Appendix A

Using the Windows PowerShell Functions

This appendix provides instructions for using the Microsoft Windows PowerShell functions included in the script file found on the companion CD. These functions are contained within a single PowerShell script that contains over 90 sample PowerShell functions for administering various aspects of the Microsoft Office Communications Server 2007 R2 environment. Note that these are sample functions only, not finished tools. They are provided as proof-of-concept examples of how to administer Office Communication Server using Windows PowerShell. Although the Microsoft Office Communications Server 2007 R2 Resource Kit author team has made every effort to test these functions to ensure that they work properly, these functions are provided "as is" with no warranty or guarantee concerning their functionality. You should therefore become thoroughly familiar with using these functions in a test environment before using them in your production environment.

Because these functions are provided as proof-of-concept samples only, you might need to customize these functions if you intend to use them in your production environment. For example, the functions as provided include only minimal support for error handling and assume that the servers they are being run against exist and are configured appropriately. The Microsoft Office Communications Server 2007 R2 Resource Kit author team therefore encourages you to customize these functions to meet the needs of your own networking environment.

Script Signing and Customizing the PowerShell Functions

PowerShell has an Execution Policy setting that controls the behavior of scripts. The *Set-ExecutionPolicy* execution policy, cmdlet, allows this to be set to one of the following four possible values:

Restricted

- This is the default setting, and it specifies that no scripts at all will be allowed to run.

AllSigned

- This setting requires all scripts to have a digital signature.

RemoteSigned

- This setting specifies that only scripts run from file shares, downloaded in Internet Explorer, or received as mail attachments must be signed.

Unrestricted

- This setting specifies that all scripts can run.

The execution policy can be changed by running *Set-ExecutionPolicy* <*value*> from the PowerShell window, where <*value*> is one of the preceding four values. Before you begin, you should determine what the scripting policy is on your system by entering the command **Get-ExecutionPolicy** at the PowerShell prompt. If it returns *Restricted*, you need to use the *Set-ExecutionPolicy* cmdlet to allow scripts to run. Microsoft recommends that in a production environment all PowerShell scripts be signed and the execution policy be configured as *AllSigned*. If you feel the risks of running unsigned scripts have been sufficiently mitigated (for example, you are running Office Communications Server in a self-contained test environment), you can reduce the level of protection using either *Set-ExecutionPolicy RemoteSigned* or *Set-ExecutionPolicy Unrestricted*.

The companion CD includes the OCS-R2.PS1 script file. This script file has not been digitally signed by Microsoft, but you can self-sign the scripts to be used in environments where the execution policy is configured as *AllSigned*. Best practice would have you sign it from a code signing certificate from your CA, which will allow it to be run anywhere in your environment. The companion CD also provides these functions in the form of an unsigned text file, OCS-ALL.txt, so that you can customize these functions as needed. To use the functions after you have customized them, start by renaming the text file, so that it has the .ps1 file extension. If you are following the recommendation to run under the *AllSigned* policy, you need a code-signing certificate and you should use the *Set-AuthenticodeSignature* cmdlet to sign your script. If you set the policy to *RemoteSigned* or *Unrestricted* to avoid signing scripts, you should understand the risks involved in doing this and take suitable steps to mitigate these risks.

> **Note** For more information about signing, enter **Get-Help about_signing** at the PowerShell prompt.

How to Load the PowerShell Functions

The PowerShell functions provided are contained within a single PowerShell script (.ps1) file named *OCS-R2.PS1*. To use these functions, you first have to load them from the PowerShell command prompt using the following syntax:

```
. <path>\OCS-R2.PS1
```

The initial dot tells PowerShell, "Keep all of these functions in memory after you've finished running the script." This detail is important because developers often forget the dot, and users end up wondering why the script seemed to have loaded but they didn't get any additional functions.

Note that PowerShell doesn't add the current directory to the path, so if the files are in the current directory, you need to invoke the script using the following syntax:

```
. .\OCS-R2.PS1
```

You will get a message telling you the functions have been loaded.

If you are going to the use the functions frequently, it is worth adding the dot commands to your PowerShell profile. To find the path to the profile, you can enter *$Profile* at the PowerShell command prompt. The path to your profile, by default, should be:

C:\Users\<your username>\Documents\PowerShell\Microsoft.PowerShell_profile.ps1

Usually, you need to create the file (for example, using Notepad) and directories, then add the appropriate lines—for example:

```
. <path>\OCS-R2.PS1
. .\OCS-R2.PS1
```

OCS-R2.PS1 is a PowerShell script, and if you are using the *AllSigned* execution policy, it needs to be signed. Of course, you *can* (not recommended, due to malware potential) set your execution policy to *UnRestricted*. (See preceding note about signing of PowerShell scripts, and do the Secure thing and sign the script..)

