13

Integration

In this chapter, we will cover:

* OAuth configuration
* Archiving and conversation history
* Unified Contact Store
* Integrate Skype for Business with OWA
* Using high-resolution photos
* Office Online Server integration
* Exchange Hybrid validation

# Introduction

In this chapter, we will focus on how Exchange 2016 integrates with Skype for Business, Office Online Server and Exchange Online.

To give you an idea of the existing infrastructure setup when reading this chapter, it’s based on one Exchange server, one Skype for Business server and one Office Online server. However, all the recipes also apply if you have a large setup of servers.

Some examples of what you will learn in this chapter include how to use high-resolution photos, archiving, Skype for Business integration in OWA and configuring Unified Contact Store. We will also take a look at how to integrate with Office Online Server and Exchange Online.

When reading this chapter, it’s recommended to start reading the OAuth configuration part and then continue with selecting which recipe you find most interesting.

This chapter also assumes that you are familiar with Skype for Business, Office Online Server and Exchange Online since we are not looking into these for specific details. They are included to illustrate the integration that can be done.

## Performing some basic steps

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

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.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 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. |  |

# OAuth configuration

This recipe should be treated as a prerequisite for the following recipes that are based on Exchange 2016 and Skype for Business 2015 integration. OAuth is being used for all type of integration between Exchange 2016 and Skype for Business, which we will cover in this recipe. The only recipe that doesn’t require OAuth for it’s integration is the High-Resolution photos.

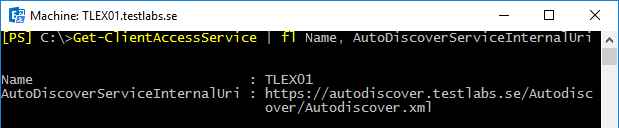
## How to do it...

Let’s quickly start collecting the information we will need later in this chapter and configure OAuth in our Exchange and Skype for Business solution. Beware in the following cmdlet examples, it’s a mix of cmdlet running in both Exchange Management Shell (EMS) and Skype for Business Server Management Shell (SFBMS).

1. To start with, we have to retrieve the Autodiscover URI since this is needed further down the road. This is done from the EMS and following example:

Get-ClientAccessService | fl Name,` AutoDiscoverServiceInternalUri

The output would look similar to the screenshot below:



7081EN\_13\_01

1. Before we start to configure the OAuth, we want to make sure it’s not already configured with any value. If the value of ExchangeAutodiscoverUrl is blank, it means that it’s not currently configured. This is done by running the cmdlet from SFBMS:

Get-CsOAuthConfiguration

1. Our next step is to configure OAuth in our Skype for Business 2015 solution. This is done by using the output (AutoDiscoverServiceInternalUri) from the first step and replace xml with svc. This should be done in the SFBMS and the cmdlet should look similar to the example below:

Set-CsOAuthConfiguration -Identity Global -ExchangeAutodiscoverUrl “https://autodiscover.testlabs.se/autodiscover/autodiscover.svc”

1. The final step in the OAuth configuration is to validate that the certificate is in place, it is a requirement for having the server-to-server communication to work. This is simply done by running one of the following cmdlets:

Get-CsCertificate

If you want to be even more specific, you can look for the OAuth certificate by running:

Get-CsCertificate -Type OAuthTokenIssuer

## How it works...

The integration between Exchange and Skype for Business is based on OAuth and server-to-server communication, which is using certificate(s). The requirement when it comes to the certificate on the Skype for Business server is that the same OAuth certificate needs to be configured on all of your Front-End servers and the certificate needs to be at least 2048 bits. This will be taken care of by the Skype for Business Central Management Store, the certificate will get replicated to all Front-End servers in the organization.

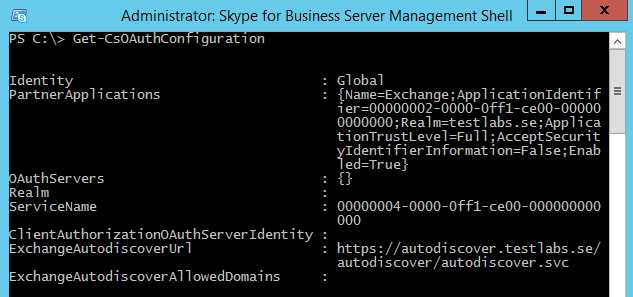
We started by collecting the Autodiscover information by running this cmdlet:   
Get- ClientAccessService | fl Name, AutoDiscoverServiceInternalUri.

The outcome is shown in the picture (link to 7081EN\_13\_01), in the environment we are using, it gave us the following URI “https://autodiscover.testlabs.se/autodiscover/autodiscover.xml", then replacing the .xml with .svc and using that as input when configuring OAuth.

Before we configure OAuth, we want to make sure that we are not overwriting it by mistake. This is as simple as running the cmdlet Get-CsOAuthConfiguration, then look for the attribute and value of ExchangeAutodiscoverUrl. If it doesn’t contain any value it means that it’s not currently configured. Then we can continue to configure OAuth by running the following one-liner:

Set-CsOAuthConfiguration -Identity Global –ExchangeAutodiscoverUrl ` "https://autodiscover.testlabs.se/autodiscover/autodiscover.svc"

With the OAuth configuration in place it should look similar to the screenshot below:



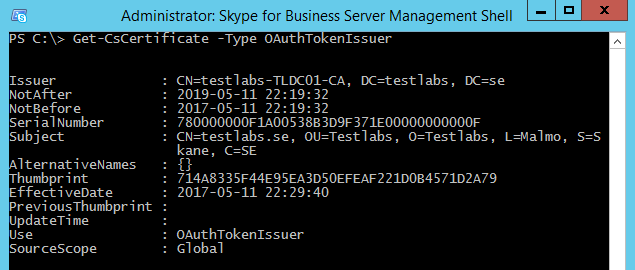
7081EN\_13\_02

A prerequisite for having a fully working server-to-server connection is to have an OAuth certificate installed and assigned on our Skype for Business server together with a working certificate (preferably a bought 3rd party certificate) on the Exchange server.

