Microsoft Exchange Server PowerShell Cookbook
Third Edition

Microsoft Exchange Server 2013 is a complex messaging system. Windows PowerShell 5 can be used in conjunction with Exchange Server 2013 to automate and manage routine and complex tasks to save time and money and eliminate errors.

Starting by going through key PowerShell concepts and the Exchange Management Shell, this book will get you automating tasks that used to take hours in no time. Diving deeper, you will then manage your mailbox database, client access, and your transport servers with simple but effective scripts.

This book finishes with advanced recipes on Exchange Server problems, such as managing distribution groups and maintaining high availability and security.

Who this book is written for
This book is for messaging professionals who want to build real-world scripts with Windows PowerShell 5 and the Exchange Management Shell. If you are a network or systems administrator responsible for managing and maintaining Exchange Server 2013, you will find this highly useful.

What you will learn from this book

- New features and capabilities of PowerShell 5 and Exchange Server 2013 SP1
- Get to grips with the core PowerShell concepts required to master the Exchange Management Shell, such as pipelining, working with objects, formatting output, and writing scripts
- Use simple PowerShell scripts and commands to get powerful effects
- Generate detailed reports, send the output of commands in e-mail messages, and schedule scripts to run automatically
- Import, export, move mailboxes, and delete messages from mailboxes using the command line
- Configure transport server settings such as mail relay, tracking logs, transport rules, delivery reports, and more
- Manage mailbox and public folder databases

Over 120 recipes to help you manage and administrate Exchange Server 2013 Service Pack 1 with PowerShell 5
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 7 'Managing Client Access'
- A synopsis of the book’s content
Jonas Andersson is a devoted person who is constantly developing himself and his skills. He started out in the IT business in 2004 and initially worked in a support center, where he got basic knowledge of the industry. In 2007, he started his career as a Microsoft Infrastructure consultant, and from 2008 onward, his focus has been on Microsoft Exchange.

Even though his focus is on Microsoft Exchange, his interests include migrations, backup, storage, archiving, and so on. At the start of 2010, he was employed at a large outsourcing company as a messaging specialist, specializing in Microsoft Exchange. His work includes designing, implementing, and developing messaging solutions for enterprise customers.

His unique knowledge makes him a key figure in large and complex migration projects, where he works on design and implementation. Examples of these projects include migrations from the IBM Domino mail platform to Microsoft Exchange 2007/2010/2013 and Office 365, using Quest Software with full coexistence between the systems for mail flow, directory synchronization, and free busy lookups.

In 2014, he joined Microsoft Consulting Services, and from then onward, his focus has been on Office 365 but also on-premises Exchange Server. At the start of 2015, he changed his role to a deployment consultant with Microsoft's Office 365 Global Practice EMEA team.

He writes articles on his blog (http://www.testlabs.se/blog), Twitter, and other forms of social media.
As a reward for his work in the community, he was awarded the Microsoft Most Valuable Professional for the Microsoft Exchange Server product in 2014. He was also awarded the Microsoft Community Contributor Award in both 2011 and 2012.

Mike Pfeiffer is an accomplished IT architect, consultant, and conference speaker, with over 15 years of experience in the tech industry. He has published books, blogs, white papers, and training courses on a variety of topics related to infrastructure architecture, deployment automation, configuration management, and more. He has a passion for technology and enjoys learning as much as writing and teaching.
Preface

This book is full of immediately usable task-based recipes for managing and maintaining your Microsoft Exchange 2013 environment with Windows PowerShell 5.0 and the Exchange Management Shell. The focus of this book is to show you how to automate routine tasks and solve common problems. While the Exchange Management Shell literally provides hundreds of cmdlets, we will not cover every single one of them individually. Instead, we’ll focus on the common, real-world scenarios. You’ll be able to leverage these recipes right away, allowing you to get the job done quickly, and the techniques that you’ll learn will allow you to write your own amazing commands and scripts with ease.

What this book covers

Chapter 1, PowerShell Key Concepts, introduces several PowerShell core concepts, such as command syntax and parameters, working with the pipeline, loops, and conditional logic. The topics covered in this chapter lay the foundation for the remaining code samples in each chapter.

Chapter 2, Exchange Management Shell Common Tasks, covers day-to-day tasks and general techniques for managing Exchange from the command line. The topics include configuring manual remote shell connections, exporting reports to external files, sending e-mail messages from scripts, and scheduling scripts to run with the Task Scheduler.

Chapter 3, Managing Recipients, demonstrates some of the most common recipient-related management tasks, such as creating mailboxes, distribution groups, and contacts. You’ll also learn how to manage server-side inbox rules, Out of Office settings, and import user photos into Active Directory.

Chapter 4, Managing Mailboxes, shows you how to perform various mailbox management tasks that include moving mailboxes, importing and exporting mailbox data, and the detection and reparation of corrupt mailboxes. In addition, you’ll learn how to delete and restore items from a mailbox and manage the new public folders.
Chapter 5, *Distribution Groups and Address Lists*, takes you deeper into distribution group management. The topics include distribution group reporting, distribution group naming policies, and allowing end users to manage distribution group membership. You'll also learn how to create Address Lists and Hierarchal Address Books.

Chapter 6, *Mailbox Database Management*, shows you how to set database settings and limits. Report generation for mailbox database size, average mailbox size per database, and backup status is also covered in this chapter.

Chapter 7, *Managing Client Access*, covers the management of ActiveSync, OWA, POP, and IMAP. It also covers the configuration of these components in Exchange 2013. We'll also take a look at controlling connections from various clients, including ActiveSync devices.

Chapter 8, *Managing Transport Servers*, explains the various methods used to control the mail flow within your Exchange organization. You'll learn how to create, send, and receive connectors, allow application servers to relay mail, and manage transport queues.

Chapter 9, *High Availability*, covers the implementation and management tasks related to Database Availability Groups. The topics include creating DAGs, adding mailbox database copies, and performing maintenance on DAG members. It also covers a new feature called Automatic Reseed.

