

# Managing Active Directory Users and Computers with Powershell

**Adam Bertram**

PowerShell MVP

**AVAILABILITY™**  
for the Modern Data Center

## Introduction

An organization's Active Directory (AD) environment often grows and becomes very large to include thousands of user, computer and group objects. Although AD provides the familiar ADUC (Active Directory Users and Computers) or Active Directory Administrative Center (ADAC) tools to manage these thousands of object types, the process can be cumbersome. When large additions or changes need to be made to AD, ADUC tends to be inconvenient, at best, and downright impossible, at worst. Windows PowerShell, however, can save the day with its great automation capabilities.

Microsoft has released a PowerShell module for managing AD, which lets an admin manage AD from a PowerShell command-line interface. By moving the functionality to the command line, an admin doesn't have to fool with the GUI (Graphical User Interface) anymore. Instead, this allows the admin to write automation around AD. This is a very powerful tool that can save organizations with large AD environments countless man hours managing various AD objects.

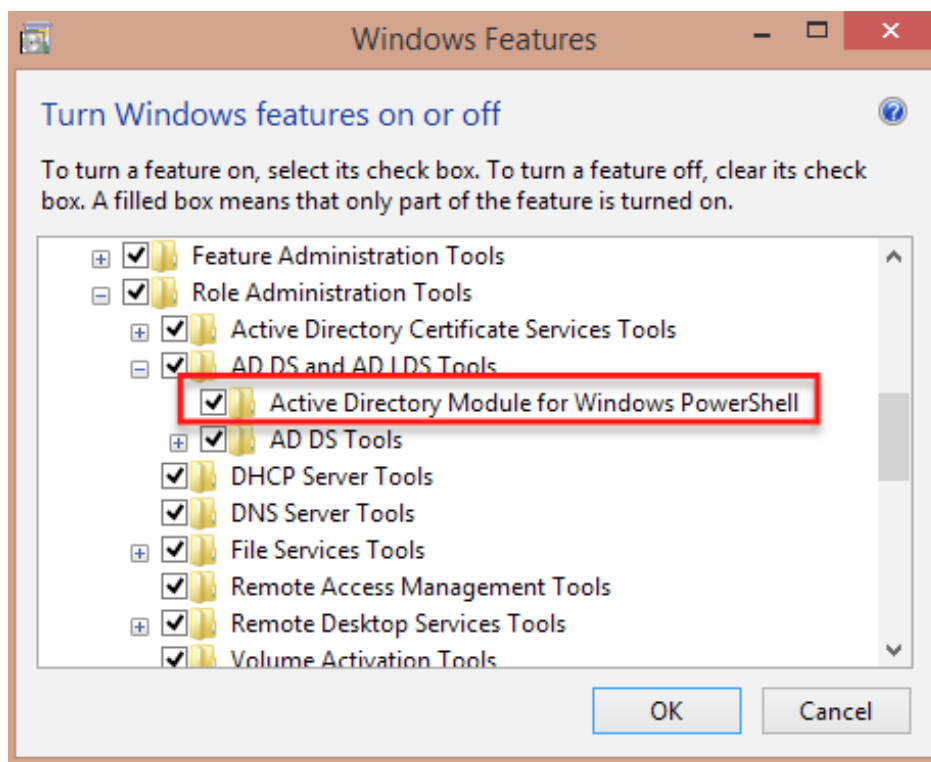
In this white paper, you'll be introduced to managing AD using PowerShell. You'll learn how to:

- Perform the initial setup
- Manage common AD objects like users, computers and groups
- Restore these objects if they're removed

## Getting started

To get started, you'll need the AD module installed, regardless of whether it is running locally on a domain controller or remotely on a domain-joined client. This module is part of a Microsoft software package called RSAT (Remote Server Administration Tools), which is meant to be installed on client-operating systems and used to connect remotely to servers. RSAT is available on [Windows 7 SP1](#), [Windows 8.1](#) and Windows 10.

Once installed, the Windows PowerShell feature Active Directory Module must be enabled in the **Control Panel** and the **Programs** and **Features** applet.



Once this feature has been turned on, you should see the **Active Directory Module for Windows PowerShell** module in your PowerShell console. You can verify this by using the **Get-Module** command.

