS C:\> $certs = Import-QADCertificate C:\Certs\xchg.p7b
PS C:\> $certs | ForEach-Object {
    if ($_.CertificateAuthority -eq $false) {
        Get-QADLocalCertificateStore My | Add-QADCertificate -Certificate $_
    } elseif ($_.CertificateAuthority -eq $true -and $_.SubjectDN -ne $_.IssuerDN) {
        Get-QADLocalCertificateStore CA | Add-QADCertificate -Certificate $_
    } elseif ($_.CertificateAuthority -eq $true -and $_.SubjectDN -eq $_.IssuerDN) {
        Get-QADLocalCertificateStore Root LocalMachine | Add-QADCertificate
        -Certificate $_
    }
}
```

In this example, we add the self-signed root certificate to the LocalMachine store location, but the other certificates to the CurrentUser store location. This is because many applications are configured to check the certification path to any certificate stored in the Trusted Root CAs store in the LocalMachine store location.

Root certificates are self-signed and their trust is provided by external means. In the Windows operating systems, this means is the presence of the certificate in the Trusted Root CAs store. If you add a CA's root certificate to that store in the CurrentUser store location, all applications running in the context of the current user will trust any certificates issued by that CA. This poses a security risk, because a malicious user could install an untrusted certificate to the Trusted Root CAs store without a user prompt. Technically it is possible to avoid the warning dialog that appears when you install a root certificate. Today, many certificate-aware applications are designed to ignore the CurrentUser Trusted Root CAs store and use the store held in the LocalMachine store location. The most common scenario is Remote Desktop Client v7. This is a common issue when users install a root certificate to the CurrentUser store and are unable to establish a remote desktop connection using SSL. While RDC is running in the current user context, it checks the root certificate for presence in the Trusted Root CAs store in the LocalMachine store location.

Beginning with the Windows 7 release, there is a change in the behavior of the [X509Chain.Build\(\)](#) method. When the Build() method is invoked on an operating system prior to Windows 7 and Windows Server 2008 R2, the method will succeed if the chain root certificate is stored in the Trusted Root CAs store in the CurrentUser store location. When the Build() method is invoked on Windows 7 or Windows Server 2008 R2, the method will fail if the chain root certificate is not stored in the Trusted Root CAs store in the LocalMachine store location.

Exporting Certificates from a Certificate Store

This section covers the tasks of exporting certificates from certificate stores to a file and to encoded data format. There are several scenarios where you may want to export certificates, such as when you perform manual backups of your encryption certificates or when you want to install certain certificates on a different computer.

Simple Certificate Export

The following example exports a certificate to a file:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object
{Export-QADCertificate $_ -File c:\certs\$($_.SerialNumber).cer}
```

-----BEGIN CERTIFICATE-----

```
MIIC/jCCAeagAwIBAgIQaiNduoW52JVJ2TyiQ7kG0DANBgkqhkiG9w0BAQUFADASMRAwDgYDVQQDEwd2UG9kYW
5zMCAXDTEwMDQwNTEwNDExNloYDzIxMTAwMzEyMTA0MTE2WjASMRAwDgYDVQQDEwd2UG9kYW5zMIIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmAnXiQA5r7MxcLSS6ujhQhrr5DmcGIhuNteYglQvidxPZ1/Vq/9Zm3
u48Lg2taK2ldBpwRLicUurEB1hKbMMqSZvafYqTGBJ6GjdGjMgVQN4mC65Va3rY1KH838UT9ADo6Gq10n8GB3W
Jhe43APZi2/viXMDAA/fhFCA8h6rFHHupnfrw3Ej490D3iDIq8Oxbl6wvYrWlmOhQeJTMw7hw2G5ibfQur3IeP
lW/8yqSbKJYdhM8/23xPTtzVNrgIcjdU7Bro5WEcNJULBFbnMdIi5FNOIMmK9eeOA7nW0Pr6AoCpdYOeEXi1Nc
cqfy/6yclQeS0gqnNjHuShd0YBhctQIDAQABo04wTDAVBgNVHSUEDjAMBgorBgEEAY <...>
```