Understanding Command and Parameter Naming

PowerShell *cmdlets* are named in the form *Verb-Noun*, and the PowerShell functions developed for this Resource Kit follow this naming convention. In particular, nouns that we have provided are named *OCSSomething*, or *OCSEdgeSomething*, if they apply specifically to an edge server role. The functions were developed for Office Communications Server 2007 R2 from scripts that were published for the initial release of Office Communications Server 2007. Many of them should also work with the previous version, Live Communications Server 2005 SP1.

By convention, PowerShell commands that begin with the *Get* verb return objects that can be used by other commands. For example, *Get-Process* returns a set of process *Objects*. When these objects have been retrieved, they can then be used as input to other PowerShell commands using the pipe syntax ("|"). Following are examples of using the pipe syntax:

- Sorted by property name using **get-process | sort-object -Property ProcessName**
- Filtered to specific processes with **get-process | Where-Object {$_.processname -eq 'notepad'}**
- Exported to a comma-separated values (CSV) file with **get-process | export-CSV -path "Processes.csv"**
- Counted with **get-process | Measure-Object**

The PowerShell functions developed for this Resource Kit also follow this convention. By convention, PowerShell commands use nouns written in the singular, and although the commands used in the previous version of the Resource Kit used plural nouns, we have come into line with this convention in this release. In addition, PowerShell developers are recommended to avoid creating new verbs where it is possible to use existing ones, and some of the commands have been renamed to use verbs already used by PowerShell

Office Communications Server exposes management interfaces through Windows Management Instrumentation (WMI). (See MSDN for WMI documentation.) Many of the GET functions simply fetch the corresponding WMI object or objects. By default, outputting these objects to the screen will not produce nicely formatted results. However, in this release we have provided a custom formatting XML file named `ocsTypes.format.ps1xml`. This tells PowerShell how the objects should be formatted. This is loaded using the PowerShell cmdlet, ***`Update-formatData`***. The name of the XML file only needs to be specified when it is first used. If it is updated during a PowerShell session, then it can be run without adding the file a second time. In the previous release of the Resource Kit, no XML was provided. Functions named List were provided for each Get- command. The XML file made these redundant, and they have been removed.

For example, the *Get-OCSUser* function is the PowerShell command version of *Get-WMIObject –Class MSFT_SIPESUserSetting*. It returns an array of *MSFT_SIPESUserSetting* objects. We can pipe these objects into another PowerShell command, which might also return an array of objects, which can be piped into another command, and so on. For example, the following line of PowerShell syntax moves all the users on one server to another:

```
Get-OcsUser | where-object {$_.HomeServerDN –eq $OldServer} |
    forEach-Object {$_.HomeServerDN = $NewServer ; $_.Put() }
```

The GET function returns objects for all Office Communications Server users, and *where-object* selects only users whose home server matches variable *$OldServer*. Then for each of the selected users, the *HomeServer* property is set to whatever is stored in *$NewServer*. The

change is not committed until the *Put()* method is called. (Variables are explained at the end of this appendix.)

This example shows how *GET-OCSUser* can be used; however, if you run it on its own, you'll see the objects formatted according to the XML provided. You can customize the display permanently by changing the XML, but there are other options.

PowerShell can create a new object from an existing one, using a subset of its properties—for example:

```
Get-WMIObject –Class MSFT_SIPESUserSetting | Select-Object –property

    DisplayName, PrimaryUri, HomeServerDN
```

This code creates new PowerShell custom objects with just the *Display*, *URI*, and *Server* properties and these are far easier to read. But because they are no longer *MSFT_SIPESUserSetting* objects, we can't change a property and save the results.

PowerShell provides *format-table* and *format-list* commands, which allow you to choose how the objects are displayed.

```
Get-OCSUser | format-Table –property DisplayName, PrimaryUri,

    HomeServerDN.
```

This command gets all the *MSFT_SIPESUserSetting* objects and produces a table with their *Name*, *URI*, and *Home Server* properties.

If you make changes to the OCS-R2.PS1 PowerShell script, you should understand that many of the *Get-* functions have other functions that depend on their output.

Using the Common Parameters

Some of these PowerShell functions can be passed parameters. For functions that allow parameters to be passed, such parameters are optional. However, for example, *Remove-OCSUser* must to be told which user, *Export-OCSUsers,* does need to be given the path where its file should be written, and so on. Parameters in PowerShell can be identified by their sequence, but if you want to refer back to what you have done, it is always better to specify them explicitly. Parameters that are used frequently include the following:

-URI

- Specifies the SIP URI of a user. This parameter checks for the presence of **SIP:** and adds it if it is not found.

-Path

- Specifies the path to a file. It is safest to always enclose paths in quotes, although if the file name doesn't include spaces, the quotes aren't required.

-Condition

- Specifies a term to include in a WHERE statement. No checking of this is done.

-Server

- Specifies the name of the remote server on which the command is to be run.

Summary of the PowerShell Functions

Table A-1 summarizes the various functions provided in the OCS-R2.PS1 script. The sections that follow this table provide additional details concerning each of these functions, along with a few examples to illustrate their use.