### Get-Module -ListAvailable -Name ActiveDirectory

If the module is available, you should see an output similar to this:

```
PS C:\> Get-Module -ListAvailable -Name ActiveDirectory

Directory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version      Name
-----
Manifest    1.0.0.0      ActiveDirectory
ExportedC    {Add-ADCe
```

Once the module is installed and available, you will be able to begin managing AD with PowerShell!

## Managing users

Users are the most common AD object type that you'll manage with PowerShell, which will allow you to perform just about every function possible in the ADUC GUI.

### Finding user accounts

AD accounts are found by using the **Get-AdUser** cmdlet. This powerful cmdlet provides a much more robust method of finding user accounts than the ADUC GUI, and it only requires a single parameter: **-Filter**. This parameter limits the results returned by **Get-AdUser**.

At its most basic use, Get-AdUser can be executed with this -Filter parameter followed by an asterisk:

#### Get-AdUser -Filter\*

This would return all user accounts in the entire Active Directory environment. You're probably not going to do this too often though. Typically, you're either looking for a single user account or all user accounts that match a specific set of criteria.

With **Get-AdUser** you can find user accounts in several different ways. The easiest way is to simply search for a single user using the user's **samAccountName**. Here's an example of finding the user account for **jmurphy**. By default, you'll see that Get-AdUser retrieves:

- Which OU that the account is in
- If the account is enabled or not
- The first name and last name
- Other useful information

```
PS C:\> Get-ADUser -Identity jmurphy

DistinguishedName : CN=JMurphy,OU=SomeOtherOU,DC=lab,DC=local
Enabled           : True
GivenName         : Joe
Name              : JMurphy
ObjectClass       : user
ObjectGUID        : d08277d7-a090-41a0-ba7d-525cb9f173c8
SamAccountName    : JMurphy
SID               : S-1-5-21-3353910224-3636413190-1920695484-1720
Surname           : Murphy
UserPrincipalName : JMurphy
```

Feel free to explore **Get-AdUser** further by using the **Get-Help** command:

#### Get-Help Get-AdUser -Full

This command will show you what Get-AdUser is capable performing. If you'd like more flexibility, many AD module cmdlets also have the ability to query multiple objects through the **Filter** parameter.

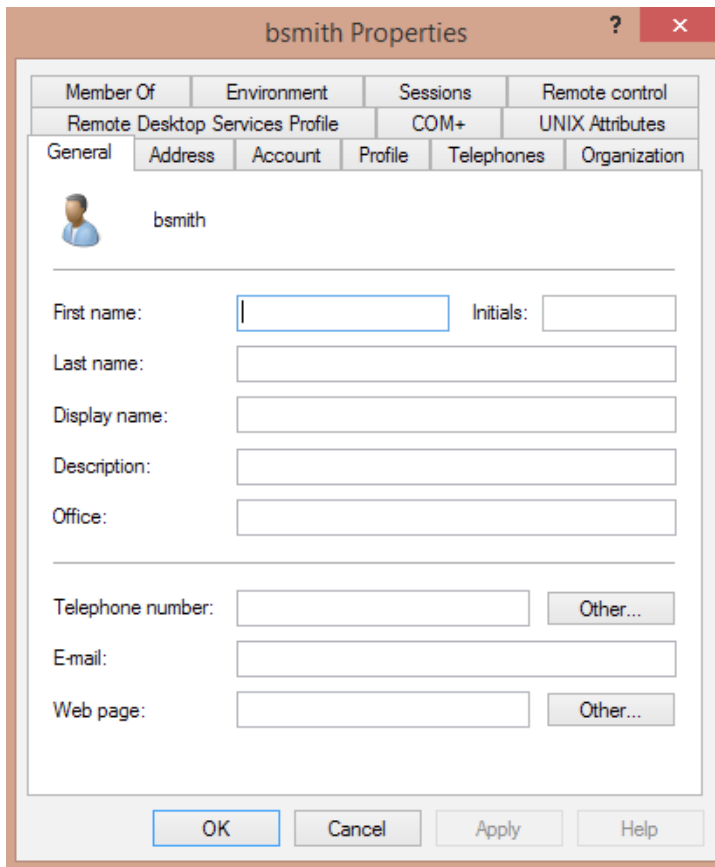
### Creating AD user accounts



AD user accounts are created with the **New-AdUser** cmdlet, which accepts many common user attributes, but doesn't actually require a large majority of the attributes taken for granted with the GUI. For example, here's a quick way to create a single AD user with a **samAccountName** of **bsmith**:

### New-ADUser -Name bsmith

Although this process is somewhat simple, what about all the other attributes such as first name, last name, what OU it has been placed in, password and so on?



