3

Managing Recipients

In this chapter, we will cover the following:

* Creating, modifying, and removing mailboxes
* Working with contacts
* Managing distribution groups
* Managing resource mailboxes
* Creating recipients in bulk using a CSV file
* Working with recipient filters
* Adding and removing recipient e-mail addresses
* Hiding recipients from address lists
* Configuring recipient moderation
* Configuring message delivery restrictions
* Managing automatic replies and out of office settings for a user
* Adding, modifying, and removing server-side inbox rules
* Managing mailbox folder permissions

Importing user photos into Active Directory

# Introduction

If you are like many other administrators, you probably spend the majority of your time performing recipient-related management tasks when dealing with Exchange. If you work in a large environment with thousands of recipients, to create, update, and delete recipients will probably be a cumbersome and time consuming process. Of course, the obvious solution to this is to use the Exchange Management Shell. Utilizing the Exchange Management Shell, you can automate all of your recipient management tasks and drastically speed up your work.

The concept of an Exchange recipient is more than just a user with a mailbox. An Exchange recipient is any Active Directory object that has been mail-enabled and can receive messages within the Exchange organization. This can be a distribution group, a contact, a mail-enabled public folder, and more. These object types include individual sets of cmdlets that can be used to completely automate the administration of the Exchange recipients in your environment.

The goal of this chapter is to show you some common solutions that can be used when performing day-to-day recipient management from the shell. Quite often, Exchange recipients are provisioned or updated in bulk through an automated process driven by a PowerShell script. The recipes in this chapter will provide solutions for these types of scripts that you can use right away. You can also use these concepts as a guide to build your own scripts from scratch to automate recipient related tasks in your environment.

## Performing some basic steps

To work with the code samples in this chapter, we'll need to launch the Exchange Management Shell using the following steps:

1. Log onto a workstation or server with the Exchange Management Tools installed.
2. You can connect using remote PowerShell if you for some reason don't have Exchange Management Tools installed. Use the following command:

$Session = New-PSSession -ConfigurationName Microsoft.Exchange `

-ConnectionUri http://tlex01/PowerShell/ `

-Authentication Kerberos

Import-PSSession $Session

1. Open the Exchange Management Shell by clicking the windows button and go to Microsoft Exchange Server 2016 | Exchange Management Shell.

If any additional steps are required they will be listed at the beginning of the recipe in the Getting ready section.

Remember to start the Exchange Management Shell using **Run as Admin** to avoid permission problems.   
  
In the chapter, notice that in the examples of cmdlets, I have used the back tick (`) character for breaking up long commands into multiple lines.  
The purpose with this is to make it easier to read. The back ticks are not required and should only be used if needed.

Notice that the Exchange variables, like $exscripts, are not available when using the method above.

# Creating, modifying, and removing mailboxes

One of the most common tasks performed within the Exchange Management Shell is mailbox management. In this recipe, we'll take a look at the command syntax required to create, update, and remove mailboxes from your Exchange organization. The concepts outlined in this recipe can be used to perform basic day-to-day tasks and will be useful for more advanced scenarios such as creating mailboxes in bulk.

## How to do it...

1. Let's start off by creating a mailbox-enabled Active Directory user account. To do this, we can use the New-Mailbox cmdlet as shown in the following example:

$password = ConvertTo-SecureString -AsPlainText P@ssw0rd `  
-Force

New-Mailbox -UserPrincipalName dave@testlabs.se `

-Alias dave `

-Database DB1 `

-Name ‘Dave Jones’ `

-OrganizationalUnit Sales `

-Password $password `

-FirstName Dave `

-LastName Jones `

-DisplayName 'Dave Jones'

1. Once the mailbox has been created we can modify it using the Set-Mailbox cmdlet:

Set-Mailbox -Identity dave `

-UseDatabaseQuotaDefaults $false `

-ProhibitSendReceiveQuota 5GB `

-IssueWarningQuota 4GB

To remove the Exchange attributes from the Active Directory user account and mark the mailbox in the database for removal, use the   
Disable-Mailbox cmdlet:

Disable-Mailbox -Identity dave -Confirm:$false

## How it works...

When running the New-Mailbox cmdlet, the -Password parameter is required and you need to provide a value for it using a secure string object. As you can see from the code, we've used the ConvertTo-SecureString cmdlet to create a $password variable that stores a specified value as an encrypted string. This $password variable is then assigned to the -Password parameter when running the cmdlet. There's no requirement to first store this object in a variable; we could have done it inline, as shown next:

New-Mailbox -UserPrincipalName dave@testlabs.se `

-Alias dave `

-Database DB1 `

-Name ‘Dave Jones’ `

-OrganizationalUnit Sales `

-Password (ConvertTo-SecureString -AsPlainText P@ssw0rd -Force) `

-FirstName Dave `

-LastName Jones `

-DisplayName 'Dave Jones'

Keep in mind that the password used here needs to comply with your Active Directory password policies, which may enforce a minimum password length and have requirements for complexity.

Only a few parameters are actually required when running New-Mailbox, but the cmdlet itself supports several useful parameters that can be used to set certain properties when creating the mailbox. You can run Get-Help New-Mailbox -Detailed to determine which additional parameters are supported.

The New-Mailbox cmdlet creates a new Active Directory user and then mailbox-enables that account. We can also create mailboxes for existing users with the Enable-Mailbox cmdlet, using syntax similar to the following:

Enable-Mailbox steve -Database DB1

The only requirement when running the Enable-Mailbox cmdlet is that you provide the identity of the Active Directory user that should be mailbox-enabled. In the previous example, we've specified the database in which the mailbox should be created, but this is optional. The Enable-Mailbox cmdlet supports a number of other parameters that you can use to control the initial settings for the mailbox.

You can use a simple one-liner to create mailboxes in bulk for existing Active Directory users:

Get-User -RecipientTypeDetails User |

Enable-Mailbox -Database DB1

Notice that we've run the Get-User cmdlet specifying User as the value for   
the -RecipientTypeDetails parameter. This will retrieve only the accounts in   
Active Directory that have not been mailbox-enabled. We then pipe those objects   
down to the Enable-Mailbox cmdlet and mailboxes are created for each of those   
users in one simple operation.

Once mailboxes have been created, they can be modified with the Set-Mailbox cmdlet. As you may recall from our original example, we used the Set-Mailbox cmdlet to configure custom storage quota settings after creating a mailbox for Dave Jones. Keep in mind that the Set-Mailbox cmdlet supports almost 200 parameters, so anything that can be done to modify a mailbox can be scripted.

Bulk modifications to mailboxes can be done easily by taking advantage of the pipeline and the Set-Mailbox cmdlet. Instead of configuring storage quotas on a single mailbox, we can do it for multiple users at once:

Get-Mailbox -OrganizationalUnit testlabs.se/sales |

Set-Mailbox -UseDatabaseQuotaDefaults $false `

-ProhibitSendReceiveQuota 5GB `

-IssueWarningQuota 4GB

Here we are simply retrieving every mailbox in the Sales OU using the Get-Mailbox cmdlet. The objects returned from that command are piped down to Set-Mailbox which modifies the quota settings for each mailbox in one shot.

The Disable-Mailbox cmdlet will strip the Exchange attributes from an Active Directory user and will disconnect the associated mailbox. By default, disconnected mailboxes are retained for 30 days. You can modify this setting on the database that holds the mailbox. In addition to this, you can also use the Remove-Mailbox cmdlet to delete both the Active Directory account and the mailbox at once:

Remove-Mailbox -Identity dave -Confirm:$false

After running this command, the mailbox will be purged once it exceeds the deleted mailbox retention setting on the database. One common mistake is when administrators use the Remove-Mailbox cmdlet when the Disable-Mailbox cmdlet should have been used. It's important to remember that the Remove-Mailbox cmdlet will delete the Active Directory user account and mailbox while the Disable-Mailbox cmdlet only removes the mailbox, but the Active Directory user account still remains.

## There's more...

When we ran the New-Mailbox cmdlet in the previous examples, we assigned a secure string object to the –Password parameter using the ConvertTo-SecureString cmdlet. This is a great technique to use when your scripts need complete automation, but you can also allow an operator to enter this information interactively. For example, you might build a script that prompts an operator for a password when creating one or more mailboxes. There are a couple of ways you can do this. First, you can use the Read-Host cmdlet to prompt the user running the script to enter a password:

$pass = Read-Host "Enter Password" -AsSecureString

Once a value has been entered into the shell, your script can assign the $pass variable to the -Password parameter of the New-Mailbox cmdlet.