We need to validate that the OAuth certificate is in place by running the following cmdlet

Get-CsCertificate -Type OAuthTokenIssuer

If the certificate is in place the result from this cmdlet should look similar to the screenshot below:



7081EN\_13\_03

When running the cmdlet above, and if it wouldn’t give any result it would prompt with a warning stating that a certificate of the type OAuthTokenIssuer is missing.

If we are missing the OAuth certificate we should request a certificate from our Internal PKI running on the domain controller. This can of course be done by using PowerShell as well by running two sets of cmdlets, one for requesting a certificate directly from the Internal PKI and the second one for assigning the certificate we just requested.

Request-CsCertificate -New -Type OAuthTokenIssuer -CA "testlabs.se\testlabs-TLDC01-CA" -FriendlyName "Skype for Business Oauth certificate" -KeySize 2048 -PrivateKeyExportable $True -Country "SE" -State "Skane" -City "Malmo" -Organization "Testlabs" -OU "Testlabs"

Set-CsCertificate -Identity Global -Thumbprint 205E093F4CF20FA57874F98B09F9B12EBDDF8D98 -Type OAuthTokenIssuer -EffectiveDate "2017-05-11 22:15:02"

## See also

* Archiving and conversation history
* Integrate Skype for Business with OWA

Unified Contact Store

# Archiving and conversation history

In this recipe, we are going to configure and look into the integration of archiving for Skype for Business and Exchange 2016. If you didn’t read the previous recipe, I would like to recommend that you do. Just like all integration between Exchange server and Skype for Business, OAuth is being used for the communication.

It’s a great way of combining Exchange and Skype for Business to get additional value from both of them and also for simplifying to have all information located in the mailbox instead of spreading it out between different systems. This would be a driver when it comes to the in-place hold functionality in Exchange which are covered in chapter 10.

We will cover how to configure archiving for Skype for Business, for example the conversation history from instant messaging and from conferences. These will be logged and saved into the user’s mailbox.

## How to do it...

The cmdlets in this recipe are all being executed from the Skype for Business Server Management Shell (SFBMS).

1. We’ll start by enabling and configuring the archiving for both instant messaging and conferences, this is done by running the cmdlet (SFBMS):

Set-CsArchivingConfiguration -EnableArchiving ImAndWebConf `  
-EnableExchangeArchiving $True

1. Creating a new archiving policy and granting it to one of our users (SFBMS):

New-CsArchivingPolicy -Identity "TestlabsArchivingPolicy" `  
-ArchiveInternal $True -ArchiveExternal $True

Grant-CsArchivingPolicy -Identity "pdickson" `  
-PolicyName "TestlabsArchivingPolicy"

1. Optional step, configuring the global archiving policy for both internal and external archiving if you want to have this setting applied to the Global policy (SFBMS):

Set-CsArchivingPolicy -Identity "Global" -ArchiveInternal $True `  
-ArchiveExternal $True

1. Configuring the ExchangeArchivingPolicy property is needed if the Exchange server(s) and Skype for Business server(s) are located in different forests. By configuring this option, we are making sure that the instant messages and conference transcripts are being saved into the mailbox. Run the following cmdlet (SFBMS):

Set-CsUser -Identity pdickson `  
-ExchangeArchivingPolicy ArchivingToExchange

1. Creating a new client policy and granting it to one of our users (SFBMS):

New-CsClientPolicy -Identity "TestlabsClientPolicy" `  
-EnableExchangeDelegateSync $True -EnableIMAutoArchiving $True

Grant-CsClientPolicy -PolicyName "TestlabsClientPolicy" `  
-Identity pdickson

1. Optional step, if we want to apply the same settings for the Global clientpolicy this is done by running the cmdlet (SFBMS):

Set-CsClientPolicy -Identity "Global" `  
-EnableExchangeDelegateSync $True -EnableIMAutoArchiving $True

## How it works...

Just like the previous recipe, all integrations between Exchange and Skype for Business is based on OAuth. If you didn’t read the first recipe, I would recommend you to have a look at it before continuing reading this recipe.

We started this recipe by configuring archiving for both instant messaging and conferences. In this configuration, we also made sure that the archiving functionality was enabled. This was done by running this cmdlet:  
Set-CsArchivingConfiguration -EnableArchiving ImAndWebConf `  
-EnableExchangeArchiving $True.

The options for what kind of transcript should be archived can be set to None, ImOnly or ImAndWebConf. None means that nothing gets archived, ImOnly means that only instant messaging will be archived while ImAndWebConf will archive both instant messaging and conferences.

As our next step, we created a new archiving policy and granted this to one of our users. We created a new policy for making sure we don’t impact the Global archiving policy which can impact our users. This was completed by running:

New-CsArchivingPolicy -Identity "TestlabsArchivingPolicy" `  
-ArchiveInternal $True -ArchiveExternal $True

Grant-CsArchivingPolicy -Identity "pdickson" `  
-PolicyName  "TestlabsArchivingPolicy"

Our user named Pete Dickson was granted this newly created policy, which means that both internal and external instant messaging and conferences are enabled for archiving.