Chapter 10, *Exchange Security*, introduces you to the new Role Based Access Control permissions model. You'll learn how to create custom RBAC roles for administrators and end users, and also how to manage mailbox permissions and implement SSL certificates.

Chapter 11, *Compliance and Audit Logging*, covers the new compliance and auditing features included in Exchange 2013. Topics such as archiving mailboxes and discovery search are covered here, as well as administrator and mailbox audit logging.

Chapter 12, *Scripting with the Exchange Web Services Managed API*, introduces you to advanced scripting topics that leverage Exchange Web Services. In this chapter, you'll learn how to write scripts and functions that go beyond the capabilities of the Exchange Management Shell cmdlets.

*Appendix A, Common Shell Information*, is a reference for the variables, scripts, and the filtering functions. These references will help you when writing scripts or running interactive scripts.

*Appendix B, Query Syntaxes*, is a reference for the Advanced Query Syntax. In this section, you will find lots of different examples that can be used in the real world.
Managing Client Access

In this chapter, we will cover the following topics:

- Managing ActiveSync, OWA, POP3, and IMAP4 mailbox settings
- Setting internal and external CAS URLs
- Managing the Outlook Anywhere settings
- Blocking Outlook clients from connecting to Exchange
- Reporting on active OWA and RPC connections
- Controlling the ActiveSync device access
- Reporting on ActiveSync devices

Introduction

The Client Access Server (CAS) role was introduced in Exchange 2007 to provide a dedicated access point to various services, such as Outlook Web Access (OWA), ActiveSync, POP3, and IMAP4 to clients. However, all MAPI clients connected directly to the mailbox server role. The CAS role was extended even further in Exchange 2010 and included some new features, including a functionality that will change the architecture of every Exchange deployment. At that time, the connections to public folders were still made by MAPI clients to the mailbox server role; connections from these clients to Exchange 2010 mailboxes were handled by the CAS role.

In this latest release, with Exchange 2013, Microsoft has simplified the CAS role. The CAS role is now stateless, which means that it does not save any data. Its job is more or less to help the clients to find a route to connect to the mailbox or any required service, such as ActiveSync; in short, a proxy server. This major architectural change reminds me a bit of the Exchange 2003 frontend and backend concept. Bear in mind that the role is not called frontend, the role names are still CAS and Mailbox.
Some benefits with this new architecture are as follows:

- The Layer 7 load balancer is not required anymore, which means cheaper deployment and investments.
- Another benefit would be that the CAS servers are not required to be on the same installation build/version, even though they need to be on Exchange 2013. This means that upgrades can be done more easily, and this would also help for troubleshooting and verifying that a new build is working in the way it should.

The CAS role and the Exchange Management Shell cmdlets used to manage it provide plenty of opportunities for automating repetitive tasks, from PowerShell one-liners, scripts, and functions.

In this chapter, we'll take a look at how you can control the access to the CAS services in your environment, customize their settings, and generate usage reports using the Exchange Management Shell.

**Performing some basic steps**

To work with the code samples in this chapter, follow these steps to launch the Exchange Management Shell:

1. Log on to a workstation or server with the Exchange Management tools installed.
2. Open the Exchange Management Shell by navigating to **Start | All Programs | Exchange Server 2013**.
3. Click on the **Exchange Management Shell** shortcut.

Remember to start the Exchange Management Shell using **Run as administrator** to avoid permission problems.

In this chapter, you might notice that in the examples of cmdlets, I have used the back tick (`) character to break up long commands into multiple lines. The purpose of this is to make it easier to read. The back ticks are not required and should only be used if needed.

**Managing ActiveSync, OWA, POP3, and IMAP4 mailbox settings**

You can use the Exchange Management Shell to configure a user's ability to access services, such as ActiveSync, OWA, POP3, and IMAP4. You can also allow or disallow MAPI connectivity and the ability to connect to Exchange using Outlook Anywhere. In this recipe, you'll learn the techniques used to control these settings, whether it is done interactively through the shell or using an automated script.
To control access to CAS services for a mailbox, use the `Set-CASMailbox` cmdlet. Here's an example of how you might use this cmdlet:

```powershell
Set-CasMailbox -Identity 'Dave Smith' `-OWAEnabled $false `-ActiveSyncEnabled $false `-PopEnabled $false `-ImapEnabled $false
```

This command will disable OWA, ActiveSync, POP3, and IMAP4 for the mailbox that belongs to Dave Smith.

**How it works...**

When you create a mailbox, OWA, ActiveSync, POP3, IMAP4, and MAPI accesses are enabled by default. For most organizations, these default settings are acceptable, but if that is not the case for your environment, you can use the `Set-CASMailbox` cmdlet to enable or disable access to these services. This can be done for individual users as needed, or you can do this in bulk.

For example, let's say that all of the users in the `Sales` department should only be allowed to access Exchange internally through Outlook using MAPI, POP, and IMAP. We can use a simple pipeline command to make this change as follows:

```powershell
Get-Mailbox -Filter {Office -eq 'Sales'} | Set-CasMailbox -OWAEnabled $false `-ActiveSyncEnabled $false `-PopEnabled $true `-ImapEnabled $true
```

As you can see, we use the `Get-Mailbox` cmdlet and specify a filter that limits the results to users that have their `Office` attribute in Active Directory set to `Sales`. The results are then piped to the `Set-CASMailbox` cmdlet and access to the CAS services is modified for each mailbox. Notice that this time, we've used additional parameters to allow POP and IMAP access.
Alternatively, you may want to block MAPI access and only allow users in your organization to connect through OWA. In this case, use the following command:

```
Get-Mailbox -RecipientTypeDetails UserMailbox |
Set-CASMailbox -OWAEnabled $true ` -ActiveSyncEnabled $false ` -PopEnabled $false ` -ImapEnabled $false ` -MAPIBlockOutlookRpcHttp $true
```

This time, we use `Get-Mailbox` to retrieve all the mailboxes in the organization. We're using the `-RecipientTypeDetails` parameter to specify that we want to find user mailboxes and exclude other types, such as discovery and resource mailboxes. The results are piped to the `Set-CASMailbox` cmdlet and access to CAS services is configured with the required settings. You'll notice that this time, we've included the `-MAPIBlockOutlookRpcHttp` parameter and set its value to `$true` so that users will only be able to access Exchange through OWA.