Table A-1

Type	Add	Choose	Export	Get	Import	New	Remove	Update
ADUser			X	X				
OCSADContainer				X				
OCS * Cert[1]				X				
OCSEdgeFederationDenied				X		X	X	
OCSEdgeFederationPartner			X	X	X	X	X	
OCSEdgeIMProvider				X		X	X	X
OCSEdgeInternalDomain				X		X	X	
OCSEdgeInternalServer				X		X	X	
OCS* Event[2]				X				
OCSGlobalUCSetting				X				
OCSInstalledService				X				
OCSLocationProfile		X		X		X	X	
OCSMediationServer		X						
OCSMediationServerSetting				X				
OCSMeetingPolicy		X		X		X	X	X
OCSNormalizationRule		X		X		X	X	X

OCSPhoneRoute	X		X		X	X	X
OCSPhoneRouteUsage	X		X		X	X	
OCSPICUserCount			X				
OCSPool	X		X				
OCSSchemaVersion			X				
OCSSIPDomain			X		X	X	
OCSSipRoutingCert			X				
OCSTrustedService			X				
OCSUCPolicy	X		X		X	X	X
OCSUser		X	X	X	X	X	X
OCSUserDetail			X				
OCSWindowsService			X				

[1] The Cert row of the table covers 5 different kinds of certificates.

[2] The Event row of the table covers Warning Events and Error Events.

Active Directory

The following functions can be used to manage different aspects of Active Directory Domain Services, as they relate to Office Communications Server 2007 R2.

Get-ADUser

Optional Parameters: *-Container, filter*

This function returns a collection of Active Directory user objects. The *-Container* parameter is used to select a path to an Active Directory domain, container, or organizational unit (OU)—for example, CN=Users,DC=LitwareInc,DC=com. If it is not specified, all of Active Directory is searched for users. The filter parameter specifies an LDAP filter, which restricts the user selection

Example: This function can be piped into other commands. So, to enable all the users in the sales OU, you could use the following syntax:

```
get-aduser -container "OU=Sales,DC=litwareinc,DC=COM" |

    new-ocsuser  -homeserverDN  (Choose-OCSPool)
```

Get-ADUser returns an array of *DirectoryService.SearchResult* objects representing the users in the sales OU. For each one, the *new-OCSUser* function is invoked, assuming that the users all have mail addresses (any that don't will cause an error), and the function is running on the server where the users are to be homed.

Export-ADUser

Required Parameter: *-path*

Optional Parameters: *-Container, filter*

This function exports Active Directory user data to a CSV file to make it easier to enable them in bulk. Note that the field names in Active Directory and OCS-WMI are not the same. *Export-ADUser* preserves the Active Directory field names, and *Export-OCSUser* preserves the OCS-WMI names.

The data to be exported is returned by the *GetADUser* function, and the *–container* and *–filter* parameters are passed straight through to it. So, if no conditions are specified, all users are exported.

Get-OCSADContainer

Office Communications Server uses Active Directory objects to hold its configuration information. These objects are stored in a single Active Directory container, and this function returns the Lightweight Directory Access Protocol (LDAP) path to the container. It also sets a PowerShell local variable *$OCSADContainer*, so that other functions do not have to repeat the search.

Get-OCSSchemaVersion

This function returns the version of the Office Communications Server updates that have been applied to the Active Directory schema.

Certificates

The following functions can be used for managing certificates in an Office Communications Server 2007 R2 environment. Each displays the friendly name for the certificate, the machine that is the subject of the certificate, the authority that issued the certificate, the dates when the certificate is valid, and the usages for which it is valid.

Get-OCSSipRoutingCert

This function returns information about the Standard Edition Server, Enterprise pool front-end server, and Mediation Server certificates. It runs against the local server only.

Get-OCSEdgeAvAuthCert

This function returns information about the A/V Edge Server certificate. It runs against the local server only.

Get-OCSEdgeConferencingExternalCert

This function returns information about the Conferencing Edge Server certificate. It runs against the local server only.

Get-OCSEdgeFederationExternalCert

This function returns information about the Access Edge Server's public edge certificate. The Office Communications Server WMI object refers to this as a *Federation certificate*, and the function follows that convention. However, the same certificate is also used for remote access. It runs against the local server only.

Get-OCSEdgeFederationInternalCert

This function returns information about the Access Edge Server's private edge certificate. The Office Communications Server 2007 R2 WMI object refers to this as a *Federation certificate*, and the function follows that convention. However, the same certificate is also used for remote access. It runs against the local server only.

Edge Server: Federation Denied List

The following functions can be used for managing the Federation-Denied list in an Office Communications Server 2007 R2 environment. Note that these functions are referred to as *blocked* in the Office Communications Server 2007 R2 Management Console.