When we have validated that all settings work the way we want them to, if we want to apply these settings to the Global archiving policy this can be done:

Set-CsArchivingPolicy -Identity "Global" -ArchiveInternal $True `  
-ArchiveExternal $True

When the configuration has been updated for the Global policy we continue with the next step that only is required if the Exchange server(s) and Skype for Business server(s) are located in different forests.

Set-CsUser -Identity pdickson -ExchangeArchivingPolicy `   
ArchivingToExchange

The valid options for the ExchangeArchivingPolicy are Uninitialized, UseLyncArchivingPolicy, NoArchiving or ArchivingToExchange. All of them have a user-friendly name, which most likely is self-describing to you. The Uninitialized option might need a bit description; it means that if the user has not been enabled for   
in-place hold it will be archived to the Skype for Business server.

With these settings in place we are almost done, the final step is to create and grant a client policy to one of our users. This is simply done by running:

New-CsClientPolicy -Identity "TestlabsClientPolicy" `  
-EnableExchangeDelegateSync $True -EnableIMAutoArchiving $True

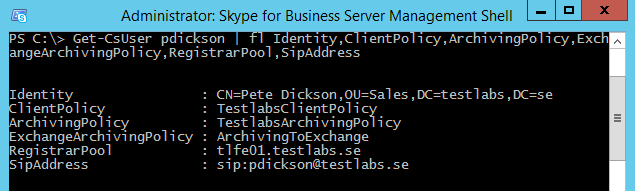
Grant-CsClientPolicy -PolicyName "TestlabsClientPolicy" -Identity pdickson

Our first cmdlet is creating a new client policy and allowing the option of having delegates being able to schedule meetings on behalf of users. The second parameter is to make sure that the archiving is being enabled and saved into the conversations folders in the mailbox. As a final step, we are granting this policy to our user.

If we want to have the same settings configured for the Global client policy, this can be easily done by running:

Set-CsClientPolicy -Identity "Global" -EnableExchangeDelegateSync `  
$True -EnableIMAutoArchiving $True

Let’s take a look at our user that now has these settings in place:



7081EN\_13\_04

## See also

* OAuth configuration
* Unified Contact Store

Placing mailboxes on in-place hold in Chapter 11, Exchange Security

# Unified Contact Store

The Unified Contact Store (UCS) enables the functionality of having all contacts from the Skype for Business client stored in the Exchange mailbox instead of the Skype for Business (SQL database). This recipe has the same requirement for having the integration to work: OAuth communication in place so that it works as expected.

By default, in Skype for Business 2015 the Unified Contact Store is enabled when OAuth is configured and working. This recipe will be completed by using both Exchange Management Shell (EMS) and Skype for Business Management Shell (SFBMS).

## How to do it...

1. Optional step, but personally I like to configure a new policy and then grant this policy to a specific user instead of changing the Global policy, which would by default impact all users. At least before rolling it to Production. Creating a new policy and granting it to the user is accomplished by running the cmdlet:

New-CsUserServicesPolicy –Identity `  
"TestlabsAllowUnifiedContactStore" -UcsAllowed $True

Grant-CsUserServicesPolicy -Identity "pdickson" `  
-PolicyName "TestlabsAllowUnifiedContactStore"

1. Configuring Enterprise partner application from Exchange is done by using the EMS and running the following cmdlets:

cd $exscripts

.\Configure-EnterprisePartnerApplication.ps1 `  
-AuthMetadataUrl "https://tlfe01.testlabs.se/metadata/json/1"`  
-ApplicationType Lync

1. Create a partner application from Skype for Business, by running the cmdlet (SFBMS):

New-CsPartnerApplication -Identity Exchange `  
-ApplicationTrustLevel Full –MetadataUrl ` "https://autodiscover.testlabs.se/autodiscover/metadata/json/1"

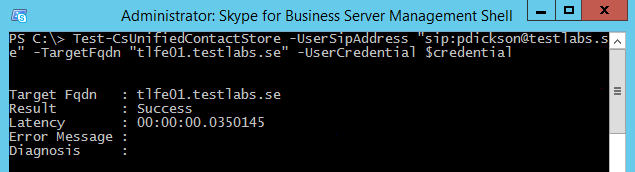
1. Validate the configuration and make sure that Exchange trusts Skype for Business and vice versa. It’s easily completed by running the cmdlet (SFBMS):

Test-CsExStorageConnectivity -SipUri ` "sip:pdickson@testlabs.se"

$credential = Get-Credential "testlabs\pdickson"

Test-CsUnifiedContactStore -UserSipAddress ` "sip:pdickson@testlabs.se" -TargetFqdn "tlfe01.testlabs.se" `  
-UserCredential $credential

The result from the cmdlet above should look similar to the screenshot below:

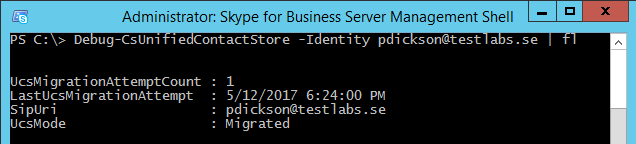


7081EN\_13\_05

1. Debugging is possible when it comes to UCS. This is as easy as running the following cmdlet (SFBMS):

Debug-CsUnifiedContactStore -Identity pdickson@testlabs.se `  
| fl

The result should look similar to the screenshot below:



7081EN\_13\_06

## How it works...

In this recipe, we started by creating a new user services policy and making sure to enable UCS, using the cmdlet New-CsUserServicesPolicy -Identity ` "TestlabsAllowUnifiedContactStore" -UcsAllowed $True and then we granted it to one of our users with Grant-CsUserServicesPolicy -Identity "pdickson" -PolicyName "TestlabsAllowUnifiedContactStore".