**There's more...**

If you are planning on provisioning all of your new mailboxes through an automated script, you may want to configure these settings at the time of mailbox creation. Consider the following script named `New-MailboxScript.ps1`:

```powershell
param(
    $name,
    $password,
    $upn,
    $alias,
    $first,
    $last
)
$pass = ConvertTo-SecureString -AsPlainText $password -Force
$mailbox = New-Mailbox -UserPrincipalName $upn ` -Alias $alias ` -Name "$first $last" ` -Password $pass ` -FirstName $first ` -LastName $last
Set-CasMailbox -Identity $mailbox ` -OWAEnabled $false ` -ActiveSyncEnabled $false ` -PopEnabled $false ` -ImapEnabled $false
```
This script can be used to create a mailbox and configure the access to CAS services based on your requirements. If the script is saved in the root of the C: drive, the syntax would look like this:

```bash
[PS] C:\>.\New-MailboxScript.ps1 -first John -last Smith -alias jsmith -password P@ssw0rd01 -upn jsmith@contoso.com
```

There are basically two phases to the script. First, the mailbox for the user is created using the `New-Mailbox` cmdlet. In this example, the result of `New-Mailbox` is saved in the `$mailbox` variable, and the mailbox is created using the parameters provided by the user running the script. Once the mailbox is created, the `Set-CASMailbox` cmdlet is used to configure the access to CAS services, and it uses the `$mailbox` variable to identify the mailbox to modify when the command executes.

**See also**

- Adding, modifying, and removing mailboxes in Chapter 3, Managing Recipients

### Setting internal and external CAS URLs

Each CAS server has multiple virtual directories, some of which can only be modified through the Exchange Management Shell. Scripting the changes made to both the internal and external URLs can be a big time-saver, especially when deploying multiple servers. In this recipe, you will learn how to use the set of cmdlets that are needed to modify both the internal and external URLs for each CAS server virtual directory.

**How to do it...**

To change the external URL of the OWA virtual directory of a server named `cas1`, use the following command:

```powershell
Set-OwaVirtualDirectory -Identity 'CAS1\owa (Default Web Site)' -ExternalUrl https://mail.contoso.com/owa
```

After the change has been made, we can view the configuration using the `Get-OwaVirtualDirectory` cmdlet:

```powershell
[PS] C:\>Get-OwaVirtualDirectory -Server cas1 | fl ExternalUrl
ExternalUrl : https://mail.contoso.com/owa
```

Notice that if you change the URL for OWA, the ECP URL should be changed as well.
Managing Client Access

How it works...

Each CAS server hosts virtual directories in IIS that support OWA, Exchange Control Panel (ECP), ActiveSync, Offline Address Book (OAB), and Exchange Web Services (EWS). Each of these services has an associated cmdlet set that can be used to manage the settings of each virtual directory. One of the most common configuration changes made during the deployment process is modifying the internal and external URLs for each of these services. The required configuration varies greatly depending on a number of factors in your environment, especially in larger multisite environments.

The following cmdlets can be used to modify several settings for each virtual directory, including the values for the internal and external URLs:

- Set-ActiveSyncVirtualDirectory: This is used to configure the internal and external URL values of the Microsoft-Server-ActiveSync virtual directory. Use the InternalUrl and ExternalUrl parameters to change the values.
- Set-EcpVirtualDirectory: This is used to configure the internal and external URL values of the ECP virtual directory. Use the InternalUrl and ExternalUrl parameters to change the values.
- Set-OabVirtualDirectory: This is used to configure the internal and external URL values of the OAB virtual directory. Use the InternalUrl and ExternalUrl parameters to change the values.
- Set-OwaVirtualDirectory: This is used to configure the internal and external URL values of the OWA virtual directory. Use the InternalUrl and ExternalUrl parameters to change the values.
- Set-WebServicesVirtualDirectory: This is used to configure the internal and external URL values of the EWS virtual directory. Use the InternalUrl and ExternalUrl parameters to change the values.

When running each of these cmdlets, you need to identify the virtual directory in question. For example, when modifying the external URL of the ECP virtual directory, the command might look similar to this:

```
Set-EcpVirtualDirectory -Identity 'CAS1\ecp (Default Web Site)' -ExternalUrl https://mail.contoso.com/ecp
```

The syntax here is similar to the first example where we modified the OWA virtual directory; the only difference is that the cmdlet name and the ExternalUrl value have changed. Notice that the identity of the virtual directory is in the format of ServerName\VirtualDirectoryName (WebsiteName). The reason this needs to be done is because it's possible, but not very common, for a particular CAS server to be running more than one site in IIS, containing virtual directories for each of these CAS services.
If you are like most folks and have only the default website running in IIS, you can also take advantage of the pipeline if you forget the syntax needed to specify the identity of the virtual directory. For example, run the following command:

```
Get-EcpVirtualDirectory -Server cas1 | 
Set-EcpVirtualDirectory -ExternalUrl https://mail.contoso.com/ecp
```

The preceding pipeline command makes the same change, as shown previously. This time, we’re using the `Get-EcpVirtualDirectory` cmdlet, with the `-Server` parameter to identify the CAS server. We then pipe the resulting object to the `Set-EcpVirtualDirectory` cmdlet that makes the change to the `ExternalUrl` value.

It is important to configure the URLs correctly, since this have an impact on end users. If they are not correctly configured and use a name that is not included in the certificate, then the end users will be prompted when they are trying to reach the services, or even worse, when the services are broken and not reachable.

**There's more...**

If you allow the access to Exchange through Outlook Anywhere, you'll need to configure the external URLs that will be handed to Outlook clients for services, such as ECP, OAB, and EWS. These URLs may need to point to an FQDN that resolves to a load balancer **Virtual IP Address (VIP)** or to your reverse proxy infrastructure, such as TMG or third-party solutions.