Alternatively, you can supply a value for the -Password parameter using the   
Get-Credential cmdlet:

New-Mailbox -Name Dave -UserPrincipalName dave@contoso.com `

-Password (Get-Credential).password

You can see that the value we are assigning to the -Password parameter in this example is actually the password property of the object returned by the Get-Credential cmdlet. Executing this command will first launch a Windows authentication dialog box where the   
caller can enter a username and password. Once the credential object has been created,   
the New-Mailbox cmdlet will run. Even though a username and password must be   
entered into the authentication dialog box, only the password value will be used when   
the command executes.

A commonly used mailbox type is the shared mailboxes, which is pretty much what the name says, a mailbox shared between different individuals. The difference between this mailbox type and the regular user mailbox is that the shared mailbox is typical a disabled active directory object. Therefore no one can use this username to logon interactively with. These mailboxes are simply just created by using the following cmdlet and parameters:

New-Mailbox -Name “UC Project Mailbox” –Shared

The example above is showing how to create the shared mailbox, the difference is to add the parameter –Shared into the cmdlet.

### Setting active directory attributes

Some of the Active Directory attributes that you may want to set when creating a mailbox might not be available using the New-Mailbox cmdlet. Good examples of this are a user's city, state, company, and department attributes. In order to set these attributes, you'll need to call the Set-User cmdlet after the mailbox has been created:

Set-User –Identity dave –Office IT –City Seattle –State Washington

You can run Get-Help Set-User -Detailed to view all of the available parameters supported by this cmdlet.

Notice that the cmdlet Set-User is an Active Directory cmdlet and not an Exchange cmdlet.  
In the examples using the cmdlet New-Mailbox for creating new mailboxes, it is not required to use all these parameters from the example. The only required parameters are UserPrincipalName, Name and Password.

## See also

* Using the help system in Chapter 1, PowerShell Key Concepts
* Creating recipients in bulk using a CSV file

Managing distribution groups

Managing resource mailboxes

# Working with contacts

Once you've started managing mailboxes using the Exchange Management Shell, you'll probably notice that the concepts and command syntax used to manage contacts are very similar. The difference of course is that we need to use a different set of cmdlets. In addition, we also have two types of contacts to deal with in Exchange. We'll take a look at how you can manage both of them in this recipe.

## How to do it...

1. To create a mail-enabled contact, use the New-MailContact cmdlet:

New-MailContact -Alias rjones `

-Name "Rob Jones" `

-ExternalEmailAddress rob@fabrikam.com `

-OrganizationalUnit sales

1. Mail-enabled users can be created with the New-MailUser cmdlet:

New-MailUser -Name 'John Davis' `

-Alias jdavis `

-UserPrincipalName jdavis@contoso.com `

-FirstName John `

-LastName Davis `

-Password (ConvertTo-SecureString -AsPlainText P@ssw0rd `  
-Force) `

-ResetPasswordOnNextLogon $false `

-ExternalEmailAddress jdavis@fabrikam.com

## How it works...

Mail contacts are useful when you have external e-mail recipients that need to show up in your global address list. When you use the New-MailContact cmdlet, an Active Directory contact object is created and mail-enabled with the external e-mail address assigned. You can mail-enable an existing Active Directory contact using the   
Enable-MailContact cmdlet.

Mail users are similar to mail contacts in that they have an associated external e-mail address. The difference is that these objects are mail-enabled Active Directory users, and that explains why we needed to assign a password when creating the object. You might use a mail user for a contractor who works onsite in your organization and needs to be able to logon to your domain. When users in your organization need to e-mail this person, they can select them from the global address list and messages sent to these recipients will be delivered to the external address configured for the account.

Just as when dealing with mailboxes, there are a couple of considerations that should be taken when it comes to removing contacts and mail users. You can remove the Exchange attributes from a contact using the Disable-MailContact cmdlet.   
The Remove-MailContact cmdlet will remove the contact object from Active Directory and Exchange. Similarly, the Disable-MailUser and Remove-MailUser cmdlets work in the same fashion.

## There's more...

Like mailboxes, mail contacts, and mail-enabled user accounts have several Active Directory attributes that can be set such as job title, company, department, and more. To update these attributes, you can use the Set-\* cmdlets available for each respective type. For example, to update our mail contact we could use the Set-Contact cmdlet with the following syntax:

Set-Contact -Identity rjones `

-Title 'Sales Contractor' `

-Company Fabrikam `

-Department Sales

To modify the same settings for a mail-enabled user, use the Set-User cmdlet:

Set-User -Identity jdavis `

-Title 'Sales Contractor' `

-Company Fabrikam `

-Department Sales

Both cmdlets can be used to modify a number of different settings. Use the help system to view all of the available parameters.

## See also

* Using the help system in Chapter 1, PowerShell Key Concepts

Adding, modifying, and removing mailboxes

# Managing distribution groups

In many Exchange environments, distribution groups are relied upon heavily and require frequent changes. This recipe will cover the creation of distribution groups and how to add members to groups, which might be useful when performing these tasks interactively in the shell or through automated scripts.

## How to do it...

1. To create a distribution group, use the New-DistributionGroup cmdlet:

New-DistributionGroup -Name Sales

1. Once the group has been created, adding multiple members can be done easily using a one-liner:

Get-Mailbox -OrganizationalUnit Sales |

Add-DistributionGroupMember -Identity Sales

1. We can also create distribution groups whose memberships are set dynamically:

New-DynamicDistributionGroup -Name Accounting `

-Alias Accounting `

-IncludedRecipients MailboxUsers,MailContacts `

-OrganizationalUnit Accounting `

-ConditionalDepartment accounting,finance `

-RecipientContainer testlabs.se

## How it works...

There are two types of distribution groups that can be created with Exchange. First, there are regular distribution groups, which contain a distinct list of users. Secondly, there are dynamic distribution groups, whose members are determined at the time a message is sent based on a number of conditions or filters that have been defined. Both types have a set of cmdlets that can be used to add, remove, update, enable, or disable these groups.

By default, when creating a standard distribution group, the group scope will be set to Universal. You can create a mail-enabled security group using the   
New-DistributionGroup cmdlet by setting the -Type parameter to Security. If you do not provide a value for the -Type parameter, the group will be created using the Distribution group type.

You can mail-enable an existing Active Directory universal distribution group using the Enable-DistributionGroup cmdlet.

After creating the Sales distribution group in our previous example, we added all of the mailboxes in the Sales OU to the group using the Add-DistributionGroupMember cmdlet. You can do this in bulk or for one user at a time using the –Member parameter:

Add-DistributionGroupMember -Identity Sales -Member administrator

Distribution groups are a large topic and we're merely covering the basics here. See Chapter 5, Distribution Groups and Address Lists for in-depth coverage of distribution groups.

Dynamic distribution groups determine their membership based on a defined set of filters and conditions. When we created the Accounting distribution group, we used the -IncludedRecipients parameter to specify that only the MailboxUsers and MailContacts object types would be included in the group. This eliminates resource mailboxes, groups, or mail users from being included as members. The group will be created in the Accounting OU based on the value used with the -OrganizationalUnit parameter. Using the –ConditionalDepartment parameter, the group will only include users that have a department setting of either Accounting or Finance. And finally, since the -RecipientContainer parameter is set to the FQDN of the domain, any user located in the Active Directory could potentially be included in the group. You can create more complex filters for dynamic distribution groups using a recipient filter; see the recipe titled Working with Recipient Filters later in this chapter for an example.

You can modify both group types using the Set-DistributionGroup and Set-DynamicDistributionGroup cmdlets.

## There's more...

Just as when dealing with other recipient types, there are a couple of considerations   
that should be taken when it comes to removing distribution groups. You can remove the Exchange attributes from a group using the Disable-DistributionGroup cmdlet. The Remove-DistributionGroup cmdlet will remove the group object from the Active Directory and Exchange.