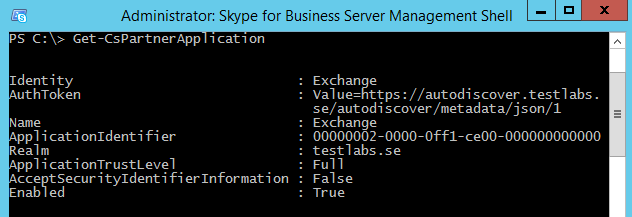
This requirement is also pending on that the systems trust each other, in addition to the already created OAuth connection we need to configure a new partner application from both Exchange towards Skype for Business and vice versa. We started by going to the built-in scripts folder in Exchange, cd $exscripts running the script named “Configure-EnterprisePartnerApplication.ps1”, and for the parameters -AuthMetadataUrl we type in the front-end server/registrar for our users. The full cmdlet looks like this: .\Configure-EnterprisePartnerApplication.ps1 -AuthMetadataUrl "https://tlfe01.testlabs.se/metadata/json/1" -ApplicationType Lync

Just to confirm, the parameter for -ApplicationType is named Lync, it’s not a typo. I have been researching for a new updated value, but by the time this book was authored this value should be Lync.

The partner application can be retrieved from the Exchange server by running the cmdlet Get-PartnerApplication -Identity Lync\*.

On the opposite side, we have to configure a partner application from Skype for Business towards Exchange and pointing the parameter -MetadataUrl to Autodiscover. This is done by running: New-CsPartnerApplication -Identity Exchange `  
-ApplicationTrustLevel Full -MetadataUrl `  
"https://autodiscover.testlabs.se/autodiscover/metadata/json/1".

When having created both applications, let’s also retrieve the recently created partner application on the Skype for Business server by running Get-CsPartnerApplication, the result would look similar to the screenshot below:



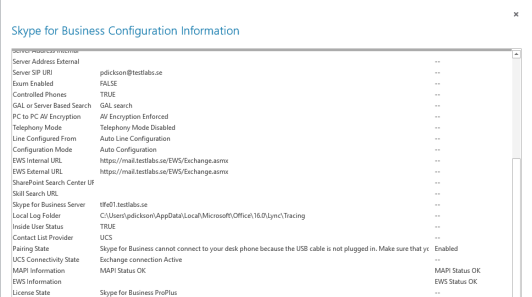
7081EN\_13\_07

With both partner applications in place we want to validate it by using: Test-CsExStorageConnectivity -SipUri "sip:pdickson@testlabs.se", the attribute –SipUri is used to identify one particular user mailbox where the test item should be created. If we get a successful result, we want to go ahead and continue validating the Unified Contact Store with a user account so see that it works. This is done by inserting the user credentials and then validate using these. This is done by running the cmdlets: $credential = Get-Credential "testlabs\pdickson” followed by:   
Test-CsUnifiedContactStore -UserSipAddress "sip:pdickson@testlabs.se" -TargetFqdn "tlfe01.testlabs.se" -UserCredential $credential.

When the connection between the two systems is established the Skype for Business contacts should be moved from the SQL database into the mailbox folder. For this to work, the user of course needs to have logged on and have at least one contact in the contact list.

In addition to these cmdlets, there is also one for debugging the UCS functionality and it can be used by running: Debug-CsUnifiedContactStore -Identity pdickson@testlabs.se | fl.

We can validate that UCS is in place as well from the Skype for Business client by checking the configuration information and see if Contact List Provider is set to UCS or Lync Server. UCS Connectivity State should also be set to Exchange connection Active when UCS is working. With UCS configured it will look similar to the screenshot below:



7081EN\_13\_08

## See also

* OAuth configuration
* Archiving and conversation history

Integrate Skype for Business with OWA

# Integrate Skype for Business with OWA

If you are using Outlook.com as your private email provider, you’ve probably noticed that the consumer version of Skype is integrated in the mail functionality nowadays. Similar functionality can also be achieved with Exchange 2016 and Skype for Business 2015. Instead of the consumer version of Skype we use Skype for Business.

A scenario when this is ideal would be if there are users that might not have the rich Skype for Business client installed or available for some reason, they can then simply go to the webmail and utilize the instant messaging functionality there.

In this recipe, we will take a look at how this can be configured and validated.

## How to do it...

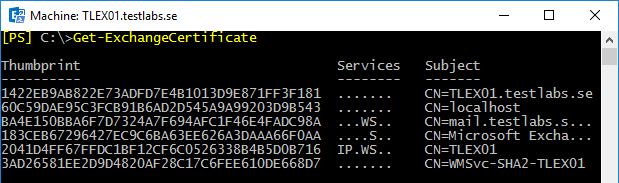
1. To be able to integrate, the Unified Communications Managed API 4.0 Runtime needs to be installed on the Exchange server(s). We can validate that this has been installed by running the cmdlet:

Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Services\MSExchange ` OWA\InstantMessaging" -Name ImplementationDLLPath

1. Retrieve the certificate thumbprint that is used on the Exchange server for webservices:

Get-ExchangeCertificate

The output will look similar to the screenshot below. Save the thumbprint from the certificate assigned for webservice from the output.