Get-OCSEdgeFederationDenied

Optional Parameters: *-Server*

This function returns the Federation-Denied list (from the WMI Object *MSFT_sipFederationDeniedDomainSetting*) on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

New-OCSEdgeFederationDenied

Required Parameter: *-Domain*

Optional Parameters: *-Server*

This function adds the specified domain to the Federation-Denied list on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Remove-OCSEdgeFederationDenied

Required Parameter: *-Domain*

Optional Parameters: *-Server*

This function removes the specified domain from the Federation-Denied list on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Edge Server: Federation Partners

The following functions can be used for managing federation partners in an Office Communications Server 2007 R2 environment. Note that these functions are referred to as *Allowed* in the Office Communications Server 2007 R2 Management Console.

Get-OCSEdgeFederationPartner

Optional Parameters: *-Server*

This function returns the federation partners on an Access Edge Server (from the WMI object *MSFT_SIPFederationPartnerTable*). If no server parameter is specified, this function runs against the local server.

New-OCSEdgeFederationPartner

Required Parameter: *-Domain*

Optional Parameters: *-Server*-*EdgeProxyAddress*, *-UserVerificationLevel*

This function adds the specified domain to the list of federation partner domains on an Access Edge Server. If an edge proxy address is specified for the partner domain, it will be located by looking up an SRV record.

Export-OCSEdgeFederationPartner

Required Parameter: *-Path*

Optional Parameters: *-Server*

This function exports federation partners to the CSV file specified in *path*. If no server parameter is specified, this function runs against the local server.

Import-OCSEdgeFederationPartner

Required Parameter: *-Path*

This function reads a CSV file and, for each line, creates a new federation partner on the local server for the given domain, including the edge proxy address if it is specified.

Remove-OCSEdgeFederationPartner

Required Parameter: *-Domain*

Optional Parameters: *-Server*, *-EdgeProxyAddress*, *-UserVerificationLevel*

This function removes the specified domain from the list of federation partner domains on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Example: To delete all domains and then re-import from partners.csv, use the following syntax:

```
get-ocsedgefederationPartner | foreach {Remove-OCSEdgeFederationPartner

    -domain $_.domain}
import-OcsEdgeFederationPartner –Path partners.csv
```

Edge Server: IM Providers

The following functions can be used for managing instant messaging (IM) providers in an Office Communications Server 2007 R2 environment.

Get-OCSEdgeIMProvider

Optional Parameters: *-Server*

This function returns a list of IM providers on an Access Edge Server (from the WMI object *MSFT_SIPFederationNetworkProviderTable*). By default, this list contains the three public IM providers: MSN, AOL, and Yahoo!. If no server parameter is specified, this function runs against the local server.

The information stored about each provider includes its EdgeProxyAddress, whether the provider is a public IM service provider and whether a connection to this IM service provider is enabled. It also shows the options for filtering incoming communications. WMI provides the following values for the options for filtering incoming communications:

- *UseSourceVerification*—This value corresponds to the following option in the Office Communications Server 2007 R2 Management Console: Allow Communications Only From Users On Recipient's Contact List.
- *AlwaysVerifiable*—This value corresponds to the following option in the Office Communications Server 2007 R2 Management Console: Allow All Communications From This Provider.
- *AlwaysUnVerifiable*—This value corresponds to the following option in the Office Communications Server 2007 R2 Management Console: Allow All Communications Only From Users Verified By This Provider.

If no server parameter is specified, this function runs against the local server.

New-OCSEdgeIMProvider

Required Parameters: *-Name*, *edgeProxyAddress*

Optional Parameters: *UserVerificationLevel -Enabled -IsPublic*

This function adds a new IM provider to an edge server, setting the name and Access Edge Server address. If *UserVerificationLevel* is not specified, this function defaults to *UseSourceVerification*. (See *Get-OCSIMProviders*.) If *-enabled* and *-Ispublic* are not specified, they are set to false.

Update-OCSEdgeIMProvider

Required Parameters: *-Name*

Optional Parameters: *UserVerificationLevel -edgeProxyAddress -Enabled -IsPublic*

This function updates the IM provider specified by *Name* on an edge server. If *edgeproxyaddress*, *userVerificationlevel* (which you can find listed under *Get-OCSIMProviders*), or both, are specified, they will be modified.

> **Warning** There is no "*=false*" for *-enabled* and *-Ispublic*. So, if they are not specified, they are set to false, even if they were previously set to true.

Remove-OCSEdgeIMProvider

Required Parameter: *-Name*

Optional Parameters: -*Server* This function removes the IM provider specified by *Name* from the Access Edge Server. If no server parameter is specified, this function runs against the local server.

Edge Server: Internal Domains

The following functions can be used for managing internal (sip) domains in an Office Communications Server 2007 R2 environment.

Get-OCSEdgeInternalDomain

Optional Parameters: -*Server*

This function returns the internal Session Initiation Protocol (SIP) domains supported on an Access Edge Server (returned by the WMI object *MSFT_sipFederationInternalDomainData*). If no server parameter is specified, this function runs against the local server.