The screenshot shows the 'bsmith Properties' dialog box with the 'General' tab selected. The user's name is 'bsmith'. The following fields are visible and mostly empty:

- First name:
- Initials:
- Last name:
- Display name:
- Description:
- Office:
- Telephone number:  Other...
- E-mail:
- Web page:  Other...

At the bottom, there are buttons for 'OK', 'Cancel', 'Apply', and 'Help'. The 'OK' button is highlighted.

As you can see, this account is not very functional because it was created without all of the common attributes.

Creating useful user accounts requires a little more effort. Once you figure out how it works, however, it can easily be scripted. Nearly every user attribute has an associated parameter to **New-AdUser**. For any attributes that **New-AdUser** does not directly support, you'll always have the **Add** parameter of **Set-AdUser**. Here's an example of creating that same **bsmith** user account again, except this time you'll see that a lot more attributes are used. This is a real-world example:

```
$NewUserParams = @{
    'UserPrincipalName' = 'bsmith'
    'Name' = 'bsmith'
    'GivenName' = 'Bob'
    'Surname' = 'Smith'
    'Title' = 'Accounting Manager'
    'SamAccountName' = 'bsmith'
    'AccountPassword' = (ConvertTo-SecureString 'supersecret' -AsPlainText -Force)
    'Enabled' = $true
    'Initials' = 'D'
}
```

#### **New-AdUser @NewUserParams**

In the example above, I'm using a technique in PowerShell called **splatting**, which is a way of providing cmdlets, like **New-AdUser**, a lot of parameter in a clean, concise way. You don't have to use this technique. You can pass these parameters to **New-AdUser** with a single line like below, but it would span 250 characters across the screen. Splatting is a great way to prevent this:

```
New-AdUser -UserPrincipalName 'bsmith' -Name 'bsmith' -GivenName 'Bob' -Surname
'Smith' -Title 'Accounting Manager' -SamAccountName 'bsmith' -AccountPassword (ConvertTo-
SecureString 'supersecret' -AsPlainText -Force) -Enabled $true -Initials 'D'
```

## Removing AD user accounts

Every user account has a useful lifespan, yet sometimes that lifespan ends and the account will need to be removed. Luckily for us, removing a user account is just about as easy as finding a user account. Using the PowerShell pipeline, we can simply find the user account and then pass that result to

**Remove-AdUser** and we're done.

```
PS C:\> Get-ADUser miltho

DistinguishedName : CN=MilTho,CN=Users,DC=lab,DC=local
Enabled           : True
GivenName        : Miles
Name             : MilTho
ObjectClass      : user
ObjectGUID       : fcdd3f8d-020c-4004-bcf9-689badc7eebf
SamAccountName   : MilTho
SID              : S-1-5-21-3353910224-3636413190-1920695484-1677
Surname          : Thompson
UserPrincipalName : MilTho

PS C:\> Get-ADUser miltho | Remove-AdUser

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove" on target "CN=MilTho,CN=Users,DC=lab,DC=local".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
PS C:\> Get-ADUser miltho
Get-ADUser : Cannot find an object with identity: 'miltho' under: 'DC=lab,DC=local'.
At line:1 char:1
+ Get-ADUser miltho
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (miltho:ADUser) [Get-ADUser], ADIdentityNotF
+ FullyQualifiedErrorId : ActiveDirectoryCmdlet:Microsoft.ActiveDirectory.Management.A
```

In the above example, I have a user named miltho. To remove this user, I simply send the results of Get-AdUser to Remove-AdUser, confirm I really want to remove the user account and then it's removed.