-----END CERTIFICATE-----

This example retrieves all certificates from the 'My' store in the CurrentUser store location and exports each to a separate CER file with the certificate serial number in the file name. The command also produces as output the certificates in the appropriate encoding (Base64 by default). This can be useful when you need to perform certain actions on the certificate within the current PowerShell session. If you don't want to see the output, add 'Out-Null' at the end of the command:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object  
{Export-QADCertificate $_ -File c:\certs\$(($_.SerialNumber).cer)} | Out-Null
```

This will avoid redirection of the output to the console. You may also choose different file naming formats. For example, the following command uses the Subject value instead of the serial number in the file name:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object  
{Export-QADCertificate $_ -File c:\certs\$(($_.IssuedTo).cer)} | Out-Null
```

Note that this command may fail if the Subject value contains any characters that are not valid in a file name, such as colon, comma, or slash characters. Therefore you need to ensure that the certificate Subject value does not contain invalid characters. Also you may have several certificates with the same Subject value. This will cause a subsequent certificate to overwrite the previously created certificate file. Therefore it is advisable to add something that would produce a unique file name, which can be accomplished by combining the IssuedTo and Thumbprint properties:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object  
{Export-QADCertificate $_ -File c:\certs\$(($_.IssuedTo)_$(($_.Thumbprint).cer)} | Out-Null
```

You may elect to export a certificate in an ASN.1 DER encoded byte array instead of Base64. To export certificates in a byte array format, you need to supply the -Encoding parameter:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object  
{Export-QADCertificate $_ -File c:\certs\$(($_.IssuedTo)_$(($_.Thumbprint).cer -Encoding  
byte)} | Out-Null
```

Encoding Options

In Internet PKI both encoding formats can be used. However, Base64 format is preferred for general use. For example, suppose you want to send your certificate to your friend or customer via e-mail or instant message. For security reasons, many e-mail clients, servers, and instant messengers reject certificate files. To work around those restrictions, you may have to add the exported certificates to a ZIP folder. However, you may copy a Base64-encoded certificate to the Clipboard and paste it into an e-mail message or instant messenger chat window. The recipient will be able to paste the received data into a file and save it with a CER extension. To accomplish this task, you can use the following command:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object  
{ExportQADCertificate $_}
```

This command will output certificates in Base64 encoding. Select the output certificate in the console (using a mouse or other pointing device), press Enter (this will copy the selected text to the Clipboard), and paste the Clipboard contents into an e-mail message or instant messenger chat window.

If you need a certificate in byte array format, add the '-Encoding byte' parameter to the command.

Simple certificate export is not limited to local certificate stores. You may also export certificates from an Active Directory user account using the following command:

```
PS C:\> Get-QADUser UserName | Get-QADCertificate | ForEach-Object
{Export-QADCertificate $_ -File c:\certs\$($_.IssuedTo)_$($_.Thumbprint).cer} | Out-Null
```

Export a Certificate with a Private Key

The only file type that may contain certificates along with their private keys is Pkcs12 or PFX. With this file type you may export multiple certificates to a single file. When exporting a certificate along with its private key, for security reasons it is advisable to have the export file include the full certificate chain.

The following examples demonstrate how to export certificates along with the private key and the certificate chain:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Where-Object
{$_.HasPrivateKey} | Export-QADCertificate -Format pfx -File C:\Certs\MyPfx.pfx
-Password (ConvertTo-SecureString P@ssw0rd -AsPlainText -force) | Out-Null
```