New-OCSEdgeInternalDomain

Required Parameter: -*Domain*

Optional Parameters: -*Server*

This function adds the specified domain to the list of internal SIP domains supported on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Remove-OCSEdgeInternalDomain

Required Parameter: -*Domain*

Optional Parameters: -*Server*

This function deletes the specified domain from the list of internal SIP domains supported on an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Edge Server: Internal Servers

The following functions can be used for managing the list of internal Office Communications Servers that an Access Edge Server trusts.

Get-OCSEdgeInternalServer

Optional Parameters: -*Server*

This function returns the internal Live Communications Server or Office Communications Server 2007 R2 servers authorized to use an Access Edge Server (returned by the WMI object *MSFT_SIPFederationInternalServerData*). If no server parameter is specified, this function runs against the local server

New-OCSEdgeInternalServer

Required Parameter: *-intServer*
Optional Parameters: *-Server*

This function adds an internal Live Communications Server or Office Communications Server 2007 R2 server to the list of servers authorized to use an Access Edge Server.
If no server parameter is specified, this function runs against the local server.

Remove-OCSEdgeInternalServer

Required Parameter: *-intServer*

Optional Parameters: *-Server*

This function removes an internal Live Communications Server or Office Communications Server 2007 R2 server from the list of trusted servers authorized to connect to an Access Edge Server. If no server parameter is specified, this function runs against the local server.

Events

The following functions can be used to query the event log for information from Office Communications Server events.

Get-OCSWarningEvent

This function displays a tabular view of events from the Office Communications Server event log that have an entry type of *Warning*. This function runs against the local server only.

Get-OCSErrorEvent

This function displays a tabular view of events from the Office Communications Server event log that have an entry type of *Error*. This function runs against the local server only.

Location Profiles

The following functions can be used for managing Enterprise Voice location profiles in an Office Communications Server 2007 R2 environment. See also *Add-OCSNormalizationRuleToOCSLocationProfile and Remove-*

OCSNormalizationRuleFromOcsLocationProfile in the "Normalization Rules" section later in this appendix.

Get-OCSLocationProfile

Optional Parameters: *-Server*, *-name*

This function returns a list of Enterprise Voice location profiles (from the WMI class *MSFT_SIPLocationProfileData*). If no name is specified, then all location profiles are returned. The name can use a wild card or specify a unique name. If no server parameter is specified, this function runs against the local server.

Choose-OCSLocationProfile

Optional Parameter: *-Server*

This function returns *MSFT_SIPLocationProfileData* objects from the list returned by *Get-OCSLocationProfile*. If there is only one object in this list, it is returned; otherwise, a list is presented to allow one to be chosen. If no server parameter is specified, this function runs against the local server.

New-OCSLocationProfile

Required Parameters: -Name, -Description –Rules

Optional Parameters: -Server

This function creates a new Location Profile, with the given name, and description (which is labeled 'Display text' in the Office Communications Server Voice Properties, Add…, Add Location Profile dialog when adding a new profile, and is required). A profile must have at least one normalization rule. There are two ways of passing the rules, either as 'rule' WMI objects or by name. If no server parameter is specified, this function runs against the local server.

Examples

The following two examples are equivalent. One stores the rule object in a variable, the other passes the rule name.

```
$rule=Get-OCSNormalizationrule –name "New-York" ;
New-OCSNormalizationProfile –name "Wall-St" –rule $rule –Description "Wall St, New York, US"
```

```
New-OCSNormalizationProfile –name "Wall-St" –rule "New-York" –Description "Wall St, New
York, US"
```

Remove-OCSLocationProfile

Required Parameter: *-Name*

Optional Parameter: *-Server*

This function deletes the named location profile. If no server parameter is specified, this function runs against the local server.

Mediation Servers

The following functions can be used to work with Mediation Servers in an Office Communications Server 2007 R2 environment.

Get-OCSMediationServerSetting

Optional Parameters: *-Server*

This function returns the settings for a Mediation Server (from the WMI class *MSFT_SIPMediationServerConfigSetting*). If no server parameter is specified, this function runs against the local server. If the local (or specified) server is not a Mediation Server, this function returns an error.

Set-OCSMediationServerPlusSign

Optional Parmeters Server, -Remove, -Preserve

This function determines whether a Mediation server preserves or removes the + sign in an E.164 formatted number for compatibility with PBXs, which do not support the + sign. A mediation server name can be piped into the function or passed as a parameter. If no server parameter is specified, this function runs against the local server. The two switches, Remove and Preserve, toggle the server behavior. If the command is invoked with no parameters or only a target server via the *Server* parameter, the + sign is preserved.

> **Note** For more information see "Choose-OCSMediationSever" in the "Services" section later in this document.

Meeting Policies

The following functions can be used to manage policy settings for Web conferencing in an Office Communications Server 2007 R2 environment.