## Managing computers

Computers are managed similarly to users. Rather than using cmdlet names like **Get-AdUser** and **New-AdUser**, computers surprisingly use cmdlet names like **Get-AdComputer** and **New-AdComputer**. These cmdlets' parameters are also very similar because, a user object and computer object in Active Directory technically aren't that different other than the attributes that create them.

Let's go over a few examples.

### Finding computer accounts

To find computer accounts you'll probably use the **Get-AdComputer** cmdlet. This cmdlet is similar to **Get-AdUser** in that it also has a **Filter** parameter and behaves exactly the same way.

Using the asterisk, we can now find all *computers* in AD, thus:

#### **Get-AdComputer -Filter\***

Let's narrow down this search a little and find only computer accounts that are not enabled:

```
PS C:\> Get-AdComputer -Filter "Enabled -eq 'False'"

DistinguishedName : CN=JOECOMPUTER,OU=Corporate Computers,DC=lab,DC=local
DNSHostName       :
Enabled           : False
Name              : JOECOMPUTER
ObjectClass       : computer
ObjectGUID        : 38d92c23-1266-4c84-b3f8-29ca1b28a275
SamAccountName    : JOECOMPUTER$
SID               : S-1-5-21-3353910224-3636413190-1920695484-1721
UserPrincipalName :
```

You'll see that by using the **Filter** parameter, I can simply perform a conditional statement on the **Enabled** attribute and find all computer accounts that have the **Enabled** attribute set to **False**.

## Modifying existing computer accounts

In addition to finding and creating user and computer accounts with PowerShell, you can also modify existing accounts using **Set** cmdlets. A common verb for PowerShell cmdlets for modifying objects is **Set**. PowerShell cmdlets are broken down in a Verb-Noun format. You'll notice that every properly named cmdlet has some verb, a dash and then a noun. Although not required, it's considered a good practice to name your custom functions in this manner. In this example, we can use **Set-AdComputer** for modifying existing AD computer accounts.

Using the **Set-AdComputer** cmdlet is simple. This cmdlet, similar to the **New** cmdlets, has many common attributes to the AD object as a parameter. For example, if a computer is moving locations you'd use the **-Location** parameter.

```
PS C:\> (Get-ADComputer JOECOMPUTER -Properties Location).Location
Building 123
PS C:\> Set-ADComputer -Identity 'JOECOMPUTER' -Location 'Building 900'
PS C:\> (Get-ADComputer JOECOMPUTER -Properties Location).Location
Building 900
```

**NOTE:** In the above example, I'm using the **-Properties** parameter. Even though the **Get** cmdlets output *common* attributes, it doesn't mean they output *all* of the object attributes. By utilizing the **-Properties** parameter, you will be able to output any of those non-default parameters. In this example, the AD attribute **Location** is not a default output attribute, so it had to be manually specified using the **-Properties** parameter.

Similarly, you can accomplish the same thing by using the **PowerShell pipeline** to pass the output of **Get-AdComputer** directly into **Set-AdComputer**:

```
PS C:\> (Get-ADComputer JOECOMPUTER -Properties Location).Location
Building 900
PS C:\> Get-ADComputer JOECOMPUTER | Set-AdComputer -Location 'Building 500'
PS C:\> (Get-ADComputer JOECOMPUTER -Properties Location).Location
Building 500
```

## Removing computer accounts

One of the great PowerShell attributes is that cmdlet behavior is nearly identical across similar cmdlets. The Active Directory module's cmdlets are no exception. By behaving identically to removing user accounts, an administrator can also remove computer accounts the exact same way. Although this time we simply replaced the word **User** with **Computer**.

```

PS C:\> Get-ADComputer JOECOMPUTER

DistinguishedName : CN=JOECOMPUTER,OU=Corporate Computers,DC=lab,DC=local
DNSHostName       :
Enabled           : False
Name              : JOECOMPUTER
ObjectClass       : computer
ObjectGUID        : 38d92c23-1266-4c84-b3f8-29ca1b28a275
SamAccountName    : JOECOMPUTER$
SID               : S-1-5-21-3353910224-3636413190-1920695484-1721
UserPrincipalName :

PS C:\> Get-ADComputer JOECOMPUTER | Remove-ADComputer

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove" on target "CN=JOECOMPUTER,OU=Corporate Computers,DC=lab,DC=local".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y
PS C:\> Get-ADComputer JOECOMPUTER
Get-ADComputer : Cannot find an object with identity: 'JOECOMPUTER' under: 'DC=lab,DC=local'.
At line:1 char:1
+ Get-ADComputer JOECOMPUTER
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (JOECOMPUTER:ADComputer) [Get-ADComputer], ADIdentityNo
+ FullyQualifiedErrorId : ActiveDirectoryCmdlet:Microsoft.ActiveDirectory.Management.ADIdentityNo