In addition, you'll probably want to configure your internal URLs to point to an FQDN that resolves to your internal load balancer VIP. In this situation, you want to make sure you do not modify the internal URL for both the OWA and ECP virtual directories in non-Internet-facing sites. This is because OWA and ECP connections from the Internet-facing CAS server will be proxied to the servers in the non-Internet-facing sites, and if the internal FQDN of the CAS server is not set on each these virtual directories, Kerberos authentication will fail and the user will not be able to access their mailbox.

Finally, for load balanced CAS servers, you'll want to configure the AutoDiscover internal URL so that it also points to an FQDN that resolves to your load balancer VIP. The syntax for this would look like the following:

```
Set-ClientAccessServer -Identity cas1 ` 
-AutoDiscoverServiceInternalUri ` 
https://mail.contoso.com/Autodiscover/Autodiscover.xml
```

Of course, you'll need to make all the changes to internal and external URLs on all the CAS servers.
Managing Client Access

Command syntax for the remaining virtual directories

We've already looked at the syntax for modifying both OWA and ECP and internal and external URLs; now, let's take a look at how we can do this for the remaining virtual directories. In these examples, we'll configure the external URL value using the -ExternalUrl parameter. If you need to modify the internal URL, simply use the -InternalUrl parameter.

To configure the external URL of the OAB, use the following syntax:

```
Set-OABVirtualDirectory -Identity "cas1\oab (Default Web Site)" `- ` `-ExternalUrl https://mail.contoso.com/oab
```

To configure the external URL of the ActiveSync virtual directory, use the following syntax:

```
```

To configure the EWS virtual directory, use the following syntax:

```
```

In each example, we're setting the value on the cas1 server. When running these commands or using them in a script, replace the server name with the name of the appropriate CAS server.

Taking it one step further

In Chapter 1, PowerShell Key Concepts, we went through an introduction about Desired State Configuration (DSC), and we can now take the advantage of using the DSC server to push out a configuration to the Exchange server(s). We will utilize a module called xExchange module to configure the OWA URLs. Since we have used the pull mode before, notice the -Force parameter that is used.

The following example is a modified example from the xExchange module:

```powershell
$configData = @{
    AllNodes = @(
        @{
            NodeName = 'tlex01.testlabs.se'
            CertificateFile = 'C:\publickey.cer'
            Thumbprint = 'C72FA4F17DBE0C5F88522AA49DF86EB410B23A71'
        }
    )
    @
    NodeName = 'tlex01.testlabs.se'
```
Chapter 7

CASID = 'Site1CAS'

Site1CAS = @{
    InternalNLBFqdn = 'mail.contoso.com'
    ExternalNLBFqdn = 'mail.contoso.com'
    AutoDiscoverSiteScope = 'Default-First-Site-Name'
    OABsToDistribute = 'Default Offline Address Book'
}

Configuration ConfigureVirtualDirectories
{
    param
    (
        [PSCredential]$ShellCreds
    )
    Import-DscResource -Module xExchange
    Node $AllNodes.NodeName
    {
        $casSettings = $ConfigurationData[$Node.CASId]
        # Thumbprint of the certificate used to decrypt credentials on the target node
        LocalConfigurationManager
        {
            CertificateId = $Node.Thumbprint
        }
        xExchOwaVirtualDirectory OWAVdir
        {
            Identity = "\$(Node.NodeName)\owa (Default Web Site)"
            Credential = $ShellCreds
            BasicAuthentication = $true
            ExternalAuthenticationMethods = 'Fba'
            ExternalUrl = 'https://$casSettings.ExternalNLBFqdn/owa'
            FormsAuthentication = $true
            InternalUrl = 'https://$casSettings.InternalNLBFqdn/owa'
            WindowsAuthentication = $false
            AllowServiceRestart = $true
        }
    }
}

if ($ShellCreds -eq $null)
{
}
$ShellCreds = Get-Credential -Message 'Enter credentials for establishing Remote Powershell sessions to Exchange'

ConfigureVirtualDirectories -ConfigurationData $configData -ShellCreds $ShellCreds
Set-DscLocalConfigurationManager -Path .\ConfigureVirtualDirectories -Verbose
Start-DscConfiguration -Path .\ConfigureVirtualDirectories -Verbose -Wait -Force

The preceding example is using a trusted certificate for encrypting and decrypting the password for creating a remote PowerShell session. When running this script, a similar result will be shown as follows:

![Screenshot of PowerShell output]

The three final rows in the example are explained as follows:

- **ConfigureVirtualDirectories**: This compiles the script
- **Set-DscLocalConfigurationManager**: This sets up the **Local Configuration Manager (LCM)** on the Exchange server to decrypt the credentials
- **Start-DscConfiguration**: This pushes the configuration to the Exchange server

The example using the Exchange module is just to show you an overview of how it can be used. The module can be used to configure all URLs, databases, Outlook Anywhere, and many more.
The module is updated over time at a pretty fast pace; to stay updated, I would recommend that you subscribe to the PowerShell team blog using RSS at http://blogs.msdn.com/b/powershell/.

See also

- Generating a certificate request in Chapter 10, Exchange Security
- Installing certificates and enabling services in Chapter 10, Exchange Security
- Importing certificates on multiple Exchange servers in Chapter 10, Exchange Security

Managing the Outlook Anywhere settings

With the new CAS architecture in place, Outlook Anywhere needs to be configured if Outlook is going to be used to access mailboxes and features. This feature allows Outlook clients to connect to Exchange through RPCs encapsulated into an HTTPS connection. This allows easy internal and external access to Exchange from Outlook, as there is no need to open RPC ports on firewalls. In this recipe, we’ll take a look at how you can use the Exchange Management Shell to manage Outlook Anywhere settings.

How to do it...