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Where-Object
{$_.HasPrivateKey} | Export-QADCertificate -Format pfx -Mode chain -File
C:\Certs\MyPfx.pfx -Password (ConvertTo-SecureString P@ssw0rd -AsPlainText -Force) |
Out-Null
```

These examples use an additional filter to select the certificates that contain a private key (by considering the HasPrivateKey property). The second command adds both the certificate, its private key and full certificate chain to a PFX file. This simplifies certificate installation on another computer, because all certificates required for end certificate validation are in place.

To export private keys, you must specify a password to protect the private key. If you omit the -Password parameter, the certificate is exported to a PFX file without the private key and an appropriate warning message will appear.

When exporting a private key, you may encounter the following error message: "Key not valid for use in specified state." This error message indicates that the private key is marked as non-exportable. To determine whether a given certificate has an exportable private key, you can check the certificate's 'PrivateKeyExportable' property. If the 'PrivateKeyExportable' property is set to False, you cannot export the certificate with its corresponding private key. If the certificate's private key is protected (the 'PrivateKeyProtected' property is set to True), the 'PrivateKeyExportable' property is set to False. There is no workaround to determine whether a certificate's private key is exportable when it is protected.

The PFX export format does not support Base64 encoding for exported data, so the -Encoding parameter has no effect if you specify the '-Format pfx' parameter.

When you export certificates with their corresponding private keys, you may want to remove the private key material from the store. You have the option to remove the certificate from the store or only remove the certificate's private key.

The following example exports the certificates along with their private keys and then removes the certificates from the certificate store:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Where-Object
{$_.HasPrivateKey} | Export-QADCertificate -Format pfx -Mode chain -File
C:\Certs\MyPfx.pfx -Password (ConvertTo-SecureString P@ssw0rd -AsPlainText -Force) |
Out-Null
```

```
PS C:\> Get-QADLocalCertificateStore My | Where-Object {$_.HasPrivateKey} |
Remove-QADCertificate
```

The following example exports the certificates along with their private keys and then removes each certificate's private key:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Where-Object
{$_.HasPrivateKey} | Export-QADCertificate -Format pfx -Mode chain -File
C:\Certs\MyPfx.pfx -Password (ConvertTo-SecureString P@ssw0rd -AsPlainText -Force) |
Out-Null
```

```
PS C:\> Get-QADLocalCertificateStore My | Where-Object {$_.HasPrivateKey} |
Remove-QADPrivateKey
```

You may use either of these techniques to remove the private key material from the store.

Export Multiple Certificates

The following example demonstrates how to export each certificate held in the 'My' store, along with the certificate chain, to a separate Pkcs7 file, without exporting private keys:

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | ForEach-Object
{Export-QADCertificate $_ -Format pkcs7 -Mode chain -File
C:\Certs\Myp7b_${_.thumbprint}.p7b} | Out-Null
```

The preceding command exports each certificate with its certificate chain to a separate file. In order to export all certificates to a single file, you need to switch the -Mode parameter from 'Chain' to 'Collection':

```
PS C:\> Get-QADLocalCertificateStore My | Get-QADCertificate | Export-QADCertificate
-Format pkcs7 -Mode collection -File C:\Certs\Myp7b.p7b | Out-Null
```

Although this command exports all certificates, it does not add the certificate chains to the export file.

As with a simple certificate export, you may export certificates from user account properties in Active Directory as follows:

```
PS C:\> Get-QADUser UserName | Get-QADCertificate | Export-QADCertificate -Format pkcs7
-Mode collection -File C:\Certs\Myp7b.p7b | Out-Null
```

By default, Base64 encoding is used. To export certificates using an ASN.1 DER encoded byte array format, supply the '-Encoding byte' parameter on the Export-QADCertificate cmdlet. Some considerations on what type of encoding to choose can be found in the section that covers simple export of certificates (see [Encoding Options](#)).

Working with Certificate Revocation Lists (CRLs)

It is common that a certificate revocation list (CRL) needs to be added to a certificate store. The following are some typical scenarios:

- You set up a new offline subordinate certification authority that will have no network access. When you install a CA certificate, the installation may fail if the parent CA's CRL is not previously installed in the store, because the CA server cannot validate its own certificate for revocation.
- The certificate URLs in the certificate CDP extension are unavailable for some reason (such as limited network connectivity or LDAP or Web server failure). Locally installed CRLs can be used by applications in the certificate status check process.