```

## Managing groups

Groups are another kind of AD object supported by the PowerShell AD module. Just like users and computers, you can create, modify and remove groups. Groups contain user and computer objects, as well as other groups.

As you might expect, the group cmdlets use the same naming scheme and have the same **Filter** parameter as users and computer objects. Let's take the **Filter** parameter to the next level. This time, instead of narrowing the results of a **Get** cmdlet to just one attribute, let's use two attributes.

```

PS C:\> Get-ADGroup -Filter "GroupScope -eq 'Global' -and Name -like 'Gig*'"

DistinguishedName : CN=Gigantic Corporation Inter-Intra Synergy Group,OU=Corporate Users,DC=lab,DC=local
GroupCategory     : Security
GroupScope        : Global
Name              : Gigantic Corporation Inter-Intra Synergy Group
ObjectClass       : group
ObjectGUID        : 1fb2c3a8-d0a5-4bb7-9b6e-c9cc71b3d3f5
SamAccountName    : Gigantic Corporation Inter-Intra Synergy Group
SID               : S-1-5-21-3353910224-3636413190-1920695484-1670

```

You'll see that the **Filter** parameter accepts multiple attributes by combining them with the **-and** operator. In this example, I am finding all global groups that start with the letters **Gig**.

The AD module also provides the **GroupMember** cmdlets: **Get-AdGroupMember**, **Add-AdGroupMember** and **Remove-AdGroupMember**. These cmdlets allow you to manipulate group memberships for all types of AD groups.

For example, if you'd like to find all members of the **Domain Users** group, then you'd use **Get-AdGroupMember**:

**Get-ADGroupMember -Identity 'Domain Users'**

This works great, but isn't that impressive. What about adding users matching a certain criteria to a specific group?

```
PS C:\> Get-ADGroupMember 'My Group'
PS C:\> Add-ADGroupMember -Identity 'My Group' -Members (Get-ADUser -Filter "Enabled -eq 'True'")
PS C:\> (Get-ADGroupMember 'My Group').Count
13
```

In this example, I have added all AD user accounts that are enabled to the **My Group** AD group. The same can be done for **Remove-AdGroupMember**, which will turn around and remove all of the user accounts that were just added to **My Group**.

```
PS C:\> (Get-ADGroupMember 'My Group').Count
13
PS C:\> Remove-ADGroupMember -Identity 'My Group' -Members (Get-ADUser -Filter "Enabled -eq 'True'") -Confirm:$false
PS C:\> (Get-ADGroupMember 'My Group').Count
0
```

## Recovering deleted objects

You should now have all of the user accounts, computers objects and groups created exactly the way you want them. What if someone then comes along and deletes a critical object from AD? After you get over the initial shock, it's time to figure out how to get the object back. There are a few ways to recover deleted objects from AD that either require using the command line or the GUI. Here, I will discuss how to recover deleted objects using Windows PowerShell, but will touch upon a free AD recovery tool from Veeam®, called Veeam Explorer™ for Microsoft Active Directory.

### Using Windows PowerShell

Introduced with Windows Server 2008 R2, the AD Recycle Bin is a feature that, instead of completely removing, simply tags an object as deleted and offers a time when it can be restored.