## See also

* Working with recipient filters
* Reporting on distribution group membership in Chapter 5, Distribution Groups and Address Lists
* Adding members to a distribution group from an external file in Chapter 5, Distribution Groups and Address Lists

Previewing dynamic distribution group membership in Chapter 5, Distribution Groups and Address Lists

# Managing resource mailboxes

In addition to mailboxes, groups, and external contacts, recipients can also include specific rooms or pieces of equipment. Locations such as a conference room or a classroom can be given a mailbox so they can be reserved for meetings. Equipment mailboxes can be assigned to physical, non-location specific resources such as laptops or projectors and can then be checked out to individual users or groups by booking time with the mailbox. In this recipe, we'll take a look at how you can manage resource mailboxes using the Exchange Management Shell.

## How to do it...

When creating a resource mailbox from within the shell, the syntax is similar to creating a mailbox for a regular user. For example, you still use the New-Mailbox cmdlet when creating a resource mailbox:

New-Mailbox -Name "CR7" -DisplayName "Conference Room 7" `

-UserPrincipalName CR7@contoso.com -Room

## How it works...

There are two main differences when it comes to creating a resource mailbox as opposed to a standard user mailbox. First, you need to use either the -Room switch parameter or the -Equipment switch parameter to define the type of resource mailbox that will be created. Second, you do not need to provide a password value for the user account. When using either of these resource mailbox switch parameters to create a mailbox, the New-Mailbox cmdlet will create a disabled Active Directory user account that will be associated with the mailbox.

The entire concept of room and equipment mailboxes revolves around the calendars used by these resources. If you want to reserve a room or a piece of equipment, you book time through Outlook or OWA with these resources for the duration that you'll need them. The requests sent to these resources need to be accepted, either by a delegate or automatically using the Resource Booking Attendant.

To configure the room mailbox created in the previous example to automatically accept new meeting requests, we can use the Set-CalendarProcessing cmdlet to set the Resource Booking Attendant for that mailbox to AutoAccept:

Set-CalendarProcessing CR7 -AutomateProcessing AutoAccept

When the Resource Booking Attendant is set to AutoAccept, the request will be immediately accepted as long as there is not a conflict with another meeting. If there is a conflict, an e-mail message will be returned to the requestor explaining that the request was declined due to scheduling conflicts. You can allow conflicts by adding the   
–AllowConflicts switch parameter to the previous command.

When working with resource mailboxes with AutomateProcessing set to AutoAccept, you'll get an automated e-mail response from the resource after booking time. This e-mail message will explain whether the request was accepted or declined, depending on your settings. You can add additional text to the response message that the meeting organizer   
will receive using the following syntax:

Set-CalendarProcessing -Identity CR7 `

-AddAdditionalResponse $true `

-AdditionalResponse 'For Assistance Contact Support at Ext. #3376'

This example uses the Set-CalendarProcessing cmdlet to customize the response messages sent from the CR23 room mailbox. You can see here that we've added a message that tells the user the help desk number to call if assistance is required. Keep in mind that you can only add additional response text when the AutomateProcessing property is set to AutoAccept.

If you do not want to automate the calendar processing for a resource mailbox then you'll need to add delegates that can accept or deny meetings for that resource. Again, we can   
turn to the Set-CalendarProcessing cmdlet to accomplish this:

Set-CalendarProcessing -Identity CR7 `

-ResourceDelegates "rmcleod@testlabs.se","trandolph@testlabs.se" `

-AutomateProcessing None

In this example, we've added two delegates to the resource mailbox and have turned off automated processing. When a request comes into the CR7 mailbox, both Ruben and Terrance will be notified and can accept or deny the request on behalf of the resource mailbox.

## There's more...

When it comes to working with resource mailboxes, another useful feature is the ability to assign custom resource properties to rooms and equipment resources. For example, you may have a total of 5, 10, or 15 conference rooms, but maybe only four of those have whiteboards. It might be useful for your users to know this information when booking a resource for a meeting where they will be conducting a training session.

Using the shell, we can add custom resource properties to the Exchange organization by modifying the resource schema. Once these custom resource properties have been added, they can then be assigned to specific resource mailboxes.

You can use the following code to add a whiteboard resource property to the Exchange organizations resource schema:

Set-ResourceConfig -ResourcePropertySchema 'Room/Whiteboard'

Now that the whiteboard resource property is available within the Exchange organization,   
we can add this to our Conference Room 7 mailbox using the following command:

Set-Mailbox -Identity CR7 -ResourceCustom Whiteboard

When users access the Select Rooms or Add Rooms dialog box in Outlook 2010, Outlook 2013 or Outlook 2016 they will see that Conference Room 7 has a whiteboard available.

### Converting mailboxes

If you’ve moved from an old version, you may have a number of mailboxes that were being used as resource mailboxes. Once these mailboxes have been moved over, they will be identified as Shared mailboxes. You can convert them to other types using the Set-Mailbox cmdlet so that they'll have all of the properties of a resource mailbox:

Get-Mailbox conf\* | Set-Mailbox -Type Room

You can run the Set-Mailbox cmdlet against each mailbox one at a time and convert them to Room mailboxes using the -Type parameter. Or, if you use a common naming convention, you may be able to do them in bulk by retrieving a list of mailboxes using a wildcard and piping them to Set-Mailbox, as shown previously.

## See also

* Adding, modifying, and removing mailboxes

Creating recipients in bulk using a CSV file

# Creating recipients in bulk using a CSV file

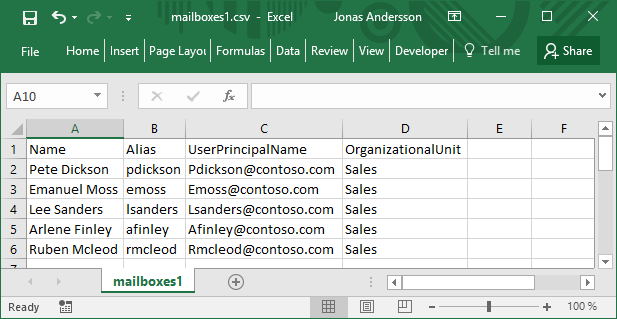
One of the most common bulk provisioning techniques used in the Exchange Management Shell makes use of comma-separated value (CSV) files. These files act sort of like a database table. Each record in this table is represented by one line in the file, and each field value is separated by a comma, which is used as a delimiter. In this recipe, you'll learn how to set up a CSV file and create recipients in bulk using the Exchange Management Shell.

## Getting ready

In addition to the Exchange Management Shell, you'll need to use Microsoft Excel to create a CSV file.

## How to do it...

1. In this example, we are going to create some mailboxes in bulk. We'll enter some data into Excel that will include the settings for five new mailboxes:



7081EN\_03\_01

1. Go to File | Save As and select CSV (Comma delimited) (\*.csv) for the file type. Save the file as C:\Scripts\mailboxes1.csv.
2. Within the Exchange Management Shell, create a secure password object to be used as an initial password for each mailbox:

$pass = ConvertTo-SecureString -AsPlainText P@ssw0rd01 `  
-Force

1. Import the CSV file and create the mailboxes:

Import-CSV C:\Scripts\mailboxes1.csv | % {

New-Mailbox -Name $\_.Name `

-Alias $\_.Alias `

-UserPrincipalName $\_.UserPrincipalName `

-OrganizationalUnit $\_.OrganizationalUnit `

-Password $pass `

-ResetPasswordOnNextLogon $true

}

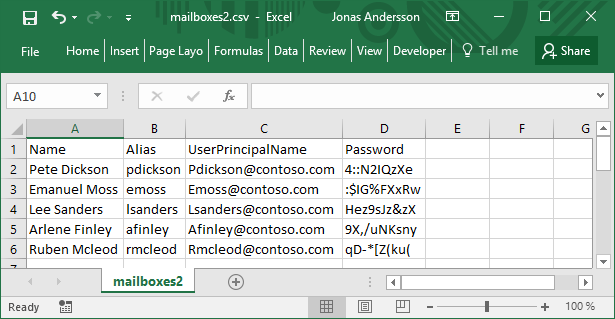
## How it works...

In this example, we're importing the CSV file into the shell and piping that information to the ForEach-Object cmdlet (using the % alias). For each record in the CSV file, we're running the New-Mailbox cmdlet, providing values for the -Name, -Alias,   
–UserPrincipalName and -OrganizationalUnit parameters. The properties for each record can be accessed inside the loop using the $\_ variable, which is the automatic variable that references the current object in the pipeline. The property names for each record match the header names used in the CSV file. As we create each mailbox, the password is set to the $pass variable. The –ResetPasswordOnNextLogon parameter is set to $true, which will require each user to reset their password after their first logon.