Get-OCSMeetingPolicy

Optional Parameters: *-Server, -Name*

This function returns a meeting policy objects (from the WMI class *MSFT_SIPGlobalMeetingPolicyData*). The *Name* parameter filters the meeting polices, to the named policy or policies if used with a wild card. If no server parameter is specified, this function runs against the local server.

Choose-OCSMeetingPolicy

Optional Parameters: *-Server*

This function calls *Get-OCSMeetingPolicy* to get a collection of *MSFT_SIPGlobalMeetingPolicyData* objects. If there is only one meeting policy, the object is returned. If there is more than onemeeting policy, a list is displayed, the user is prompted to select one or more policies, and the function returns the selected object or objects.

If no server parameter is specified, this function runs against the local server.

New-OCSMeetingPolicy

Required Parameter: *-Name*

Optional Parameters: *-MeetingSize*, *-AllowAppSharingForExternalMeeting*, *-AllowIPAudio*, *-AllowIPVideo*, *-AllowPresenterToDelegateRecording*, *-AllowPresenterToRecord*, *-ColorDepth*, *-EnableAppDesktopSharing*, *-EnableDataCollaboration*, *-RetainPPTForExternalMeeting, -Server*

This function creates a new meeting policy with the given name. The parameters have defaults assigned as follows:

- Meeting Size: defaults to10.
- Allow App Sharing For External Meeting: can be *None* (the default), *SingleApplication*, or *Desktop*.
- Allow IP Audio: defaults to false.
- Allow IP Video: defaults to false.
- Color Depth: can be *'Gray shades'*, *'256'*, *'High colors'* (the default), or *'True Colors'*.
- Enable App Desktop Sharing: defaults to false.
- Enable Data Collaboration: defaults to false.

- Retain PPT For External Meeting: defaults to false.
- Allow Presenter To Record: defaults to false.
- Allow Presenter To Delegate Recording: defaults to false.

> **Note** In the Configure Office Communications Server Users Wizard, Configure User Settings, Meeting Policy, Add or Edit Policy, the setting to retain a PPT for External Meetings is labeled "Use native format for PowerPoint Files".

If no server parameter is specified, this function runs against the local server.

Remove-OCSMeetingPolicy

Required Parameter: *-Name*

This function deletes the named meeting policy.

Update-OCSMeetingPolicy

Required Parameter: *-Name*

Optional Parameters: *-MeetingSize*, *-AllowAppSharingForExternalMeeting*, *-AllowIPAudio*, *-AllowIPVideo*, *-AllowPresenterToDelegateRecording*, *-AllowPresenterToRecord*, *-ColorDepth*, *-EnableAppDesktopSharing*, *-EnableDataCollaboration*, *-RetainPPTForExternalMeeting, -Server*

This function updates the meeting policy with the given name. If parameters are not specified, they are left as they are. (They are not reset to the defaults used by *New-OCSMeeting*.)

If no server parameter is specified, this function runs against the local server.

Normalization Rules

The following functions can be used for working with Enterprise Voice normalization rules in an Office Communications Server 2007 R2 environment.

Get-OCSNormalizationRule

Optional Parameters: *-Server, -name*

This function returns a list of Enterprise Voice phone number normalization rules (from the WMI class *MSFT_SIPLocalNormalizationRuleData*). If *–Name* is specified, the list is filtered down to the name – which can include a wild card. If no server parameter is specified, this function runs against the local server.

Choose-OCSNormalizationRule

Optional Parameters: *-Server*

This function returns *MSFT_SIPLocalNormalizationRuleData* objects from the list returned by *Get-OCSNormalizationRule*. If there is only one object in this list, it is returned; otherwise, a list is presented to allow one item to be chosen.

If no server parameter is specified, this function runs against the local server.

New-OCSNormalizationRule

Required Parameters: *-Name*, *-pattern*, *-translation*

Optional Parameter: *-Description, server*

This function creates a new Enterprise Voice normalization rule. A rule does not need to be bound to any location profiles and the new- command doesn't make the link between a normalization rule and a location profile. Use the *Add-* function shown later in this list. If no server parameter is specified, this function runs against the local server.

Pattern is a regular expression for recognizing the number, and *translation* is the string into which the recognized number is inserted. The regular expression will usually begin with the marker ^ and end with the marker $. In between, it can have \+, which designates a plus sign; [0..9] or \d , which designates a digit; * , which designates any number of digits, including 0; +, which designates at least one match; and ?, which designates zero or one matches. {n} represents exact n matches, {n,} is at least n matches, and {n,m} is between n and m matches. Finally, segments in brackets are passed on to the next stage as $1, $2, and so on.

> **Note** PowerShell expands variables inside double quoted strings, so be sure to use single quotes to prevent $1 being converted.