If you'd like to use the recycle bin and PowerShell you'll first need to ensure it's enabled. Unfortunately, this is not enabled by default. To enable this feature, you'll need to use the **Enable-AdOptionalFeature** cmdlet.

```
PS C:\> Enable-AdOptionalFeature -Identity 'Recycle Bin Feature' -Scope ForestOrConfigurationSet -Target 'lab.local' -Server labdc.lab.local
WARNING: Enabling 'Recycle Bin Feature' on 'CN=Partitions,CN=Configuration,DC=lab,DC=local' is an irreversible action! You will not be able to disable 'Recycle Bin Feature' on 'CN=Partitions,CN=Configuration,DC=lab,DC=local' if you proceed.
Confirm
Are you sure you want to perform this action?
Performing the operation "Enable" on target "Recycle Bin Feature".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): a
```

Once the feature is enabled, Active Directory will then begin placing objects in the AD Recycle Bin, rather than simply deleting them.

Once the Recycle Bin is enabled, it's just a matter of running a restore on an object.

Once the AD object has been deleted, it can only be found using the **Get-AdObject** cmdlet and the **-IncludeDeletedObjects** parameter.

```
PS C:\> Get-AdObject -Filter * -IncludeDeletedObjects | Where-Object {$_.Deleted -and ($_.Name -like '*JoeDav*')}

Deleted           : True
DistinguishedName : CN=JoeDav\0ADEL:759c9d17-9992-460b-b482-df5ae6f718da,CN=Deleted Objects,DC=lab,DC=local
Name              : JoeDav
                  : DEL:759c9d17-9992-460b-b482-df5ae6f718da
ObjectClass       : user
ObjectGUID        : 759c9d17-9992-460b-b482-df5ae6f718da
```

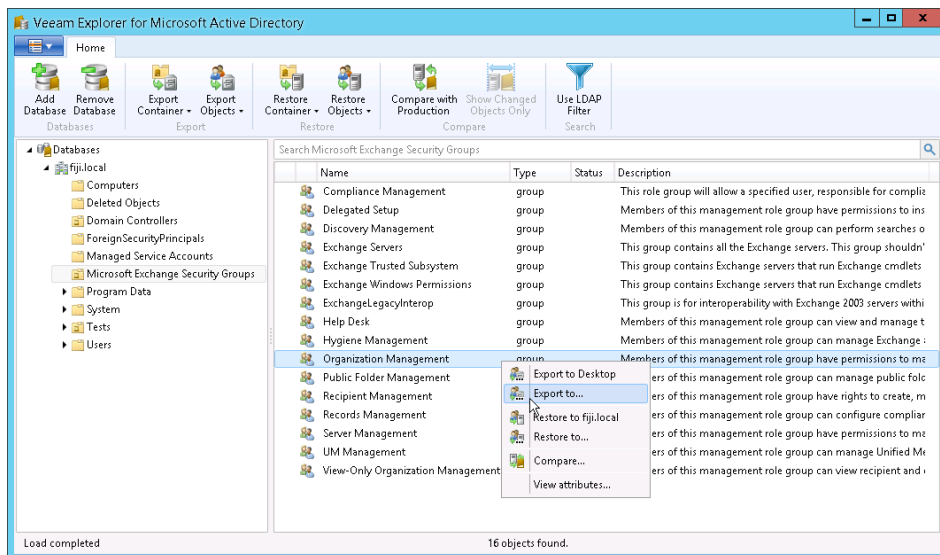
In this example, I removed a user called **JoeDav**, and by using the **Get-AdObject** cmdlet with the **-IncludeDeletedObjects** parameter, I found it. You'll also notice it was necessary to use **Where-Object** here. I'm used this cmdlet only to find objects that had been deleted. By default, **Get-AdObject** will find all objects that still exist *and* are deleted. Here, I just wanted to find all objects that had been deleted.

Now that the deleted object has been found, I just need to pipe it to the **Restore-AdObject** and it will be restored exactly how it was before it was removed.

```
PS C:\> Get-AdObject -Filter * -IncludeDeletedObjects | Where-Object {$_.Deleted -and ($_.Name -like '*JoeDav*')} | Restore-AdObject
-AdObject
PS C:\> Get-ADUser JoeDav

DistinguishedName : CN=JoeDav,CN=Users,DC=lab,DC=local
Enabled           : True
GivenName         : Joe
Name              : JoeDav
ObjectClass       : user
ObjectGUID        : 759c9d17-9992-460b-b482-df5ae6f718da
SamAccountName    : JoeDav
SID               : S-1-5-21-3353910224-3636413190-1920695484-1676
Surname           : Davis
UserPrincipalName : JoeDav
```