By default, the Outlook Anywhere feature is enabled, but it needs to be configured with the correct hostname and authentication values. This is done using the `Set-OutlookAnywhere` cmdlet. See the following example:

```
Set-OutlookAnywhere -Identity 'CAS1\Rpc (Default Web Site)' ` `-ExternalHostname mail.contoso.com ` `-ExternalClientRequireSsl $true ` `-InternalHostname mail.contoso.com ` `-InternalClientRequireSsl $true ` `-ExternalClientAuthenticationMethod Basic ` `-InternalClientAuthenticationMethod Ntlm ` `-SSLOffloading $false`
```

In the preceding example, Outlook Anywhere is configured on the CAS1 server.
Managing Client Access

How it works...

Before enabling Outlook Anywhere, there are two prerequisites that need to be met. First, you need to ensure that your CAS server has a valid SSL certificate installed from a **certificate authority (CA)** that is trusted by your client machines. Exchange installs a self-signed certificate by default, but this will not be trusted by client workstations.

In addition, you'll need to make sure that the Microsoft Windows RPC over HTTP Proxy component is installed on the server. This is typically done before the installation of Exchange when all of the operating system prerequisites are installed.

When running the `Set-OutlookAnywhere` cmdlet, you can see that we specified the `ExternalHostname` property. This will be the FQDN that Outlook clients will use to connect to Exchange. You'll need to make sure that you have a DNS record created for this FQDN that resolves to your CAS server or to your reverse proxy infrastructure, such as ISA or TMG. A much appreciated feature in Exchange 2013 is that the `InternalHostname` property can be configured. This means that the `InternalUrl` parameter can be something else, which most likely is pointed to the internal load balancer VIP to spread the load.

When specifying a value for the `ExternalClientAuthenticationMethod` and `InternalClientAuthenticationMethod` parameters, you'll want to use either **Basic** or **NTLM**. This setting determines how users authenticate to Outlook Anywhere. When using the **Basic** authentication, the user's password is sent to the server in plain text, but the connection is secured by SSL. If you have workstations, which are not domain-joined, that will connect to Exchange through Outlook Anywhere, you'll need to use the **Basic** authentication.

If only domain-joined clients connect to Outlook Anywhere, such as roaming users with laptops that connect from home; using the **NTLM** authentication is a much more secure option for the `ClientAuthenticationMethod` property. When using **NTLM**, a user's password is not sent to the server; instead, **NTLM** sends a hashed value of the user's credentials to the server. Another benefit of using **NTLM** is that Outlook clients will not be prompted for their credentials when connecting to Outlook Anywhere. Keep in mind that if you are publishing Outlook Anywhere with a reverse proxy solution, such as TMG, you'll need to use **Kerberos Constrained Delegation (KCD)**, which allows the TMG server to request a Kerberos service ticket from Active Directory on behalf of the user. Also, remember that the **NTLM** authentication may not work correctly with some firewalls; check with your firewall manufacturer's documentation for details.

Finally, **SSL Offloading** allows the CAS server to offload the encryption and decryption of the SSL connections to a third-party device. Unless you have an SSL offloading solution in place, set the `-SSLOffloading` parameter to `$false`. However, this parameter is not required if you don't use SSL offloading.
There's more...

In addition to enabling Outlook Anywhere from the shell, we can also perform some other routine tasks. For example, to view the Outlook Anywhere configuration, use the `Get-OutlookAnywhere` cmdlet, as follows:

```
[PS] C:\>Get-OutlookAnywhere | fl ServerName,ExternalHostname, InternalHostname
ServerName : CAS1
ExternalHostname : mail.contoso.com
InternalHostname : mail.contoso.com
```

The `Get-OutlookAnywhere` cmdlet will return the configuration information for servers that have the Outlook Anywhere feature enabled.

If at any time you need to change the authentication method or external host name for Outlook Anywhere, you can use the `Set-OutlookAnywhere` cmdlet as follows:

```
Set-OutlookAnywhere -Identity 'CAS1\Rpc (Default Web Site)' - `
-ExternalHostname 'outlook.contoso.com'
```

Notice that the identity of the server needs to be specified in the format of `ServerName\VirtualDirectoryName (WebsiteName)`.

See also

- Generating a certificate request in Chapter 10, Exchange Security
- Installing certificates and enabling services in Chapter 10, Exchange Security
- Importing certificates on multiple Exchange servers in Chapter 10, Exchange Security

Blocking Outlook clients from connecting to Exchange

Exchange gives you plenty of options to block clients from connecting to mailboxes, depending on the version of the Outlook client and the method used to access the mailbox. In this recipe, you'll learn how to configure these options using the Exchange Management Shell.
How to do it...

Let's take a look at how to block Outlook clients from connecting to Exchange using the following steps:

1. The `Set-CASMailbox` cmdlet can be used to block MAPI access to mailboxes based on several factors. For example, we can prevent an individual user from using Outlook to connect using Outlook Anywhere:
   ```
   Set-CASMailbox -Identity dsmith -MAPIBlockOutlookRpcHttp $true
   ```

2. In addition, we can also prevent a user whose Outlook is not configured in cached mode from connecting to their mailbox using the following command:
   ```
   Set-CASMailbox -Identity dsmith -MAPIBlockOutlookNonCachedMode $true
   ```
   In both the cases, the user can still access their mailbox using OWA; as long as the `OWAEnabled` property is set to `$true`.

3. You can also block users from connecting to clients based on their Outlook version. The following command will block all versions of Outlook earlier than 2007 for every mailbox in the organization:
   ```
   Get-CASMailbox -Resultsize Unlimited | Set-CASMailbox -MAPIBlockOutlookVersions '-11.9.9'
   ```

4. To find all the mailboxes in an organization that have `MAPIBlockOutlookVersions` defined, run the following command:
   ```
   Get-CASMailbox -ResultSize Unlimited | Where-Object{$_[MAPIBlockOutlookVersions]}
   ```

5. To remove the restriction on a single mailbox, use the following command:
   ```
   Set-CASMailbox dsmith -MAPIBlockOutlookVersions $null
   ```

6. To remove the restriction on the entire organization, use the following command:
   ```
   Get-CASMailbox -ResultSize Unlimited | Set-CASMailbox -MAPIBlockOutlookVersions $null
   ```

How it works...