Using this technique, you can literally create thousands of mailboxes in a matter of minutes. This concept can also be applied to other recipient types, such as distribution groups and contacts. You just need to specify the appropriate parameter values in the CSV file and use the corresponding cmdlet for the recipient type. For example, if you want to bulk provision contacts from a CSV file, use the code from the previous example as a guide, and, instead of using the New-Mailbox cmdlet, use the New-MailContact cmdlet and whatever parameters are required based on your settings.

## There's more...

Let's take a look at an alternative approach to the previous example. Let's say that you don't want to set an initial password for each user, and, instead, you want to include this information in the CSV file so each new mailbox gets a unique password. Again, you'll need to set up a CSV file with the required values. For this example, your CSV file would look something like this:



7081EN\_03\_02

Notice that in the previous screenshot, we are using different column names for this new file. We've removed the OrganizationalUnit column and now have a Password column which will be used to create each mailbox with a unique password. After you're done creating the file, save it again as C:\Scripts\mailboxes2.csv.

Next, you can use the following code to create the mailboxes, specifying the path and file name to the CSV file created in the previous step:

Import-CSV C:\Scripts\Mailboxes2.CSV | % {

$pass = ConvertTo-SecureString -AsPlainText $\_.Password -Force

New-Mailbox -Name $\_.Name `

-Alias $\_.Alias `

-UserPrincipalName $\_.UserPrincipalName `

-Password $pass

}

As we loop through each record in the CSV file, we create a secure password object that can be used with the -Password parameter. The main difference here compared to the previous example is that each user gets a unique password and they do not need to reset their password the first time they log on.

### Taking it a step further

When provisioning recipients you'll probably need to do multiple things, such as set Active Directory attributes and configure distribution group membership. Let's take our previous example a step further:

Import-CSV C:\Scripts\NewMailboxes.csv | % {

New-Mailbox -Name $\_.Name `

-FirstName $\_.FirstName `

-LastName $\_.LastName `

-Alias $\_.Alias `

-UserPrincipalName $\_.UserPrincipalName `

-Password (ConvertTo-SecureString -AsPlainText P@ssw0rd `  
 -Force) `

-OrganizationalUnit $\_.OrganizationalUnit `

-Database DB1

Set-User -Identity $\_.Name `

-City $\_.City `

-StateOrProvince $\_.State `

-Title $\_.Title `

-Department $\_.Department

Add-DistributionGroupMember -Identity DL\_Sales `

-Member $\_.Name

Add-DistributionGroupMember -Identity DL\_Marketing `

-Member $\_.Name

}

Here we're still using a CSV file, but as we loop through each record we're calling multiple cmdlets to first create the mailbox, set some of the Active Directory attributes, and then add the mailbox to two separate distribution groups. In order to use this code, we would just need to create a CSV file that has columns for all of the values we're setting.

Now that we have this framework in place, we can add as many columns as we need to the CSV file and we can call any number of cmdlets for each record in the CSV.

## See also

* Looping through items in Chapter 1, PowerShell Key Concepts
* Adding, modifying, and removing mailboxes

Managing distribution groups

# Working with recipient filters

Starting with Exchange 2007 and continuing with the following Exchange versions, address lists, dynamic distribution groups, e-mail address policies, and global address lists can be customized with recipient filters that use OPATH filtering syntax. This replaces the legacy LDAP filtering syntax that was used in earlier versions of Exchange. We can also perform server-side searches using filters, which can greatly speed up our work. In this recipe, you'll learn how to work with these filters in the Exchange Management Shell.

## How to do it...

1. We can filter the results from the recipient Get-\* cmdlets using the   
   -Filter parameter:

Get-Mailbox -Filter {Office -eq 'Sales'}

1. In addition, we can use attribute filters to create distribution groups, e-mail address policies, and address lists using the -RecipientFilter parameter:

New-DynamicDistributionGroup -Name DL\_Accounting `

-RecipientFilter {

(Department -eq 'Accounting') -and

(RecipientType -eq 'UserMailbox')

}

## How it works...

In our first example, you can see that we've used the Get-Mailbox cmdlet to retrieve only the users that have the Office property set to the value Sales. This is more efficient then performing the following command, which would return the same results:

Get-Mailbox | ?{$\_.Office -eq 'Sales'}

This command uses the Where-Object cmdlet (using the ? alias) to retrieve only the mailboxes with their Office property set to Sales. We get back the same results, but it is less efficient than our original example. When filtering with Where-Object, every mailbox in the organization must be retrieved and evaluated before any results are returned. The benefit of using the -Filter parameter with the Get-Mailbox cmdlet is that the filtering is done on the server and not our client machines. The –Filter method is preferred when working in large environments.

There are a number of cmdlets that support this parameter. You can get an entire list with   
a simple one-liner:

get-excommand | ?{$\_.parameters.keys -eq 'filter'}

This uses the shell function get-excommand to retrieve a list of Exchange Management   
Shell cmdlets that support the -Filter parameter. If you are writing scripts or functions that need to query a large number of recipients, you'll want to try to use server-side filtering whenever possible.

Unfortunately, there are only a certain set of properties that can be filtered. For instance, we were able to filter using the Office property when using the Get-Mailbox cmdlet. Based on that, you may assume that, since OrganizationalUnit is a property of a mailbox object that you can filter on that as well, but that is not the case. The   
Get-Mailbox cmdlet provides an -OrganizationalUnit parameter that can be used to accomplish that task, so it's not always safe to assume that a particular property can be used within a filter. To view a list of common filterable properties that can be used with the –Filter parameter, see Appendix A at the end of this book.

In our second example, we used the New-DynamicDistributionGroup cmdlet to create   
a query-based group. The membership of this group is determined using the OPATH filter defined with the –RecipientFilter parameter. The syntax is similar and the same PowerShell operators can be used. Based on the settings used with our filter when we created the DL\_Accounting group, only mailboxes with their Department attribute set to Accounting will be included. Other recipient types, such as mail contacts and mail users, will not be included in the group, even though they may be in the Accounting department.

Dynamic distribution groups, address lists, and e-mail address policies can be configured   
with these filters. Again, to get the list of cmdlets that support this functionality, use the   
get-excommand shell variable:

get-excommand | ?{$\_.parameters.keys -eq 'recipientfilter'}

These cmdlets also have a limited number of filterable properties that can be used. To view a list of the most common properties used with the –RecipientFilter parameter, see Appendix A at the end of this book.

## There's more…

Instead of using the -RecipientFilter parameter, you have the option of using   
pre-canned filters. In some cases, this may be easier as it allows you to simply use a set   
of parameters and values as opposed to an OPATH filter. The following command would   
create our DL\_Accounting distribution group with the same members using the pre-canned filter parameters:

New-DynamicDistributionGroup -Name DL\_Accounting `

-IncludedRecipients MailboxUsers `

-ConditionalDepartment Accounting

As you can see, this is a little easier to read and probably easier to type into the shell. Although, there are only a few pre-canned parameters available and they may not always be useful depending on what you are trying to do, but it helps to be aware of this functionality. You can use Get-Help to view the entire list of available parameters for each cmdlet that supports recipient filters.

### Understanding variables in filters

One of the issues you may run into when working in the shell is the expansion of variables used within a filter. For example, this syntax is completely valid but will not currently work correctly in the Exchange Management Shell:

$office = "sales"

Get-Mailbox -Filter {Office -eq $office}

You might get some results from this command, but they will probably not be what you are expecting. This is because, when running the Get-Mailbox cmdlet, the value of the $office variable will not be expanded prior to the command being executed through the remote shell. What you end up with instead is a filter checking for a $null value. In order to fix this, you'll need to use syntax similar to the following:

$office = "sales"

Get-Mailbox -Filter "Office -eq '$office'"

This syntax will force any variables assigned within the -Filter parameter to be   
expanded before sending the command through the remote session, and you should   
get back the correct results.