Examples

Change a 7 digit local number to a Seattle area US number beginning with +1425.

```
New-ocsNormalizationRule –Name 'Seattle Local area' –pattern '^(\d{7})$'
-Translation '+1425$1'
```

Remove the leading 0 from a malformed UK number.

```
New-ocsNormalizationRule –Name 'UK Leading 0' – pattern '^\+440(\d+)$'
-Translation '+44$1'
```

Update-OCSNormalizationRule

Optional Parameters: *-Name, -Description, -pattern, -translation, Server*

This function edits a normalization rule, following the same pattern as creating a new one. If no server parameter is specified, this function runs against the local server. Rules can be passed to this function using the pipe, but it does not support wild cards for rule names.

Example

```
Get-OcsNormalizationRule | where {$_.description –eq $null} | Update-OcsNomralizationRule -description “Initial Rule”
```

Add-OCSNormalizationRuleToOCSLocationProfile

Required Parameters: *-Location*

Optional Parameters: *-Rules, -Server*

This function links one or more normalization rules to a location profile. There are two ways you can pass the *Rules* and *Location* parameters: either by passing the object or by passing theiname (and letting the function get the objects for you). You can pass an array of location objects, or a list of location profile names separated by commas. For the rules, there is a wild card, and you can name rules to take advantage of this. In addition, you can pipe rules into the function. If no server parameter is specified, this function runs against the local server.

Examples

```
Link-OcsNormalizationRuleToOcsLocationProfile –rules “Generic-US” –location “Wall-st”
```

```
Get-OCSNormalizationRule –Name “Generic-US” |
Link-OcsNormalizationRuleToOCSLocationProfile -location Broadway, Madison
```

Remove-OCSNormalizationRuleFromOCSLocationProfile

Required parameter: Rule

Optional Parameters; -Location, -Server

This function removes a single Normalization rule from one or more location profiles. If no location profile is passed, all profiles are checked, and those which are linked to the rule will be unlinked. If a name is passed as the location parameter, then it can be a wild card or an array of names. Alternatively, WMI objects representing the location profiles can be passed. The rule can either be an object representing the Normalization rule or the name of the rule.

Remove-OCSNormalizationRule

Optional Parameters: -Rule, -Server

This function deletes the rule that is passed to it. The rule can be passed through the pipe, or as a parameter, either using the name of the rule or the WMI object that represents it. Because a Normalization rule cannot be deleted if it is linked to a location profile, the function calls `Remove-OCSNormalizationRuleFromOcsLocationProfile` to remove any links. Note that removing the last rule from a location profile is not a valid configuration. If no server parameter is specified, this function runs against the local server.

Phone Routes

The following functions can be used for managing phone routes in an Office Communications Server 2007 R2 environment. For more information about Office Communications Server 2007 R2 Enterprise Voice concepts, refer to Chapter 11, "VoIP Scenarios."

Get-OCSPhoneRoute

Optional Parameters: *-name; Server*

This function returns a list of Enterprise Voice Phone Routes (from the WMI class *MSFT_SIPPhoneRouteData*). If the name parameter is specified, the list is filtered to that name. If no server parameter is specified, this function runs against the local server.

Choose-OCSPhoneRoute

Optional Parameters: *-Server*

This function returns *MSFT_SIPPhoneRouteData* objects from the list returned by *Get-OCSPhoneRoute*. If there is only one object in this list, it is returned; otherwise, a list is presented to allow one object to be chosen. If no server parameter is specified, this function runs against the local server.

New-OCSPhoneRoute

Required Parameters: -Name, -GatewayList, -PhoneRouteUsages, -TargetPhoneNumber

Optional Parameter: -description

This function creates a new route. A route needs a name, a regular expression to identify target phone numbers, and one or more associated gateways. In addition, a route cannot be created without being associated with at least one route usage.

Example

```
New-OCSphoneRoute -name "MyNewRoute" -gatewayList (Choose-OcsMediationServer)`
-PhoneRouteUsages(Choose-OcsPhoneRouteUsages) -targetPhoneNumber ^650
```

Remove-OCSPhoneRoute

Required Parameter: *-Name*

Optional Parameters: *-Server*

This function deletes the named route. If no server parameter is specified, this function runs against the local server.

Update-OCSPhoneRoute

Optional Parameters: *-Name, -PhoneRouteUsages, -TargetPhoneNumber, -description*

This function updates OCSPhone routes; the parameters have the same meaning as they do in the New-OcsPhoneRoute Function. The name does not support wild cards; however multiple routes can be piped into the function. At present, it does not support changing the gateways used.

Get-OCSPhoneRouteForOCSPhoneRouteUsage

Optional Parameters: *-usage, -Server*

This function gets phone route objects that are associated with one or more usages. The usage can be either a phone route usage object, an array of these objects, or a string containing the name of the usage. Alternatively, one or more phone route usages can be piped into the function. If no server parameter is specified, this function runs against the local server.