The `Set-CASMailbox` cmdlet allows you to configure which protocols and services a particular mailbox user can access. To determine the existing settings, we can use the `Get-CASMailbox` cmdlet. For instance, if you need to retrieve all the users that have been blocked from connecting to their mailboxes in non-cached mode, use the following command:

```
Get-CASMailbox | Where-Object{$_[MAPIBlockOutlookNonCachedMode]}
```
To find all the mailboxes blocked from using Outlook Anywhere, the command is almost identical, just reference the correct property name:

```
Get-CASMailbox | Where-Object{$_._MAPIBlockOutlookRpcHttp}
```

In both the examples, we pipe the `Get-CASMailbox` cmdlet to the `Where-Object` cmdlet. Inside the filter, we will check to see whether the property values evaluate as `$true`. If that is the case, the command will return a list of users who have the corresponding setting enabled.

As always, we can use pipelining to enable or disable these settings for multiple users in a single command. Let’s say that we want to block all of the users in the `Sales` OU from using Outlook Anywhere:

```
Get-CASMailbox -OrganizationalUnit contoso.com/Sales | Set-CASMailbox -MAPIBlockOutlookRpcHttp $true
```

To remove this restriction, use the same command, but this time, set the parameter value to `$false`:

```
Get-CASMailbox -OrganizationalUnit contoso.com/Sales | Set-CASMailbox -MAPIBlockOutlookRpcHttp $false
```

In both the cases, the `Get-CASMailbox` cmdlet retrieves every mailbox from the `Sales` OU and pipes the object’s output by the command to the `Set-CASMailbox` cmdlet that then makes the change.

As we saw earlier, Outlook client versions can be blocked on a per-mailbox basis using the `Set-CASMailbox` cmdlet. This is done by specifying the client version using the `MAPIBlockOutlookVersions` parameter.

In Exchange 2007, you could check the `ClientVersion` property returned by the `Get-LogonStatistics` cmdlet to determine the version numbers used by Outlook clients in the organization. In Exchange 2013, the `ClientVersion` property will be reported based on the CAS server, making the connection to the mailbox server, not the actual Outlook client. If you need to determine the specific client versions in your environment, you can use the `Help | About` screen in Outlook to determine the exact version number.

A version number is made up of a major, minor, and build number. Here are a few version numbers for some commonly used versions of Outlook:

- Outlook 2003 SP2: 11.6568.6568
- Outlook 2007 RTM: 12.4518.1014
- Outlook 2010 RTM: 14.0.4760.1000
- Outlook 2013 RTM: 15.0.4420.1017
The major build numbers are consistent across the entire Office Suite, and never change. For example, for Office 2003, the build number is 11, for Office 2007, the build number is 12, for Office 2010, the build number is 14, and so on.

The minor and build numbers will change, depending on the hotfixes and service packs deployed to the clients. Therefore, the -MAPIBlockOutlookVersions parameter will accept a range of values that will allow you to be very specific about which versions should be blocked. You can even specify multiple version ranges and separate each one by a semicolon.

For example, the following command can be used to block access to Exchange for all versions of Outlook below 2007 and 2010:

```powershell
Set-CASMailbox dsmith -MAPIBlockOutlookVersions '-5.9.9;7.0.0-11.9.9'
```

As you can see here, we've specified two values. The first value indicates that any client version below 5.9.9 will be unable to connect to this mailbox. The second value specifies a range from 7 to 11.9.9, which effectively blocks all access to any client versioned lower than 12.x.x, except for those versioned at 6.x.x. This allows only Outlook 2007 and 2010 clients to connect to this mailbox. It also allows Exchange server MAPI connections from other servers, identified using 6.x.x version numbers.

Keep in mind that when you are making these changes, they will not take effect right away. If you want to force this change so that it is effective immediately, restart the RPC CAS service on the Mailbox server used to access the mailbox. Make sure to do this outside the production hours, as it will knock every user connected to that CAS server offline.

**There's more...**

In addition to blocking Outlook versions at the mailbox level, we can also block them at the server level. Since the MAPI client endpoint is now at the Mailbox role again, we can use the Set-RPCClientAccess cmdlet to accomplish this:

```powershell
Set-RpcClientAccess -Server cas1 `-BlockedClientVersions '-5.9.9;7.0.0-13.9.9'
```

You can see here that we used the BlockedClientVersions parameter to define the client versions that should be blocked, and it works in exactly the same way as it does when using the Set-CASMailbox cmdlet. In this example, all client versions below Outlook 2010, with the exception of client versions 6.x.x, will be blocked at the CAS server level. Notice that the server name has been specified with this command, and you'll need to run it against each Mailbox server that should block specific Outlook versions.
Reporting on active OWA and RPC connections

One of the nice things about using PowerShell to manage Exchange is that you have a great deal of flexibility when it comes to solving problems. When the Exchange Management Shell does not provide a cmdlet that specifically meets your needs, you can often tap into other resources, which are accessible through PowerShell. This recipe provides a great example for this. In this section, we'll use PowerShell to query the performance counter data to determine the number of active OWA and HTTP/RPC (Outlook Anywhere) connections on one or more Mailbox servers.

How to do it...

Let's see how to report on active OWA and RPC connections using the following steps:

1. To determine the number of users currently logged into OWA on a Mailbox server, use the following command syntax:

   ```powershell
   Get-Counter -Counter '\tlex01\MSExchange OWA\Current Users'
   ```

   This retrieves the total number of users logged into OWA on the TLEX01 server. The output of this command will look similar to the following screenshot:

   ![Screenshot of PowerShell output](image-url)

   In the preceding screenshot, we can see that two users are currently logged on to OWA.

2. To find the total number of HTTP/RPC (Outlook Anywhere) connections, we simply need to use another performance counter:

   ```powershell
   Get-Counter '\tlex01\MSExchange RpcClientAccess\User Count'
   ```
Managing Client Access

Similar to the previous example, the total number of HTTP/RPC connects will be reported, as shown in the following screenshot:

![Screenshot](image)

**How it works...**

The `Get-Counter` cmdlet can be used to retrieve the performance counter data from both local and remote machines. In the previous example, we collected the total number of current OWA users using the `\MSExchange OWA\Current Users` counter and the total number of HTTP/RPC connections using the `MSExchange RpcClientAccess\User Count` counter on the MBX1 server.