## See also

* Managing distribution groups
* Using the help system in Chapter 1, PowerShell Key Concepts

Previewing dynamic distribution group membership in Chapter 5, Distribution Groups and Address Lists

# Adding and removing recipient e-mail addresses

There are several recipient types in Exchange 2016 and each one of them can support multiple e-mail addresses. Of course, the typical user mailbox recipient type is probably the first that comes to mind, but we also have distribution groups, contacts, and public folders, each of which can have one or more e-mail addresses. The syntax used for adding and removing e-mail addresses to each of these recipient types is essentially identical: the only thing that changes is the cmdlet that is used to set the address. In this recipe, you'll learn how to add or remove an e-mail address from an Exchange recipient.

## How to do it...

1. To add a secondary e-mail address to a mailbox, use the following command syntax:

Set-Mailbox dave -EmailAddresses @{add='dave@west.contoso.com'}

1. Multiple addresses can also be added using this technique:

Set-Mailbox dave -EmailAddresses @{

add='dave@east.contoso.com',

'dave@west.contoso.com',

'dave@corp.contoso.com'

}

1. E-mail addresses can also be removed using the following syntax:

Set-Mailbox dave -EmailAddresses @{remove='dave@west.contoso.com'}

1. Just as we are able to add multiple e-mail addresses at once, we can do the same when removing an address:

Set-Mailbox dave -EmailAddresses @{

remove='dave@east.contoso.com',

'dave@corp.contoso.com'

}

## How it works...

Adding and removing e-mail addresses was more challenging back in the days with Exchange 2007 management shell because it required that you work directly with the EmailAddresses collection, which is a multi-valued property. In order to modify the collection, you first had to save the object to a variable, modify it, and then write it back to the EmailAddresses object on the recipient. This made it impossible to update the   
e-mail addresses for a recipient with one command.

The Set-\* cmdlets used to manage recipients in Exchange 2016 now support a new syntax that allows us to use a hash table to modify the EmailAddresses property. As you can see from the code samples, we can simply use the Add and Remove keys within the hash table, and the assigned e-mail address values will be either added or removed as required. This is a nice change that makes it easier to do this in scripts and especially when working interactively in the shell.

The Add and Remove keywords are interchangeable with the plus (+) and minus (-) characters that serve as aliases:

Set-Mailbox dave -EmailAddresses @{

'+'='dave@east.contoso.com'

'-'='dave@west.contoso.com'

}

In the previous example, we've added and removed e-mail addresses from the mailbox.   
Notice that the + and - keywords need to be enclosed in quotes so PowerShell does not   
try to interpret them as the += and -= operators.

This syntax works with all of the Set-\* cmdlets that support the -EmailAddresses parameter:

* Set-CASMailbox
* Set-DistributionGroup
* Set-DynamicDistributionGroup
* Set-Mailbox
* Set-MailContact
* Set-MailPublicFolder
* Set-MailUser
* Set-RemoteMailbox

Keep in mind that in most cases the best way to add an e-mail address to a recipient is through the use of an e-mail address policy. This may not always be an option, but should be used first if you find yourself in a situation where addresses need to be added to a large number of recipients. With that said, it is possible do this in bulk using a simple foreach loop:

foreach($i in Get-Mailbox -OrganizationalUnit Sales) {

Set-Mailbox $i -EmailAddresses @{

add="$($i.alias)@west.contoso.com"

}

}

This code simply iterates over each mailbox in the Sales OU and adds a secondary e-mail address using the existing alias at west.contoso.com. You can use this technique and modify the syntax as needed to perform bulk operations.

## There's more...

Imagine a situation where you need to remove all e-mail addresses under a certain domain from all of your mailboxes. These could be secondary addresses that were added manually to each mailbox, or that used to be applied as part of an e-mail address policy that no longer applies. The following code can be used to remove all e-mail addresses from mailboxes under a specific domain:

foreach($i in Get-Mailbox -ResultSize Unlimited) {

$i.EmailAddresses |

?{$\_.SmtpAddress -like '\*@corp.contoso.com'} | %{

Set-Mailbox $i -EmailAddresses @{remove=$\_}

}

}

This code iterates through each mailbox in the organization and simply uses a filter to discover any e-mail addresses at corp.contoso.com. If any exist, the Set-Mailbox cmdlet will attempt to remove each of them from the mailbox.

## See also

* Adding, modifying, and removing mailboxes
* Working with contacts

Managing distribution groups

# Hiding recipients from address lists

There may be times when you'll need to hide a particular mailbox, contact, or distribution group from your Exchange address lists. This is a common task that is required when you have mailboxes, contacts, or public folders used by applications or staff in your IT department that should not be seen by end-users. In this recipe, we'll take a look at how you can disable these recipient types from the address lists using the Exchange Management Shell.

## How to do it...

To hide a mailbox from the Exchange address lists, use the Set-Mailbox command:

Set-Mailbox dave –HiddenFromAddressListsEnabled $true

## How it works...

As you can see, hiding a mailbox from address lists is pretty straight forward as it requires only a simple PowerShell one-liner. The –HiddenFromAddressListsEnabled parameter accepts a Boolean value, either $true or $false. To enable this setting, set the value to $true, and to disable it, set the value to $false.

There are multiple recipient types that can be hidden from address lists. Each of the following Set-\* cmdlets supports the -HiddenFromAddressListsEnabled parameter:

* Set-DistributionGroup
* Set-DynamicDistributionGroup
* Set-Mailbox
* Set-MailContact
* Set-MailPublicFolder
* Set-MailUser

Set-RemoteMailbox

## There's more...

Once you've hidden your recipients from the address lists, you may need to generate a report to list the objects that currently have the HiddenFromAddressListsEnabled setting enabled. Use the following command syntax to obtain this information:

Get-Mailbox -Filter {HiddenFromAddressListsEnabled -eq $true}

This searches for all mailboxes that have been hidden from address lists. It makes use of the –Filter parameter which keeps you from having to perform the filtering on the client side with the Where-Object cmdlet.

## See also

Working with recipient filters

# Configuring recipient moderation

The recipient moderation allows you to require approval for all e-mail messages sent to a particular recipient by a designated moderator. In this recipe, you'll learn how to configure the moderation settings on recipients using the Exchange Management Shell.

## How to do it...

1. To enable moderation for a distribution group, use the   
   Set-DistributionGroup cmdlet:

Set-DistributionGroup -Identity Executives `

-ModerationEnabled $true `

-ModeratedBy administrator `

-SendModerationNotifications Internal

1. These same parameters can be used to configure moderation for a mailbox when using the Set-Mailbox cmdlet:

Set-Mailbox -Identity dave `

-ModerationEnabled $true `

-ModeratedBy administrator `

-SendModerationNotifications Internal

## How it works...

When you enable moderation for a recipient, any e-mail message sent to that recipient must be reviewed by a moderator. When a message is sent to a moderated recipient, the moderator will receive the message and determine whether or not it should be accepted. This is done by the moderator through Outlook or OWA by clicking on an Approve or Reject button in the e-mail message. If the moderator accepts the message, it is delivered to the group. If it is rejected, the message is deleted, and, depending on the SendModerationNotifications setting, the sender may receive an e-mail informing them the message has been rejected.

Moderation can be enabled for any recipient, whether it's a mailbox, mail contact, mail user, distribution group, or mail-enabled public folder. The cmdlets for each of these recipient types can be used to configure moderation when a recipient is being created with the New-\* cmdlets, or after the fact using the Set-\* cmdlets. To view the list of cmldets that can be used to enable moderation, run the following command:

get-excommand | ?{$\_.parameters.keys -eq 'ModerationEnabled'}

In our first example, we enabled moderation for the Executives distribution group, specifying that the administrator account will be used as the moderator for the group. As you can see in the example, we've used multiple parameters when running the command, but only the -ModerationEnabled parameter is required to change the moderation setting for the group. If no value is specified for the -ModeratedBy parameter, the group owner will review and approve the messages sent to the group. You can specify one or more owners when running the Set-DistributionGroup cmdlet with the -ManagedBy parameter.

The -SendModerationNotifications parameter allows you to control the status messages sent to the originator of a message that was sent to a moderated recipient.   
We have the option of using the following values for this parameter:

* Always: Notifications are sent to all internal and external senders
* Internal: Notifications are only sent to users within the organization

Never: Notifications are not sent at all

If no value is provided for the -SendModerationNotifications parameter when you enable moderation for a group, the setting will default to Always.