Add CRLs to a Certificate Store

CRL installation is similar to certificate installation and consists of two steps:

1. Import the CRL from a file.
2. Add the CRL to the certificate store.

To import the CRL from a file, use the `Import-QADCertificateRevocationList` cmdlet, as follows:

```
PS C:\> $crl = Import-QADCertificateRevocationList -File C:\CRLs\EVSecure2006.crl
```

Once you have the CRL imported, use the following command to add the CRL to the CA store in the `CurrentUser` store location:

```
PS C:\> Get-QADLocalCertificateStore CA LocalMachine | Add-QADCertificateRevocationList -CRL $crl | Format-Table -AutoSize
```

Type	Number	BaseNumber	KeyIndex	EffectiveDate	NextUpdate	Entries	Issuer
----	-----	-----	-----	-----	-----	-----	-----
Base	0			2010.04.02.	2010.04.16.	1187	CN=VeriSign..

Technically it is possible to import a CRL to any certificate store. However, you should use only the CA store to store CRLs in the `CurrentUser` or `LocalMachine` store location. Applications are typically designed to look up only the CA container for locally installed CRLs. Therefore you should avoid using stores other than CA to store CRLs. Also you should install CRLs to the `LocalMachine` store location, so both the current user and the local system will be able to use the locally stored CRLs.

To add multiple CRLs to the store, you can use the following sequence of commands:

```
PS C:\> $CRLs = dir C:\CRLs | Import-QADCertificateRevocationList
PS C:\> Get-QADLocalCertificateStore CA LocalMachine | Add-QADCertificateRevocationList -CRL $crls | Out-Null
```

The first command populates the `$CRLs` variable with an array of CRL objects. The second command adds all those CRLs to the CA certificate store.

Export CRLs from a Certificate Store

CRLs use a single CRL file format. Thus there are no CRL container file formats, and CRL export is quite simple. As certificates, CRL may be encoded using Base64 encoding or using an ASN.1 DER encoded byte array. By default, Base64 is used. The following examples export a CRL to a file using Base64 and an ASN.1 DER encoded byte array, respectively:

```
PS C:\> $CRL = Get-QADLocalCertificateStore CA LocalMachine |
Get-QADCertificateRevocationList
PS C:\> Export-QADCertificateRevocationList $CRL[0] -File C:\CRLs\custom.crl

-----BEGIN X509 CRL-----
MIIE3jCCA8YCAQEwDQYJKoZIhvcNAQEFBQAwRzETMBEGCgmSJomT8ixkARkWA2NvbTEXMBUGCgmSJomT8ixkAR
kWB2NvbNvc28xFzAVBgNVBAMTDmNvbNvc28tREMyLUNBFw0xMDAzMjAxMTEwNDFaFw0xMDAzMjgxMTMwNDFa
MIICpDAbAgoW2JlUAAAAA <...>
-----END X509 CRL-----

PS C:\> Export-QADCertificateRevocationList $CRL[0] -File C:\CRLs\custom.crl -Encoding
byte | Out-Null
```

In order to distinguish a Base64 encoded certificate from a CRL, headers and footers are used.

Some considerations on what type of encoding to choose (Base64 or ASN.1 DER) can be found in the section that covers export of certificates (see [Encoding Options](#)).

Remove a CRL from a Certificate Store

Normally you do not need to remove CRLs from a certificate store. The only reason is that you have an offline subordinate CA and too many parent CA CRLs are installed. In that case you might elect to remove expired CRLs from the store. The following example demonstrates CRL removal based on CRL issuer information:

```
PS C:\> $Store = Get-QADLocalCertificateStore CA LocalMachine
PS C:\> Get-QADLocalCertificateStore CA LocalMachine | Get-QADCertificateRevocationList
| WhereObject {$_.IssuedBy -like "contoso*"} | Remove-QADCertificateRevocationList
-Store $Store
```

Type	Number	BaseNumber	KeyIndex	EffectiveDate	NextUpdate	Entries	Issuer
----	-----	-----	-----	-----	-----	-----	-----
Base	380		0	2010.03.20.	2010.03.28.	18	CN=contoso...
Base	380		2	2010.03.20.	2010.03.28.	1	CN=contoso...

The first command in this example retrieves the certificate store. As recommended, the CA store in the LocalMachine store location is selected. The second command retrieves all CRLs that were issued by any CA whose name begins with 'Contoso', and then removes those CRLs from the store.