In both these examples, we've specified the computer name in the counter name assigned to the `-Counter` parameter. Another way to gather the performance counter data from a remote computer is to use the `-ComputerName` parameter, as shown in the following command:

```powershell
Get-Counter 'MSExchange OWA\Current Unique Users' -ComputerName tlex01,tlex02
```

Notice that in the alternate syntax used previously, we've removed the computer name from the counter name, and we have assigned a list of server names using the `-ComputerName` parameter. This is a quick way to check the number of connections on multiple computers.

There are many Exchange-related performance counters on each Exchange server. You can also use the `Get-Counter` cmdlet to discover these counters:

```powershell
Get-Counter -ListSet *owa* -ComputerName cas1 | Select-Object -expand paths
```

This will do a wildcard search and return a list of counters on the specified server that have the letters `owa` in their name. You can use this syntax to quickly find the counter names that can be used with the `Get-Counter` cmdlet.
There's more...

To create more advanced and customizable reports, we can create a PowerShell function that returns a custom object with only the information we're interested in. Add the following function to your shell session:

```powershell
function Get-ActiveUsers {
    [CmdletBinding()]
    param(
        [Parameter(Position=0, ValueFromPipelineByPropertyName=$true, Mandatory=$true)]
        [string[]] $Name
    )
    process {
        $Name | %{
            $RPC = Get-Counter "\MSExchange RpcClientAccess\User Count" -ComputerName $_
            $OWA = Get-Counter "\MSExchange OWA\Current Unique Users" -ComputerName $_
            New-Object PSObject -Property @{
                Server = $_
                'RPC Client Access' = $RPC.CounterSamples[0].CookedValue
                'Outlook Web App' = $OWA.CounterSamples[0].CookedValue
            }
        }
    }
}
```

You can call the function and provide one or more Mailbox server names that you'd like to generate the report for, as shown in the following screenshot:

![Screenshot of PowerShell output](image-url)
If you look closely at the code in the function, you’ll notice that we’ve added some attributes to the $Name parameter. As you can see, in addition to being a mandatory parameter, it also accepts its value from the pipeline by a property name. This means that, instead of calling the function and providing a list of server names, we can leverage the objects that are returned by the Get-MailboxServer cmdlet to quickly generate a report using a pipeline command:

You can continue to pipe this command down to the Export-CSV or ConvertTo-Html cmdlet to generate an external report file that can be viewed outside the shell.

See also

- Understanding the pipeline in Chapter 1, PowerShell Key Concepts

Controlling ActiveSync device access

With the increase of smartphones being deployed, and the fact that ActiveSync can now be used pretty much on all mobile devices, Exchange 2010 introduced new functions that allowed you to control which devices are able to connect to your server. Using device access rules, you can define the specific devices or device types that can form an ActiveSync partnership with an Exchange server. This recipe will explore the options that can be used to allow, block, or quarantine ActiveSync devices using the Exchange Management Shell for Exchange 2013.

How to do it...

By default, there is an organization-wide configuration setting that will allow any ActiveSync device to connect to Exchange. You can modify this so that all devices are initially quarantined and need to be approved by an administrator before they can gain access. To implement this, use the following steps:

1. First run the following command:

```
Set-ActiveSyncOrganizationSettings -DefaultAccessLevel Quarantine -AdminMailRecipients administrator@contoso.com
```
2. After the previous command completes, all the devices that attempt to form an ActiveSync device partnership will be quarantined. When a device is quarantined, the address provided by the –AdminMailRecipients parameter will be notified via e-mail. The user will also receive a message on their mobile device informing them that access needs to be granted by an administrator before they'll be able to access any content. Based on the information in the e-mail message, the administrator can choose to enable the device using the Set-CASMailbox cmdlet:

```
Set-CASMailbox -Identity dsmith -ActiveSyncAllowedDeviceIDs `BAD73E6E02156460E800185977C03182`
```

3. Once the command has been run, the user will be able to connect to your server.

**How it works...**

In Exchange 2013, you can manage the devices in the Exchange Control Panel (ECP), and, of course, the cmdlets can still be used if you want to do this work from the shell.

When configuring the ActiveSync organization settings, you have the option of adding a custom message that will be sent to the user when they receive the e-mail notification, explaining that their device has been quarantined. Use the -UserMailInsert parameter when running the Set-ActiveSyncOrganizationSettings cmdlet to configure this setting:

```
Set-ActiveSyncOrganizationSettings -DefaultAccessLevel Quarantine -AdminMailRecipients helpdesk@contoso.com -UserMailInsert 'Call the Help Desk for immediate assistance'
```

In addition to the administrative e-mail notifications, you can find all the devices that are currently in a quarantined state using the Get-ActiveSyncDevice cmdlet:

```
Get-MobileDevice | Where {$_.DeviceAccessState -eq 'Quarantined'} | Select UserDisplayName,DeviceAccessState,DeviceID
```

The output of the preceding command will be similar to the following screenshot:
This command retrieves ActiveSync devices and is filtered on the DeviceAccessState property. The output will provide the username, device access state, and the DeviceID that can be used to allow access using the Set-CASMailbox cmdlet.

**There's more...**

Manual approval of ActiveSync devices may not be something you want to take on as an administrative task. An alternative to this is to use device access rules. For instance, let's say that you want to block all ActiveSync devices that are not iPhone devices. You could set the DefaultAccessLevel property for the organization to Block and create a device access rule allowing only those devices:

```
New-ActiveSyncDeviceAccessRule -Characteristic DeviceType `-QueryString iPhone -AccessLevel Allow
```

You can create multiple access rules for different types of devices if needed. To determine the device type, you can use the Get-MobileDevice cmdlet. The property values of DeviceModel, DeviceType, DeviceOS, DeviceUserAgent, and XMSWLHeader can be used with the -QueryString parameter to define the device type when creating a device access rule.