## There's more...

There is an exception to every rule, and, of course, there may be times where we need to bypass moderation for certain recipients. Let's say that we need to bypass specific users from moderation on the Executives distribution group. The group moderator or group owners are already exempt from moderation. To exclude others we can specify a list of one or more recipients using the -BypassModerationFromSendersOrMembers parameter when running the Set-DistributionGroup cmdlet.

For example, to exclude a recipient named Bob from moderation on the Executives distribution group, run the following command:

Set-DistributionGroup -Identity Executives `

-BypassModerationFromSendersOrMembers bob@contoso.com

If you want the members of the moderated group, or any other distribution group, to be excluded from moderation, simply use the previous syntax and assign the identity of the   
group to the -BypassModerationFromSendersOrMembers parameter. You can assign multiple users or distribution groups at once; by separating each value with a comma.

Keep in mind that running the previous command will overwrite the existing list of bypassed members if any have been defined. For an example of how to add a new item to a multi-valued property, see the Working with arrays and hash tables in Chapter 1, PowerShell Key Concepts.

Additionally, you may need to bypass moderation for a group of several individual recipients. While you could add them one by one, this could be very time-consuming if you are dealing with a large number of recipients. Let's say that you want to exclude all the users in the San Diego office from moderation:

$exclude = Get-Mailbox –Filter {Office –eq ‘Seattle’} |

Select-Object -ExpandProperty alias

Set-DistributionGroup -Identity Executives `

-BypassModerationFromSendersOrMembers $exclude

In this example, we create a collection that contains the alias for each mailbox in the Seattle Office. Next, we use the Set-DistributionGroup cmdlet to exclude all of those recipients from moderation using a single command. While this might be useful in certain situations, it's easier to bypass moderation based on groups. If a group has been bypassed   
for moderation, you can simply manage the membership of the group and you don't need to worry about continuously updating individual recipients that are on the bypass list.

## See also

Managing distribution groups

# Configuring message delivery restrictions

Since distribution groups contain multiple members, you may want to place restrictions on who can send messages to these recipients. Exchange allows you to tightly control these settings and provides several options when it comes to placing message delivery restrictions on groups. We can also place restrictions on other recipient types in the organization. This recipe will show you how to configure these options from the Exchange Management Shell.

## How to do it...

To restrict who can send messages to a group, use the Set-DistributionGroup cmdlet:

Set-DistributionGroup -Identity Sales `

-AcceptMessagesOnlyFrom 'Bob Smith','John Jones'

After running this command, only the users Bob Smith and John Jones can send messages to the Sales distribution group.

## How it works...

The -AcceptMessagesOnlyFrom parameter allows you to specify one or more recipients who are allowed to send messages to a distribution group. These recipients can be regular users with mailboxes or contacts.

You can add individual recipients and distribution groups to the accepted senders list using the following syntax:

Set-DistributionGroup -Identity Sales `

-AcceptMessagesOnlyFromSendersOrMembers Marketing,bob@contoso.com

In this example, we're allowing both the Marketing distribution group and Bob, an individual recipient, to the accepted senders list for the Sales distribution group. Doing so will allow Bob and any members of the Marketing distribution group to send messages to the Sales group.

Keep in mind that, when using these parameters, any existing accepted recipients that   
have been configured will be overwritten. For an example of how to add a new item to a   
multi-valued property, see the in Chapter 1 titled Working with arrays and hash tables.

Delivery restrictions can be placed on any recipient, whether it's a mailbox, mail contact, mail user, distribution group, or mail-enabled public folder. The Set-\* cmdlets for each of these recipient types can be used to configure delivery restrictions. To view the list of cmdlets that can be used to do this, run the following command:

get-excommand | ?{$\_.parameters.keys -eq 'AcceptMessagesOnlyFrom'}

If you need to add a large list of users to the accepted senders list, you can create a collection and assign it to the -AcceptMessagesOnlyFrom parameter:

$finance = Get-Mailbox -Filter {Office -eq 'Finance'}

Set-DistributionGroup -Identity Sales `

-AcceptMessagesOnlyFrom $finance

You can wipe out these settings and allow messages from all senders by setting the   
value to $null:

Set-DistributionGroup -Identity Sales `

-AcceptMessagesOnlyFromSendersOrMembers $null

Similar to the previous examples, we can reject messages from a specific user or member   
of a distribution list using the -RejectMessagesFromSendersOrMembers parameter:

Set-DistributionGroup -Identity Executives `

-RejectMessagesFromSendersOrMembers HourlyEmployees

In this example, Exchange will reject any message sent from a member of the HourlyEmployees distribution group to the Executives group.

## There's more...

When you create a distribution group, the default configuration is to reject messages from senders who are not authenticated. This means that users outside of your organization will not be able to send messages to your distribution groups. Generally, this is the desired configuration, but if needed, you can modify this setting on a distribution group to accept messages from external users using the following syntax:

Set-DistributionGroup -Identity HelpDesk `

-RequireSenderAuthenticationEnabled $false

You can see here that we've disabled sender authentication for the HelpDesk distribution group. You can re-enable it at any time by setting the previous parameter value to $true.

## See also

Managing distribution groups

# Managing automatic replies and out of office settings for a user

Since Exchange 2010, we got some valuable set of cmdlets that can be used to manage and automate the configuration of a user's Out of Office settings. In this recipe, we'll take a look at how to use these cmdlets from the Exchange Management Shell for Exchange 2016.

## How to do it...

1. To view the Out of Office settings for a mailbox, use the following syntax:

Get-MailboxAutoReplyConfiguration dave

1. You can change the Out of Office settings for a mailbox using the syntax shown next. For example, to disable Out of Office for a mailbox, use the following command:

Set-MailboxAutoReplyConfiguration dave `  
-AutoReplyState Disabled

## How it works...

Retrieving the settings for a mailbox simply requires that you run the   
Get-MailboxAutoReplyConfiguration cmdlet and specify the identity of the mailbox, as shown in the previous example.   
The Set-MailboxAutoReplyConfiguration cmdlet supports multiple parameters that can be used to customize the settings use for the mailbox autoreply configuration:

Set-MailboxAutoReplyConfiguration dave `

-AutoReplyState Scheduled `

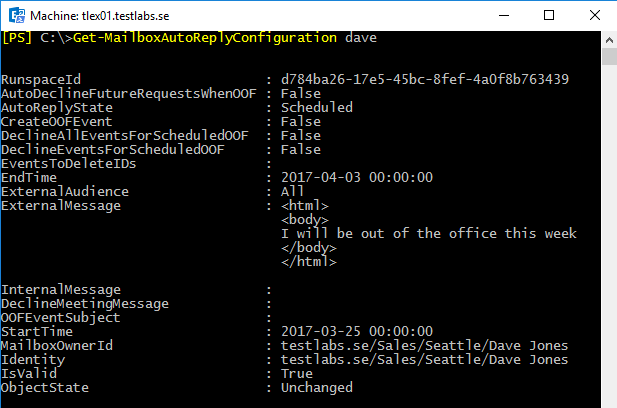
-StartTime 3/25/2017 `

-EndTime 4/3/2017 `

-ExternalMessage "I will be out of the office this week"

In this command, we set the AutoReplyState, specify a StartTime and EndTime, and set the ExternalMessage. When the StartTime date is reached, the mailbox will proceed to automatically reply to messages using the specified ExternalMessage until the EndTime date is reached. If you want automatic replies to be enabled indefinitely, set the AutoReplyState to Enabled.

To view the settings configured in the previous command, we can use the   
Get-MailboxAutoReplyConfiguration cmdlet, as shown in the following screenshot:



7081EN\_03\_03

You can see from viewing the mailbox auto-reply settings for this mailbox that only external replies are enabled. To enable internal Out of Office messages, you could run the previous set command and specify a message using the –InternalMessage parameter. Or you can use them both using a single command.

The -InternalMessage and -ExternalMessage parameters support HTML-formatted messages. If you want to set custom HTML code when configuring the auto-reply configuration from the shell, you can use the following command syntax:

Set-MailboxAutoReplyConfiguration dave `

-ExternalMessage (Get-Content C:\Scripts\oof.html)

This command will read in a custom HTML formatted message from an external file   
and use that data when setting the internal or external message. This will allow you to work on the file from the HTML editor of your choice and import the code using a simple command from the shell.

By default, the -ExternalAudience parameter will be set to All if no value is specified. The remaining options are Known and None. Setting the external audience to Known will only send automatic replies to external users who are listed as contacts in the users mailbox.