7081EN\_13\_09

1. Create an override setting for OWA by specifying the IM server and the IM certificate thumbprint. This is accomplished by running the cmdlet:

New-SettingOverride -Name "IM Override" -Component   
OwaServer -Section IMSettings –Parameters @("IMServerName=tlfe01.testlabs.se","IMCertificateThumbprint=BA4E150BBA6F7D7324A7F694AFC1F46E4FADC98A") -Reason "Configure IM"

1. Refresh the IM settings on the Exchange server by running the cmdlet:

Get-ExchangeDiagnosticInfo -Process Microsoft.Exchange.Directory.TopologyService -Component VariantConfiguration -Argument Refresh

1. Retrieve the Skype for Business Site IDs

Get-CsSite | fl DisplayName, SiteID

1. Create a new trusted application pool and a trusted application for OWA usage, this is done by executing the cmdlet:

New-CsTrustedApplicationPool -Identity "mail.testlabs.se" -Registrar tlfe01.testlabs.se -Site "Default Site" -RequiresReplication $false

New-CsTrustedApplication -ApplicationId OutlookWebApp -TrustedApplicationPoolFqdn mail.testlabs.se -Port 5199

1. Enable the changes that were just done for Skype for Business:

Enable-CsTopology

1. Once the configurations are set, restart the application pool for OWA by running the cmdlet:

Restart-WebAppPool MSExchangeOWAAppPool

1. Configure OWA to be enabled for instant messaging of the instant messaging type OWA:

Get-OwaVirtualDirectory | Set-OwaVirtualDirectory -InstantMessagingEnabled $True -InstantMessagingType OCS

1. Optional step: if you want to enable the OWA integration for the Default OwaMailboxPolicy this can be done by running:

Set-OwaMailboxPolicy -Identity "Default" -InstantMessagingEnabled $True - InstantMessagingType "OCS"

1. Validate that the instant messaging integration works as expected, pending on version run the cmdlet differently:

CU3 and earlier:

[xml]$diag = Get-ExchangeDiagnosticInfo -Server tlex01 -Process Microsoft.Exchange.Directory.TopologyService -Component VariantConfiguration -Argument Config; $diag.Diagnostics.Components.VariantConfiguration.Configuration.OwaServer.IMSettings

CU4 and later:

[xml]$diag = Get-ExchangeDiagnosticInfo -Server tlex01 -Process MSExchangeMailboxAssistants -Component VariantConfiguration -Argument "Config,Component=OwaServer"; $diag.Diagnostics.Components.VariantConfiguration.Configuration.OwaServer.IMSettings

## How it works...

In this recipe, we start by validating that the Unified Communications Managed API 4.0 Runtime has been installed by validating that the ImplementationDLLPath has a valid value set.

Before creating the instant messaging override settings, we need to retrieve the thumbprint of the Exchange certificate assigned for webservices. This is done by running Get-ExchangeCertificate and saving the thumbprint for next step.

Next step is to create a new override setting for OWA, defining the thumbprint that are going to be used for this. We are also specifying the Skype for Business server. The full cmdlet will be: New-SettingOverride -Name "IM Override" `  
-Component OwaServer -Section IMSettings -Parameters ` @("IMServerName=tlfe01.testlabs.se","IMCertificateThumbprint=7800000`0040BC7C72721CE1FAD000000000004") -Reason "Configure IM"

With this setting in place it’s time to refresh the configuration by running:   
Get-ExchangeDiagnosticInfo -Process ` Microsoft.Exchange.Directory.TopologyService -Component ` VariantConfiguration -Argument Refresh

Now we need to collect the Site ID from Skype for Business, this is done from the Skype for Business Management Shell by running: Get-CsSite | fl DisplayName, SiteID. The Site ID will be used in our next step when creating the trusted application pool: New-CsTrustedApplicationPool -Identity "mail.testlabs.se" `  
-Registrar tlfe01.testlabs.se -Site "Default Site" `  
-RequiresReplication $false.

As you see in the example above, the Site ID in my Skype for Business setup is “Default Site”. With the application pool in place, we can continue and create the trusted application too: New-CsTrustedApplication -ApplicationId OutlookWebApp `  
-TrustedApplicationPoolFqdn mail.testlabs.se -Port 5199.

Our Skype for Business server has now been configured, the final step is to enable the configuration we did above by running: Enable-CsTopology.

Let’s finish this configuration by changing the configuring for Outlook on the web, enabling instant messaging for OWA. This is done by using:   
Get-OwaVirtualDirectory | Set-OwaVirtualDirectory `  
-InstantMessagingEnabled $True -InstantMessagingType OCS

It’s correct, the parameter InstantMessagingType value should be configured to OCS. This has for some reason not been updated by Microsoft yet, so it’s still named OCS. However, the important part is to make the integration work.

If you want to implement these settings for the OwaMailboxPolicy as well, it can simply be done by running: Set-OwaMailboxPolicy -Identity "Default" `  
-InstantMessagingEnabled $True - InstantMessagingType "OCS".

Our final configuration step is to restart the application pool for OWA, done by running: Restart-WebAppPool MSExchangeOWAAppPool. It may take a couple of seconds for the restart, but it’s rather quick. Keep in mind to plan for these integration changes since they require restart of services that impact the end-users.