Managing Active Directory PKI-related Containers

This section covers Active Directory PKI-related containers and certificate management. All containers are described in the [Active Directory Certificate Containers](#) section of this document. Since all those containers reside in the forest's configuration naming context, each domain controller in the current forest maintains all PKI-related containers. If you publish a certificate or CRL to a PKI-related AD container, it is available in any domain in the current forest after the next replication event. If you remove (unpublish) a certificate or CRL from any PKI-related container in AD, that certificate or CRL is unavailable in any domain (and domain controller) within the current forest.

Publish a Certificate to Active Directory Containers

The following examples demonstrate the use of certificate management cmdlets in certain typical scenarios that involve the publication of certificates in Active Directory.

Example 1. Suppose your organization has implemented a PKI hierarchy with an offline root CA. Since the root CA has no network access, it cannot automatically publish its certificate to the Active Directory RootCA container. Certificates from that container are automatically downloaded by each forest member computer to the Trusted Root CAs (Root) store when the computer starts, the user logs on, or group policy is refreshed. The following example demonstrates how to publish your organization's root CA certificate to the AD RootCA container:

```
PS C:\> $RootCertificate = Import-QADCertificate C:\Certs\root.cer
PS C:\> Publish-QADCertificate -Certificate $RootCertificate -Container RootCA |
Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
sysadmins-LV-CA	sysadmins-LV-CA	2009.08.06.	2014.08.06.

The first command in this example imports the root CA certificate from a file to an internal certificate object. The second command publishes that certificate to the RootCA container.



For the purposes of the certificate chaining engine, every certificate published to any AD container is also added to the SubCA (AIA) container.

You should carefully choose which root CA certificates to trust. Any current forest user and computer will trust any certificate issued by the root or root subordinate CAs (as long as the particular certificate is not explicitly revoked). If you are not sure whether you can trust a particular root CA, do not publish its certificate to the RootCA AD container. Instead you should consider implementing qualified subordination for that root CA. For details on qualified subordination, see Microsoft's article at [http://technet.microsoft.com/library/cc739804\(WS.10\).aspx](http://technet.microsoft.com/library/cc739804(WS.10).aspx).

Example 2. Suppose your organization has implemented an offline subordinate CA. Since the offline subordinate CA has no network access, it cannot automatically publish its certificate to Active Directory. You can use the Publish-QADCertificate cmdlet to publish the CA's certificate to the SubCA (AIA) container by hand.

Example 3. Suppose your organization has implemented qualified subordination with a partner organization's PKI. To simplify certificate chain building, you may elect to publish the partner organization CA's certificates to the SubCA (AIA) container. This can be easily accomplished by using the Publish-QADCertificate cmdlet.

Example 4. Suppose your organization's CA has renewed its own CA certificate. To distribute the new certificate to clients, you need to publish the new certificate to the SubCA (AIA) container:

```
PS C:\> $SubCAertificate = Import-QADCertificate C:\Certs\subca.cer
PS C:\> Publish-QADCertificate -Certificate $SubCAertificate -Container SubCA |
Format-Table -AutoSize
```

IssuedBy -----	IssuedTo -----	ValidFrom -----	ValidTo -----
Contoso CA	contoso-DC2-CA	2009.03.30.	2011.03.30.

Certificates from the SubCA container are automatically downloaded by each current forest computer to the Intermediate CAs (CA) store. Those certificates will be considered as trusted, however, only if their chain root certificates are installed in the Trusted Root CAs (Root) container.

Example 5. Suppose your organization uses a third-party or other external CA to issue user logon certificates. To allow logon certificate validation by domain controllers, you must publish the issuing CA's certificate to the NTAuthCertificates (NTAuthCA) container:

```
PS C:\> $LogonCACert = Import-QADCertificate C:\Certs\subca.cer
PS C:\> Publish-QADCertificate -Certificate $LogonCACert -Container NTAuthCA |
Format-Table -AutoSize
```

IssuedBy -----	IssuedTo -----	ValidFrom -----	ValidTo -----
Contoso CA	contoso-DC2-CA	2009.03.30.	2011.03.30.

Certificates from the NTAuthCA container are automatically downloaded by each current forest computer to the Intermediate CAs (CA) store. Those certificates will be considered as trusted, however, only if their chain root certificates are installed in the Trusted Root CAs (Root) container.



There is an additional requirement for logon certificate validation by domain controllers. The authenticating domain controller must have a certificate based on either the Domain Controller, Domain Controller Authentication, or Kerberos Authentication template.

If you need to publish the same CA certificate to multiple AD containers, you can specify multiple containers at a time. For example:

```
PS C:\> Publish-QADCertificate -Certificate $CACert -Container RootCA, NTAuthCA, SubCA
```

Example 6. Suppose your organization has implemented qualified subordination with a partner organization's PKI. You need to publish the cross-certificate to the AD container and specify that this is a cross-certificate. The following example demonstrates how to publish a cross-certificate to the SubCA (AIA) container:

```
PS C:\> $CrossCert = Import-QADCertificate C:\Certs\cross.cer
PS C:\> Publish-QADCertificate -Certificate $CrossCert -CrossCertificate -Container
SubCA | Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
contoso-DC2-CA	sysadmins-LV-CA	2009.11.24.	2011.03.30.



You should publish cross-certificates to the SubCA (AIA) container only.

To review the contents of AD containers, you can use the Get-QADPKIObject cmdlet as follows:

```
PS C:\> Get-QADPKIObject RootCA, NTAAuthCA | Get-QADCertificate
```

The output of this command lists all certificates stored in the Certification Authorities (RootCA) and NTAAuthCertificates (NTAuthCA) AD containers. You may specify multiple AD containers at a time. If you want to examine only cross-certificates, you can use the following command:

```
PS C:\> Get-QADPKIObject SubCA | Get-QADCertificate | Where-Object {$_.CrossCertificate}
| Format-Table -AutoSize
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
contoso-DC2-CA	sysadmins-LV-CA	2009.11.24.	2011.03.30.

Each certificate object has the CrossCertificate property. If this property is set to True, the certificate is a cross-certificate.

Remove a Certificate from Active Directory Containers

When you decommission your organization's CA or your organization ends a relationship with a partner's PKI, all unused certificates should be removed from the Active Directory containers. For example, if you decommission a CA named Adatum Class 1 Root Certification Authority, you can use the following command to remove all related certificates from AD:

```
PS C:\> Get-QADPKIObject RootCA, SubCA, NTAAuthCA | Get-QADCertificate -IssuedTo "adatum*"
| Unpublish-QADCertificate RootCA, SubCA, NTAAuthCA -Force
```

IssuedBy	IssuedTo	ValidFrom	ValidTo
-----	-----	-----	-----
Adatum Class 1 Root Cert...	Adatum Class 1 Root Cert...	2010.03.21.	2020.03.21.

The command in this example retrieves all certificates from the RootCA, SubCA and NTAAuthCA containers, selects the certificates whose IssuedTo property begins with 'Adatum', and then removes all the selected certificates from the specified containers.

Technically, CA certificates are stored in a special entry type certificationAuthority. If a particular CA has multiple certificates, they are stored in a single entry as a certificate array. If you remove certificates using Unpublish-QADCertificate, only certificates are removed. In order to remove the empty certificationAuthority entry, you need to supply the -Force parameter. The -Force parameter does not remove the certificationAuthority entry if the entry contains any certificates.

In order to remove cross-certificates only, you can use the following command:

```
PS C:\> Get-QADPKIOBJECT SubCA | Get-QADCertificate | Where-Object {$_.CrossCertificate}
| Unpublish-QADCertificate SubCA -Force
```

For details on how to decommission a Windows enterprise certification authority, see Microsoft's article at <http://support.microsoft.com/kb/889250>.

Publish CRLs to Active Directory

This section covers publication of CRLs to Active Directory. While there are several containers for CA certificates, only one container is used for CRLs: **CDP**. You should not to publish CRLs to other containers because the CRLs published to a container other than **CDP** are unusable.

Unlike the contents of certificate containers, the contents of the **CDP** container is not automatically downloaded by the current forest members. This is the most common mistake administrators make when they publish CRLs to Active Directory and believe that forest members will download the published CRLs. You may only need to publish a CRL to the **CDP** container if a certificate's CDP extension refers to that container. Thus, if your CA issues certificates and places only HTTP URLs to the certificate CDP extension, you do not need to publish a CRL to Active Directory because no one will download that CRL from Active Directory.

Also, as the best practice for revocation checking planning, it is strongly recommended not to include LDAP URLs in the certificate CDP and AIA extensions. Instead it is recommended to use a highly available Web server that will serve the CA certificates and CRLs. This would, for instance, enable internet users to validate your organization's SSL certificates when they access your Web site using the HTTPS protocol.

By default, a CA includes LDAP URLs in both certificate CDP and AIA extensions. Your organization may implement an offline root CA for internal use only: all certificates issued by this root and any subordinate CAs can be used within your organization only, with no external (internet) access. For internet users, your organization may acquire SSL certificates from third-party commercial CAs. In that case, you may allow your offline CA to add your LDAP URLs to the certificate CDP and AIA extensions. Since the CA is offline and has no network access, you will have to manually copy the certificates and CRLs to a removable drive, copy them to any domain-joined computer, and then manually publish those files to Active Directory.

In order to publish a CRL to Active Directory, you need to supply the `CAName` parameter. This is because each CA server creates a subcontainer under the CDP container with the CA computer short (NetBIOS) name. The following example demonstrates how to publishes CRLs issued by 'Contoso-DC2-CA' with the CA computer name of DC2:

```
PS C:\> $CRL = Import-QADCertificateRevocationList C:\CRLs\contoso-DC2-CA.crl
PS C:\> Publish-QADCertificateRevocationList -CRL $CRL -CAName DC2 | Format-Table
-AutoSize
```

Type	Number	BaseNumber	KeyIndex	EffectiveDate	NextUpdate	Entries	Issuer
Base	380		0	2010.03.20.	2010.03.28.	18	CN=contos...

Since the CRLs are published to the **CDP** container only, you do not need to specify a container where to publish CRLs.

To review all published CRLs, you can use the following command:

```
PS C:\> Get-QADPKIObject CDP | Get-QADCertificateRevocationList | Format-Table -AutoSize
```

Type	Number	BaseNumber	KeyIndex	EffectiveDate	NextUpdate	Entries	Issuer
Base	369		0	2010.03.06.	2010.06.07.	14	CN=Contos...
Delta	362		0	2010.03.06.	2010.06.07.	0	CN=Contos...
Base	380		0	2010.03.20.	2010.03.28.	18	CN=contos...

Note that you need to specify the container name for the `Get-QADPKIObject` cmdlet. This is because the `QADPKIObject` cmdlet is used to retrieve both certificates and CRLs from various Active Directory containers.

Remove CRLs from Active Directory

Normally you do not need to remove CRLs from Active Directory because they are not accumulated; rather, they are replaced with the newest CRLs. That is, when you publish the most recent CRL (base or delta) issued by a certain CA, the previous appropriate CRL is replaced. The only reason that you might want to remove a CRL is a CA decommissioning procedure. The following example demonstrates how to remove all CRLs issued by the 'Contoso-DC2-CA' CA server with the computer name of DC2:

```
PS C:\> Get-QADPKIObject CDP | Get-QADCertificateRevocationList | Where-Object
{$_.Issuer -like "CN=contoso-dc2-ca*"} | Unpublish-QADCertificateRevocationList -CAName
DC2 -Force
```

Type	Number	BaseNumber	KeyIndex	EffectiveDate	NextUpdate	Entries	Issuer
Base	380		0	2010.03.20.	2010.03.28.	18	CN=contos...

As in the `Publish-QADCertificateRevocationList` example, you need to specify the CA computer name for CRL removal from Active Directory. It is advisable to supply the `-Force` parameter because this will cause the unpublish operation to delete the following:

- CRL entries that contain neither base nor delta CRLs.
- The corresponding CA subcontainer when it is empty.