## There's more...

These cmdlets can be useful when making mass updates and when running reports. For example, to determine all of the users that currently have Out of Office enabled, you can run the following command:

Get-Mailbox –ResultSize Unlimited |

Get-MailboxAutoReplyConfiguration |

?{$\_.AutoReplyState -ne "Disabled"} |

Select Identity,AutoReplyState,StartTime,EndTime

This one-liner will check every mailbox in the organization and return only the mailboxes with the auto-reply state set to either Enabled or Scheduled.

Notice that Out of Office configuration set by administrators will override any configuration done by the end-user.

# Adding, modifying, and removing server-side inbox rules

This set of cmdlets were introduced back in Exchange 2010, so they have been around for some time but still worth bringing up. They can be used to manage server-side inbox rules for mailboxes in your organization. We now have the ability to add, remove, update, enable, and disable the inbox rules for mailboxes from within the Exchange Management Shell. This new functionality allows administrators to quickly resolve mailbox issues related to inbox rules, and allows them to easily deploy and manage inbox rules in bulk using just a few simple commands. In this recipe, you'll learn how to work with the inbox rules cmdlets in Exchange 2016.

## How to do it...

1. To create an inbox rule, use the New-InboxRule cmdlet:

New-InboxRule -Name Sales -Mailbox dave `

-From sales@contoso.com `

-MarkImportance High

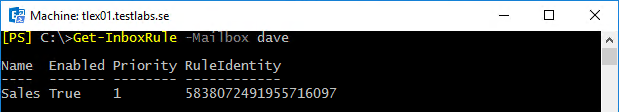
1. You can change the configuration of an inbox rule using the   
   Set-InboxRule cmdlet:

Set-InboxRule -Identity Sales -Mailbox dave `  
-MarkImportance Low

1. Use the Enable-InboxRule and Disable-InboxRule cmdlets to turn a rule on or off:

Disable-InboxRule -Identity Sales -Mailbox dave

1. The Get-InboxRule cmdlet will return all of the server-side rules that have been created for a specified mailbox. The output from the command is shown in the following screenshot:



7081EN\_03\_04

1. To remove an inbox rule, use the Remove-InboxRule cmdlet:

Remove-InboxRule -Identity Sales -Mailbox dave `  
-Confirm:$false

## How it works...

Inbox rules are used to process messages sent to a mailbox based on a certain set of criteria, and to then take an action on that message if the condition is met. In the previous example, we created an inbox rule for the mailbox that would mark messages from the sales@contoso.com address with high importance. The New-InboxRule cmdlet provides a number of rule predicate parameters that allow you to define the conditions used for the rules you create.

Let's take a look at another example. Say that we want to create a rule that will check the subject or body of all incoming messages for a certain keyword. If there is a match, we'll send the message to the deleted items folder:

New-InboxRule -Name "Delete Rule" `

-Mailbox dave `

-SubjectOrBodyContainsWords "Delete Me" `

-DeleteMessage $true

In addition to conditions and actions, we can also add exceptions to these rules. Consider the following example:

New-InboxRule -Name "Redirect to Andrew" `

-Mailbox dave `

-MyNameInToOrCcBox $true `

-RedirectTo "Andrew Castaneda" `

-ExceptIfFrom "Alfonso Mcgowan" `

-StopProcessingRules $true

In this example, once again we're creating an inbox rule in Dave's mailbox. The condition MyNameInToOrCcBox is set to $true so that any message with the mailbox name in the   
To or CC fields will be processed by this rule. The action is the RedirectTo setting, and   
that will redirect the message to Andrews's mailbox, except if the message was sent from Alfonso's mailbox. Finally, the -StopProcessingRules parameter is set to $true, meaning that, once this rule is processed, Exchange will not process any other rules in this mailbox. The -StopProcessingRules parameter is an optional setting and is provided to give you another level of flexibility when it comes to controlling the way the rules are applied.

It's important to note that when you add, remove, update, enable, or disable server-side rules using the \*-InboxRule cmdlets, any client-side rules created by Outlook will be removed.

In all of these examples, we've specified the mailbox identity and have been configuring   
the rules of a single mailbox. If you do not provide a value for the -Mailbox parameter, the \*-InboxRule cmdlets will execute against the mailbox belonging to the user that is running the command.

## There's more...

Now let's take a look at a practical example of how you might create inbox rules in bulk.   
The following code will create an inbox rule for every mailbox in the Sales OU:

$sales = Get-Mailbox -OrganizationalUnit contoso.com/sales

$sales | %{

New-InboxRule -Name Junk `

-Mailbox $\_.alias `

-SubjectContainsWords "[Spam]" `

-MoveToFolder "$($\_.alias):\Junk Email"

}

What we are doing here is using the -SubjectContainsWords parameter to check for a subject line that starts with "[Spam]". If there is a match, we move the message to the Junk Email folder within that user's mailbox. As you can see, we are looping through each mailbox using the ForEach-Object cmdlet (using the % alias) and, within the loop, we specify the identity of the user when creating the inbox rule and when specifying the folder id, using the $\_.alias property.

Even if you are logged in using an account in the Organization Management group, you may receive errors when trying to use the –MoveToFolder parameter when creating an inbox rule in another user's mailbox. Assigning FullAccess permissions to the mailbox in question should resolve this issue. For more details, see Granting administrators full access to mailboxes in Chapter 10, Exchange Security.

Make sure to verify the folder names, these varies depending on the selected language and is set per mailbox.

## See also

Granting users full access permissions to mailboxes in Chapter 10,   
Exchange Security

# Managing mailbox folder permissions

These cmdlets where introduced back in Exchange 2010, they can be used to manage the permissions on the folders inside a mailbox. When it comes to managing recipients, one of the most common tasks that administrators and support personnel perform on a regular basis is updating the permissions on the calendar of a mailbox. In most corporate environments, calendars are shared amongst employees and often special rights need to be delegated to other users allowing them to add, remove, update, or change the items on a calendar. In this recipe, we'll cover the basics of managing mailbox folder permissions from within the shell, but we will focus specifically on calendar permissions since that is a common scenario. Keep in mind that the cmdlets used in this recipe can be used with any folder within a mailbox.

## How to do it...

To allow users to view the calendar for a specific mailbox, use the following command:

Set-MailboxFolderPermission -Identity dave:\Calendar `

-User Default `

-AccessRights Reviewer

## How it works...

In this example, we're giving the Default user the ability to read all items in the calendar of the specified mailbox by assigning the Reviewer access right. This would give every user in the organization the ability to view the calendar items for this mailbox. There are four cmdlets in total that can be used to manage the mailbox folder permissions:

* Add-MailboxFolderPermission
* Get-MailboxFolderPermission
* Remove-MailboxFolderPermission

Set-MailboxFolderPermission

The Add and Set-MailboxFolderPermission cmdlets both provide an   
-AccessRights parameter that is used to set the appropriate permissions on the folder specified in the command. In the previous example, instead of assigning the Reviewer role, we could have assigned the Editor role to the Default user, giving all users the ability to completely manage the items in the calendar. The possible values that can be used with the -AccessRights parameter are as follows:

* ReadItems: The user assigned this right can read items within the designated folder.
* CreateItems: The user assigned this right can create items within the   
  designated folder.
* EditOwnedItems: The user assigned this right can edit the items that the user owns in the designated folder.
* DeleteOwnedItems: The user assigned this right can delete items that the user owns in the designated folder.
* EditAllItems: The user assigned this right can edit all items in the   
  designated folder.
* DeleteAllItems: The user assigned this right can delete all items in the designated folder.
* CreateSubfolders: The user assigned this right can create subfolders in the designated folder.
* FolderOwner: The user assigned this right has the right to view and move the folder and create subfolders. The user cannot read items, edit items, delete items, or create items.
* FolderContact: The user assigned this right is the contact for the   
  designated folder.

FolderVisible: The user assigned this right can view the specified folder, but can't read or edit items within it.