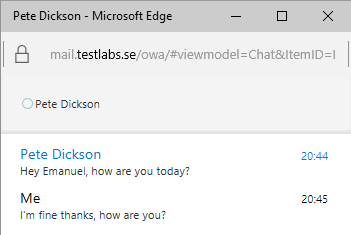
The configuration is in place and we want to validate that everything works as we expect it to, pending on the cumulative update status on your environment the cmdlets differs a bit. If you are running CU3 or earlier you can validate the configuration by running:

[xml]$diag=Get-ExchangeDiagnosticInfo -Server tlex01 -Process ` Microsoft.Exchange.Directory.TopologyService -Component ` VariantConfiguration -Argument Config; ` $diag.Diagnostics.Components.VariantConfiguration.Configuration.`  
OwaServer.IMSettings

If you are you on CU4 or later , the following cmdlet should be used:

[xml]$diag=Get-ExchangeDiagnosticInfo -Server tlex01 -Process ` MSExchangeMailboxAssistants -Component VariantConfiguration `  
-Argument "Config,Component=OwaServer"; ` $diag.Diagnostics.Components.VariantConfiguration.Configuration.`  
OwaServer.IMSettings

After all these steps, it’s time to do a live test and see if the OWA integration works as we want it to. I have signed in with one user on the rich Skype for Business application and with another one to the OWA. The instant messaging seems to work great between the two, take a look at the screenshot below:



7081EN\_13\_10

## See also

* OAuth configuration
* Archiving and conversation history

Using high-resolution photos

# Using high-resolution photos

We have had photos in the GAL for some time now, they used to be saved in an Active Directory attribute. With Exchange 2013 and later we now have the option of using high-resolution photos instead of the traditional 48 \* 48 pixels photo that were saved in the thumbnailPhoto attribute in Active Directory.

With Exchange 2013 and later, OWA, Outlook and Skype for Business Web App and Skype for Business client support the usage of larger photos, 96 \* 96 pixels.

We also have the option of having even larger photos that are sized to 648 \* 648 pixels, those will work with both Skype for Business client and the Web App client.

Keep in mind that since these photos are saved in the mailbox and not in the Active Directory attribute, the mailbox size will increase a bit.

In this recipe, we will take a look on how we can configure a user with a high-resolution photo, which can be accomplished by end-users themselves through OWA or the Skype for Business client, but in this book we are focusing on the PowerShell piece.

## How to do it...

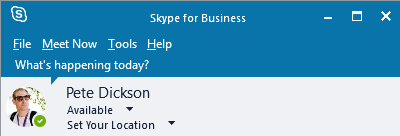
1. Load the photo into a byte variable, by running:

$photo = ([Byte[]] $(Get-Content -Path ` "C:\Photos\pdickson.jpg" -Encoding Byte -ReadCount 0))

1. Set the photo for a specific user:

Set-UserPhoto -Identity pdickson -PictureData $photo -Confirm:$False

The end-result of this recipe should look similar to the picture below:



7081EN\_13\_11

1. Optional step, if we want to validate how the photo looks like for this specific update we simply follow the link below and update the email value with a matching one:

https://mail.testlabs.se/ews/Exchange.asmx/s/GetUserPhoto?email=pdickson@testlabs.se&size=HR648x648

## How it works...

we are now focusing on configuring the end-user with a high definition photo instead of saving the photo into the Active Directory attribute thumbnailPhoto.

We started this recipe by reading the jpg file into a byte variable that was saved as $photo and then used that variable to set it as the photo for a specific user. All this was done by running the following two cmdlets:

$photo = ([Byte[]] $(Get-Content -Path "C:\Photos\pdickson.jpg" -Encoding Byte -ReadCount 0))

Set-UserPhoto -Identity pdickson -PictureData $photo -Confirm:$False

To validate that the photo got saved and that it was saved as the 648 \* 648 pixel that it should, we can easily use a link to look it up.

https://mail.testlabs.se/ews/Exchange.asmx/s/GetUserPhoto?email=pdickson@testlabs.se&size=HR648x648.

## See also

* OAuth configuration

Integrate Skype for Business with OWA

# How to integrate Office Online Server

In this recipe, we will take a look at how to integrate the Office Online Server (OOS) with Exchange 2016. In Exchange Online in Office 365, we already have this functionality/workload with Office Online, where you receive an email with an attachment and it gives our end-users the option of opening the attachment in a feature rich browser-based application without having to download them first.

A requirement for having this integration is to be on Exchange 2016, Cumulative Update 1 level or later. If you want your users to be able to view the attachments even outside of your network, the OOS server needs to be reachable from Internet through TCP port 443.

With this short introduction, let’s go through the steps we have to perform to have this integration.

## How to do it...

Before jumping into the steps below, make sure that you have the name configured in DNS for the OOS. If you are about to publish this server you also need to have the DNS record in your public DNS zone. Your chosen OOS name needs to be included in the certificate on the OOS server(s). In my example, I only have one OOS server, I’ve added the oos.testlabs.se into the web server certificate on the OOS server and into the internal DNS zone.

1. Validate if any existing configuration is already done, if not then create a new Office Web Apps Farm, by running:

Get-OfficeWebAppsFarm

New-OfficeWebAppsFarm -InternalURL "https://oos.testlabs.se" ` -ExternalURL "https://oos.testlabs.se" -CertificateName `  
"OOS certificate"

1. Start up any internet browser and go to, validate there is no certificate errors and that the XML file is being shown:

https://oos.testlabs.se/hosting/discovery

1. Before configuring, let’s validate that the OOS isn’t already configured for Exchange. If not, then continue to configure OOS at the organizational level for Exchange 2016 by running:

Get-OrganizationConfig | fl WacDiscoveryEndpoint

Set-OrganizationConfig -WacDiscoveryEndpoint `  
"https://oos.testlabs.se/hosting/discovery"

1. Restart the web application pool for OWA:

Restart-WebAppPool MsExchangeOwaAppPool

If you have Exchange 2013 servers in your organization, remember to configure OOS at the mailbox server level instead of the organizational level because it’s not supported to use OOS together with Exchange 2013.