## Using Veeam Explorer for Microsoft Active Directory



Windows PowerShell is very handy, yet it can also be cumbersome at times. In addition, some users prefer to handle tasks like restoring critical AD objects via the GUI. This is a great opportunity to introduce **Veeam Explorer for Microsoft Active Directory**, which takes restoring AD objects to a whole new level. With this tool, you not only have the ability to instantly restore individual objects, but you can also restore entire OUs (organizational units), user accounts and even a user or Active Directory password, both individually or in a bulk.



Veeam Explorer *for Microsoft Active Directory* does not require the **AD Recycle Bin** to be enabled. This means if you've lost one or more objects and didn't enable the recycle bin, the objects can still be recovered. This is because Veeam directly mounts a raw AD database.

With Veeam Explorer *for Microsoft Active Directory* there is no need to restore a VM in a virtual lab. Instead, an administrator can open the database and explore all the items in the backup by using an intuitive point-and-click interface. An advanced search option makes the whole process of finding and restoring many AD object types very easy and fast, even for new users.

Once a required object has been found, an administrator can either export Active Directory objects and user attributes data from a backup into an **LDIFDE** format or complete an Active Directory object restore directly back into the production Active Directory database.

Veeam Explorer *for Active Directory* was introduced with the Veeam Availability Suite™ v8. To learn more about its restore possibilities, check out the following free resources:

[Active Directory backup and recovery with Veeam](#)

[Active Directory basics: Under the hood of Active Directory](#)

## Summary

Using Windows PowerShell to manage AD objects is an easy way to create, modify and remove common AD objects. If you're new to PowerShell, it may seem a little confusing at first, but the more you use it, the quicker you'll notice how much time you can save. Plus, PowerShell is not just a command prompt replacement as you have seen here, but also a full-featured scripting language that's capable of automating much larger AD tasks.

PowerShell allows you to, in a sense, customize the way you manage AD, rather than being forced to do things the GUI way. Once you're able to grasp the core concepts, you'll be well on your way to developing robust and efficient AD tools written in PowerShell.

## About the Author



**Adam Bertram** is an independent consultant, technical writer, trainer and presenter. Adam specializes in consulting and evangelizing all things IT automation mainly focused around Windows PowerShell. Adam is a Microsoft Windows PowerShell MVP, 2015 powershell.org PowerShell hero and has numerous Microsoft IT pro certifications. He is a writer, trainer and presenter and authors IT pro course content for Pluralsight. He is also a regular contributor to numerous print and online publications and presents at various user groups and conferences. You can find Adam at [adamtheautomator.com](http://adamtheautomator.com) or on Twitter at [@adbertram](https://twitter.com/adbertram).

## About Veeam Software

**Veeam**<sup>®</sup> recognizes the new challenges companies across the globe face in enabling the Always-On Business<sup>™</sup>, a business that must operate 24/7/365. To address this, Veeam has pioneered a new market of *Availability for the Modern Data Center*<sup>™</sup> by helping organizations meet recovery time and point objectives (RTPO<sup>™</sup>) of less than 15 minutes for all applications and data, through a fundamentally new kind of solution that delivers high-speed recovery, data loss avoidance, verified protection, leveraged data and complete visibility. **Veeam Availability Suite**<sup>™</sup>, which includes **Veeam Backup & Replication**<sup>™</sup>, leverages virtualization, storage, and cloud technologies that enable the modern data center to help organizations save time, mitigate risks, and dramatically reduce capital and operational costs.

Founded in 2006, Veeam currently has 30,500 ProPartners and more than 145,500 customers worldwide. Veeam's global headquarters are located in Baar, Switzerland, and the company has offices throughout the world. To learn more, visit <http://www.veeam.com>.

**COMING SOON**

# **NEW Veeam® Availability Suite™ v9**

RTPO™ <15 minutes for ALL applications and data  
Enabling the Always-On Business™  
with *Availability for the Modern Data Center™*

To learn more, visit [www.veeam.com](http://www.veeam.com)