**See also**

- Reporting on ActiveSync devices
- Managing ActiveSync, OWA, POP3, and IMAP4 mailbox settings

**Reporting on ActiveSync devices**

The Exchange Management Shell provides several cmdlets that can be used for generating reports. We can obtain information about users and their devices, and we can also generate reports based on the end user activity and server usage. In this recipe, we'll take a look at how we can use these cmdlets to generate multiple ActiveSync reports from the Exchange Management Shell.

**How to do it...**

Let's see how to generate reports on ActiveSync devices using the following steps:

1. To generate a report for an individual user's device synchronization status, use the following command:

   ```
   Get-MobileDeviceStatistics -Mailbox nate
   ```
2. This cmdlet will output a lot of information, some of which may not be very interesting. You can limit the data returned by selecting only the properties that provide useful information:

```
Get-MobileDeviceStatistics -Mailbox nate |
select LastSuccessSync, Status, DevicePhoneNumber, DeviceType
```

The output of the previous command will look similar to the following screenshot:

![Screenshot of output](image)

3. To export this information, you can pipe the command even further to the `Export-CSV` cmdlet:

```
Get-MobileDeviceStatistics -Mailbox nate |
select LastSuccessSync, Status, DevicePhoneNumber, DeviceType | Export-CSV -Path c:\report.csv -NoType
```

**How it works...**

Using the `Get-MobileDeviceStatistics` cmdlet, we can retrieve the mobile phones that are configured to synchronize with a particular user's mailbox. As you can see from the previous examples, it's quite easy to generate a report for an individual user. This cmdlet requires that you either specify the identity of the ActiveSync device or the mailbox of the owner. In order to generate reports based on statistics for multiple devices, we have a couple of options.

First, we can use the `Get-MobileDevice` cmdlet to retrieve a list of allowed devices and then pipe the results to the `Get-MobileDeviceStatistics` cmdlet:

```
$dev = Get-MobileDevice | ?{$_._DeviceAccessState -eq 'Allowed'}
$dev | ForEach-Object {
    $mailbox = $_._UserDisplayName
    $stats = Get-MobileDeviceStatistics -Identity $_
    $stats | Select-Object @{n="Mailbox";e={$mailbox}}, LastSuccessSync, Status,
}```
Managing Client Access

```powershell
DevicePhoneNumber,
DeviceType
}

The preceding code retrieves all the ActiveSync devices with the *Allowed* access state. We loop through each device, retrieve the device statistics for each one, and return several properties that provide details about the user and the status of their device. Notice that, in the example, we're using a calculated property to return the mailbox name, since that information is not included in the output of the `Get-MobileDeviceStatistics` cmdlet.

The other method for obtaining this information is using the `Get-CASMailbox` cmdlet to find all the users with an ActiveSync device partnership, and then sending those objects down the pipeline to the `Get-MobileDeviceStatistics` cmdlet:

```powershell
$mbx = Get-CASMailbox | ?{$_.HasActiveSyncDevicePartnership}
$mbx | ForEach-Object {
    $mailbox = $_.Name
    $stats = Get-MobileDeviceStatistics -Mailbox $mailbox
    $stats | Select-Object @{n="Mailbox";e={$mailbox}},LastSuccessSync,
    Status,
    DevicePhoneNumber,
    DeviceType
}
```

Similar to the previous example, we loop through each mailbox, retrieve the ActiveSync statistics, and then return the same properties as before. This version is considerably slower, since it has to first check every mailbox to determine whether a device partnership exists, but if you need specific filtering capabilities based on the properties returned by the `Get-CASMailbox` cmdlet, this may be a useful method. This will also filter out system mailboxes.

**There's more...**

The Exchange Management Shell also provides the `Export-ActiveSyncLog` cmdlet that can be used to generate reports based on ActiveSync usage. The cmdlet generates reports based on IIS log files and then outputs the following six separate CSV files that contain detailed information about the usage of ActiveSync devices:

- **Users.csv**: This provides details on ActiveSync usage for each user that includes the number of sent and received items
- **UserAgents.csv**: This provides details on the various user agents used by devices to access Exchange
Chapter 7

- **StatusCodes.csv**: This provides the HTTP response code issued to ActiveSync clients
- **Servers.csv**: This provides details on the server usage, including total number of bytes sent and received
- **PolicyCompliance.csv**: This provides details on ActiveSync device compliance, such as the total number of compliant, noncompliant, and partially compliant devices
- **Hourly.csv**: This provides an hourly breakdown of the device synchronization activity

The cmdlet supports a number of parameters that can be used to generate reports. For example, the following command generates reports for ActiveSync activity taking place on March 24, 2015:

```
Export-ActiveSyncLog
-Filename C:\inetpub\logs\LogFiles\W3SVC1\u_ex150324.log
-OutputPath c:\report
```

When running this command, make sure that the directory specified for the output path has already been created. The given command generates the six CSV files discussed previously in the `c:\report` folder.

To generate reports for multiple log files, you'll need to do a little extra work. For example, run the following command:

```
$path = "C:\inetpub\logs\LogFiles\W3SVC1"
Get-ChildItem -Path $path -Filter u_ex1503*.log | %{
    Export-ActiveSyncLog -Filename $_.fullname
    -OutputPrefix $_.basename
    -OutputPath c:\report
}
```

Here, we're using the `Get-ChildItem` cmdlet to retrieve a list of log files from March 2015. Each time we run the `Export-ActiveSyncLog` cmdlet for a log file, a new set of six CSV reports will be generated. Since we can only define one `OutputPath`, we use the log file base name as a prefix for each CSV report file generated. After the cmdlet has been run, six CSV reports for each day of the month will be located in the `c:\report` directory. You can read these reports in the shell using the `Import-Csv` cmdlet, or open them in Excel or Notepad for review.
See also

- Creating custom objects in Chapter 1, PowerShell Key Concepts
- Controlling ActiveSync device access
- Managing ActiveSync, OWA, POP3, and IMAP4 mailbox settings
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.