## How it works...

In this recipe, the prerequisite is to have Microsoft .NET Framework 4.5.2, Visual C++ Redistributable for Visual Studio 2015 plus a full stack of roles and features. All of them can be installed by running the following one-liner:

Install-WindowsFeature Web-Server, Web-Mgmt-Tools, Web-Mgmt-Console, Web-WebServer, Web-Common-Http, Web-Default-Doc, Web-Static-Content, Web-Performance, Web-Stat-Compression, Web-Dyn-Compression, Web-Security, Web-Filtering, Web-Windows-Auth, Web-App-Dev, Web-Net-Ext45, Web-Asp-Net45, Web-ISAPI-Ext, Web-ISAPI-Filter, Web-Includes, InkandHandwritingServices, Windows-Identity-Foundation.

With the mentioned prerequisites above in place, we also need to have the DNS record in place, both internal and external (if you are going to publish the server(s) externally) and include the DNS name in the web certificate on each server (if you have more than one).

The publishing for the OOS servers is easy, it only uses TCP/443 port and there is no requirement for any pre-authentication so you can simply publish HTTPS for the OOS server(s) and you are good to go.

Regarding the steps, we started by validating that there are no existing Office Web Apps Farm by running: Get-OfficeWebAppsFarm.

In our example, we did not have any current configuration for the Office Web Apps Farm so we continued to create a new farm by running:

New-OfficeWebAppsFarm -InternalURL "https://oos.testlabs.se" `  
-ExternalURL "https://oos.testlabs.se" -CertificateName `  
"OOS certificate"

The cmdlet is specifying the parameters -InternalURL and -ExternalURL, these can differ from each other and should of course match the internal and external DNS record. Also keep in mind to include both names if you don’t use the same for both of them.

The parameter -CertificateName is specifying the friendly name of the OOS certificate installed on the OOS server, in our example it was “OOS certificate”.

With the farm in place we want to validate that it responds to requests. We tested this by starting an internet browser and navigating to: https://oos.testlabs.se/hosting/discovery.

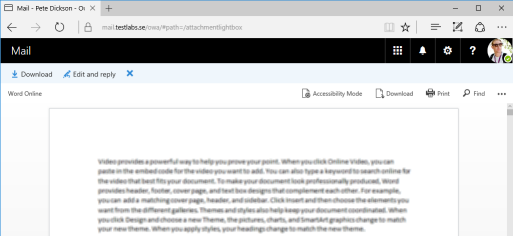
Before taking the next step in our configuration, we want to make sure that there is not already a configuration in place that we might overwrite. We are checking the organization configuration and looking for the parameter -WacDiscoveryEndpoint, by running: Get-OrganizationConfig | fl WacDiscoveryEndpoint.

The parameters value was empty, meaning there are no current configuration so we can continue by running: Set-OrganizationConfig -WacDiscoveryEndpoint "https://oos.testlabs.se/hosting/discovery".

The only step left is to restart the application pool for OWAby running:

Restart-WebAppPool MsExchangeOwaAppPool.

With the integration in place, it should look similar to the screenshot below when opening up an attachment directly in a received email:



7081EN\_13\_12

## See also

* OAuth configuration
* Archiving and conversation history

Integrate Skype for Business with OWA

# Exchange Hybrid validation

It’s very common to find articles, blog posts and other similar information about how to configure and setup the Exchange hybrid, but it’s not very common to find similar information about validating that the functionality is working as expected. We are trying to fill that gap with this recipe.

Therefore, we won’t go into detail on how to configure and setup the hybrid, instead we will focus on getting the functionality validated.

In this recipe, we will assume that you have an existing Exchange 2016 hybrid solution configured for Exchange 2016 and Exchange Online that have fulfilled the requirements that are well documented at TechNet.

With that said, let’s jump in and start on how to validate the integration.

## How to do it...

Before the steps below are performed, we assume that the Exchange Hybrid is successfully configured and fulfils all prerequisites. For reference, for this recipe we have one Exchange 2016 server, configured successfully by running the hybrid wizard configuration using a third-part certificate by a trusted issuer (DigiCert).

1. Validate that the federation trust certificate is in place, this is done by running:

Test-FederationTrustCertificate

1. With the federation trust certificate in place, let’s retrieve the federation trust and test by running the cmdlet:

Get-FederationTrust

Test-FederationTrust

1. Retrieve the organization relationship information and save the attribute Name into the $name variable, then test it using the cmdlet:

$name = (Get-OrganizationRelationship).Name

Test-OrganizationRelationship -UserIdentity ` pdickson@testlabs.se -Identity $name

Get-OrganizationRelationship | Test-OrganizationRelationship `  
-UserIdentity pdickson@testlabs.se

1. Final step is to validate OAuth since it’s being used when having a Hybrid solution, we can validate the Hybrid by running three different setups of cmdlets:

Test-OAuthConnectivity

Test-OAuthConnectivity -Service EWS -TargetUri ` https://mail.testlabs.se/ews/ -Mailbox "pdickson"

Test-OAuthConnectivity -Service AutoD -TargetUri ` https://mail.testlabs.se/autodiscover/autodiscover.xml `  
-Mailbox "pdickson"

Test-OAuthConnectivity -Service Generic -TargetUri `https://mail.testlabs.se/ -Mailbox "pdickson"

## How it works...

In this recipe, the prerequisite is to have a third-part certificate that is issued by a trusted issuer, in my example I’m using a certificate from DigiCert. The certificate is required for having a fully working hybrid solution, since Office 365 (Exchange Online) won’t trust a certificate issued by an internal issuer (Internal PKI).