The following roles are made up by one or more of the permissions specified in the previous list and can also be used with the -AccessRights parameter:

* None: FolderVisible
* Owner: CreateItems, ReadItems, CreateSubfolders, FolderOwner, FolderContact, FolderVisible, EditOwnedItems, EditAllItems, DeleteOwnedItems, DeleteAllItems
* PublishingEditor: CreateItems, ReadItems, CreateSubfolders, FolderVisible, EditOwnedItems, EditAllItems, DeleteOwnedItems, DeleteAllItems
* Editor: CreateItems, ReadItems, FolderVisible, EditOwnedItems, EditAllItems, DeleteOwnedItems, DeleteAllItems
* PublishingAuthor: CreateItems, ReadItems, CreateSubfolders, FolderVisible, EditOwnedItems, DeleteOwnedItems
* Author: CreateItems, ReadItems, FolderVisible, EditOwnedItems, DeleteOwnedItems
* NonEditingAuthor: CreateItems, ReadItems, FolderVisible
* Reviewer: ReadItems, FolderVisible

Contributor: CreateItems, FolderVisible

## There's more...

Using the \*-MailboxFolderPermission cmdlets makes it easier to perform bulk operations on many mailboxes at once. For example, let's say that you need to assign Reviewer permissions to all employees on every mailbox calendar in the organization.   
You can use the following code to accomplish this task:

$mailboxes = Get-Mailbox -ResultSize Unlimited

$mailboxes | %{

$calendar = Get-MailboxFolderPermission `  
"$($\_.alias):\Calendar" -User Default

if(!($calendar.AccessRights)) {

Add-MailboxFolderPermission "$($\_.alias):\Calendar" `

-User Default -AccessRights Reviewer

}

if($calendar.AccessRights -ne "Reviewer") {

Set-MailboxFolderPermission "$($\_.alias):\Calendar" `

-User Default -AccessRights Reviewer

}

}

First, we use the Get-Mailbox cmdlet to retrieve all mailboxes in the organization and store that result in the $mailboxes variable. We then loop through each mailbox in the $mailboxes collection. Within the loop, we retrieve the current calendar settings for the Default user, using the Get-MailboxFolderPermission cmdlet, and store the output in the $calendar variable. If the Default user has not been assigned any rights to the calendar, we use the Add-MailboxFolderPermission cmdlet to add the Reviewer access right.

If the Default user has been assigned calendar permissions, we check to see if the access rights are set to Reviewer. If not, we modify the existing setting for the Default user to the Reviewer access right.

Note that if these users above have never been logging into their mailbox, the cmdlet will fail on that particular mailbox.

## See also

Granting users full access permissions to mailboxes in Chapter 10,   
Exchange Security

# Importing user photos into Active Directory

One of the most popular features back when it was introduced in Exchange 2010 was the ability to view user photos. Even though newer Outlook versions has a built-in social connector this may be applicable to Outlook 2013/2016 as well for organizations. This feature is based on possibility by importing an image into the thumbnailPhoto attribute for a given user account in Active Directory. This image can then be displayed when viewing a message or browsing the Global Address List within Outlook 2010 or newer. This was a highly requested enhancement, and the addition of this new feature makes it easier, especially in large organizations, to identify co-workers and get to know the people you are working with. In this recipe, we'll look at how you can import user photographs into Active Directory.

## Getting ready

In addition to the Exchange Management Shell, you will need access to the Active   
Directory administration tools for this recipe. The Remote Server Administration Tools   
pack (RSAT-ADDS) is a prerequisite required by Exchange 2016 setup, so it will already   
be installed on an Exchange 2016 server and you can use the tools from there, if needed.

## How to do it...

First, you need to update the Active Directory schema to ensure that the thumbnailPhoto attribute will be replicated to the Global Catalog. Your account will need to be a member of the schema admins group in Active Directory. On a machine with the Active Directory administration tools installed, do the following:

1. In the Exchange Management Shell or a cmd console, run the following command to register the Active Directory Schema extension:  
   Regsvr32 schmmgmt.dll.
2. Start the MMC console by clicking on Windows button | Run, type MMC, and click OK.
3. Go to File and click on Add/Remove Snap-In.
4. Add the Active Directory Schema Snap-In and click OK.
5. Under Active Directory Schema, highlight the Attributes node, and locate the thumbnailPhoto attribute.
6. Right click on the thumbnailPhoto attribute and click on Properties.
7. On the Properties page, select Replicate this attribute to the Global Catalog, and click OK.

At this point, the required Active Directory steps have been validated and you can now import a photo into Active Directory using the new cmdlet Set-UserPhoto:

Set-UserPhoto -Identity dave -PictureData ` ([System.IO.File]::ReadAllBytes("C:\Scripts\dave.jpg"))

## How it works...

Each user account or contact object in Active Directory has a thumbnailPhoto attribute that can be used to store binary data. The Set-UserPhoto cmdlet is used for reading a .jpg file into a byte array and importing it into the thumbnailPhoto attribute of the user account or contact in Active Directory, using the parameter PictureData followed by the .NET class [System.IO.File]::ReadAllBytes. Once the data has been imported into Active Directory, Outlook will query the thumbnailPhoto attribute of each user and display their photo when you receive an e-mail message from them, or when you are viewing their information in the Global Address List.

If you need to remove a photo for a user or a contact, use the   
Remove-UserPhoto cmdlet followed by the parameter Identity and a unique identifier of the user/contact object.

There are a few things to keep in mind when you decide to load photos into Active Directory for your users. The recommended thumbnail photo size in pixels is 96x96 pixels. Finally, be conscious about the size of your NTDS database in Active Directory. If you only have a small amount of users, then this will probably not be a huge issue. If you have hundreds of thousands of users there will be some serious replication traffic if you suddenly import photos for each of those users. Make sure to plan accordingly.

## There's more…

Outlook clients operating in cached mode will use the thumbnailPhoto attribute configuration of the Offline Address Book (OAB) to determine how to access photos. By default, the thumbnailPhoto attribute is an Indicator attribute, meaning that it points Outlook to Active Directory to retrieve the image. If you want to disable thumbnail photos for cached-mode clients, remove the attribute using the Remove method of the ConfigureAttrbutes collection:

$oab = Get-OfflineAddressBook ‘Default Offline Address Book’

$oab.ConfiguredAttributes.Remove('thumbnailphoto,indicator')

Set-OfflineAddressBook ‘Default Offline Address Book’ `

-ConfiguredAttributes $oab.ConfiguredAttributes

If you want offline clients to be able to view thumbnail photos, you can add the thumbnailPhoto attribute as a value attribute using the Add method:

$oab = Get-OfflineAddressBook ‘Default Offline Address Book’

$oab.ConfiguredAttributes.Add('thumbnailphoto,value')

Set-OfflineAddressBook ‘Default Offline Address Book’ `

-ConfiguredAttributes $oab.ConfiguredAttributes

If you work in a medium or large organization, this could make for an extremely large OAB. Again, make sure to plan accordingly. Use the following command to update the OAB after these configuration changes have been made:

Update-OfflineAddressBook 'Default Offline Address Book'

### Taking it a step further

If you are going to take advantage of this function, you are likely going to do this in bulk for existing employees, or as new employees are hired, and this may require some automation. Let's say that your company issues a security badge with a photo for each employee. You have each of these photos stored on a file server in jpeg format. The file names of the photos use the Exchange alias for the users associated mailbox. The following script can be used in this scenario to import the photos in bulk:

$photos = Get-ChildItem \\server01\employeephotos -Filter \*.jpg

foreach($i in $photos) {

[Byte[]]$data = gc -Path $i.fullname -Encoding Byte `  
 -ReadCount 0

Set-UserPhoto -Identity $i.basename -PictureData $data `   
 -Confirm:$false

}

First, this code creates a collection of jpeg files in the \\server01\employeephotos share and stores the results in the $photos object. We're using the -Filter parameter with the Get-ChildItem cmdlet so that the command only returns files with a .jpg extension. The items returned from the Get-ChildItem cmdlet are FileInfo objects which contain several properties that include detailed information about each file, such as the filename and the full path to the file.

As we loop through each photo in the collection, you can see that inside the loop we're casting the output from Get-Content (using the gc alias) to [Byte[]] and storing the result in the $data variable. We can determine the path to the file using the FullName property of the FileInfo object that represents the current jpeg file being processed in the loop. We then use the Set-UserPhoto cmdlet to import the data for the current user in the loop. The BaseName property of a FileInfo object returns the file name without the extension; therefore, we use this property value to identify which user we're importing the photo for when executing the Set-UserPhoto cmdlet.

## See also

Transferring files through remote PowerShell in Chapter 2, Exchange Management Shell Common Tasks