The certificate should be assigned to at least the IIS and SMTP service, since it’s going to be used for these services for Autodiscover, Webservices and for the mail flow between the on-premises solution and the cloud solution.

We started to validate that there is an existing federation trust certificate, this is being issued during the hybrid configuration and it's important to have in-place in a hybrid scenario. This is done by running: Test-FederationTrustCertificate.

With the certificate in place, we want to continue and see if the federation trust was created successfully and we also would like to execute one of the Test- cmdlets.

The cmdlet: Get-FederationTrust, is being used to retrieve the federation trust (if any). Normally, we should get a result back with the federation trust information. Now we want to test the federation trust to see if it actually works or not. This is done by running:

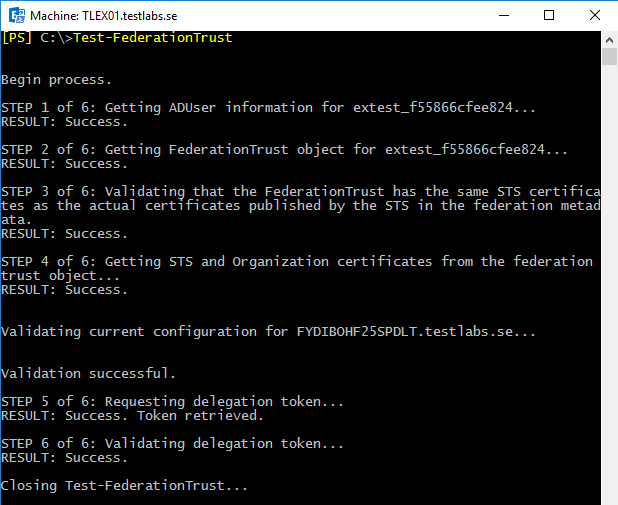
Test-FederationTrust

One thing to note here is that this Test- cmdlet might fail if the   
New-TestCasConnectivityUser.ps1 haven’t been executed prior to the Test- cmdlet.

If needed, the creation of the extest mailbox can be done by running:

& $env:ExchangeInstallPath\Scripts\New-TestCasConnectivityUser.ps1

The outcome from the Test- cmdlet should look similar to the screenshot below:



7081EN\_13\_13

If the federation trust was successful, then we continue to our next step and validate the organization relationship. This is also being created and configured by the hybrid wizard configuration. There are two ways of validating this, select which method you prefer since both of them works great.

Method 1:

$name = (Get-OrganizationRelationship).Name

Test-OrganizationRelationship -UserIdentity pdickson@testlabs.se `  
-Identity $name

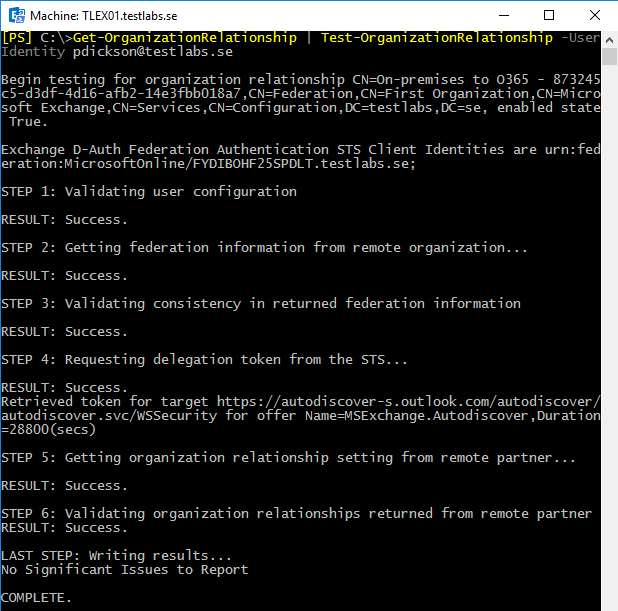
The name of the organizational relationship is being saved into a variable called $name, which is then used as an input variable to the Test- cmdlet instead of specifying a long name, similar to the example: "On-premises to O365 – 279245a5-d3df-4d16-afb2-14e3fbb018a7".

Method 2:

Get-OrganizationRelationship | Test-OrganizationRelationship `  
-UserIdentity pdickson@testlabs.se

This method is my personal preferred method since it’s both retrieving and then pipelining the organization relationship information into the Test- cmdlet and just using the parameter -UserIdentity to test with.

The outcome and result from these two methods of the Test- cmdlets should look similar to the screenshot below:



7081EN\_13\_14

The fourth and final validation is to check that OAuth is working as expected, and this can easily be done by running the cmdlet Test-OAuthConnectivity, an example for validating the EWS service:

Test-OAuthConnectivity -Service EWS -TargetUri ` https://mail.testlabs.se/ews/ -Mailbox "pdickson"

In addition to EWS, Autodiscover and Generic can also be used as an input to the service parameter. See the examples in the how to do it section.

## See also

* OAuth configuration
* Office Online Server integration

Integrate Skype for Business with OWA