Get-OCSPhoneRouteForOCSUser

Required Parameter: *-URI*

This function takes a URI representing a user and gets the policy associated with the user (using *Get-OCSPolicyForUser*). It then gets the usages associated with that policy (using *GetOCSUsageForOCSUCPolicies*), and gets the routes associated with those usages (using *Get-OcsPhoneRouteForOCSPhoneUsage*). If the *SIP:* portion is not specified in the URI, it will be added.

Get-OCSPhoneRouteForNumber

Required Parameter: *-Number*

Optional Parameter: *-Routes*

This function gets the phone routes that can be used to reach a particular number. If the *routes* parameter is not specified, the function checks against all routes. If routes are passed to the function, only those routes are checked for a match.

Phone Route Usages

The following functions can be used for working with phone route usage objects in an Office Communications Server 2007 R2 environment.

Get-OCSPhoneRouteUsage

Optional Parameters: *-name, -Server*

This function returns a list of Unified Communications (UC) phone route usages (from the WMI class *MSFT_SIPPhoneRouteUsageData*). If the name parameter is specified, the list is filtered to the named route. A wild card can be used in the name to select multiple routes. If no server parameter is specified, this function runs against the local server.

Choose-OCSPhoneRouteUsage

Optional Parameters: *-Server*

This function returns *MSFT_SIPPhoneRouteUsageData* objects from the list returned by *Get-OCSUCPhoneRouteUsage*. If there is only one object in this list, it is returned; otherwise, a list of objects is presented to allow one to be chosen. If no server parameter is specified, this function runs against the local server.

New-OCSPhoneRouteUsage

Required Parameter: *-Name*

Optional Parameter: *-Description, -Server*

This function creates a new phone route usage and sets the Default Usage parameter to false. It doesn't add any routes to the usage. (See the *Add-OCSPhoneRouteUsageForOCSUCPolicy* function that appears later in this list.) If no server parameter is specified, this function runs against the local server.

Remove-OCSPhoneRouteUsage

Required Parameter: *-Name*

Optional Parameters: *-Server*

This function deletes the named phone route usage. If no server parameter is specified, this function runs against the local server.

Get-OCSPhoneRouteUsageForOCSUCPolicy

Optional Parameters: *-policy, -Server*

This function returns the phone route usage or usages associated with a UC policy. Policy can be passed as a parameter (either using the name of the policy or the WMI object which represents it), or can be passed through using the pipe command. If no server parameter is specified, this function runs against the local server.

Example

The following 2 examples produce the same result:

```
Get-OcSPhoneRouteUsageForOCSUCPolicy -policy (choose-ocsUCpolicy)
```

```
Choose-ocsUCpolicy | Get-OcSPhoneRouteUsageForOCSUCPolicy
```

Add-OCSPhoneRouteUsageToOCSPhoneRoute

Required Parameters: *-Usages*, *-Route*

Optional Parameter: *-Server*

This function links one or more usages to one or more phone route objects. There are two ways you can pass the parameters: either by passing the object or objects or by passing their names (and letting the function get the objects for you). For the usages, there is an automatic wild card and you can name phone usage objects to take advantage of this. Routes can use a wild card if explicitly specified. If objects are passed, the function allows single or multiple objects to be passed. If no server parameter is specified, this function runs against the local server.

Add-OCSPhoneRouteUsageToOCSUCPolicy

Required Parameters: *-Usages*, *-Route*

Optional Parameter: *-Server*

This function links one or more usages to one or more UC policy objects. There are two ways you can pass the parameters: either by passing the object or objects, or by passing their names (and letting the function get the objects for you). For the usages, there is an automatic wild card, and you can name phone usage objects to take advantage of this. UC Policies can use a wild card, if explicitly specified. If objects are passed, the function allows single or

multiple objects to be passed. If no server parameter is specified, this function runs against the local server.

Remove-OCSPhoneRouteUsageFromOCSPhoneRoute

Required Parameter: *-Usage*

Optional Parameter *–Route, -Server*

This function unlinks a single phone route usage from one or more OCS phone routes. The *usage* parameter can be passed as an object or as a string containing its name, as well as the parameter. The *route* parameter supports wild cards but the *usage* parameter does not. If no route is specified, the usage is removed from all routes. If no server parameter is specified, this function runs against the local server.

Remove-OCSPhoneRouteUsageFromOCSUCPolicy

Required Parameter: *-Usage*

Optional Parameter *–Policy, -Server*

This function unlinks a single phone route usage from one or more UC Policies. The usage or the policy can be passed as an object or as a string containing its name. The policy parameter supports wild cards, but the usage parameter does not. If no policy is specified, the usage is removed from all Policies. If no server parameter is specified, this function runs against the local server.

Pools

The following functions can be used for managing pools in an Office Communications Server 2007 R2 environment. Internally, a Standard edition server is considered to be a pool.

Get-OCSPool

Optional Parameters: *-name, -Server*

This function returns pool settings (from the WMI class *MSFT_SIPPoolSetting*). If a name is specified, the list of pools is filtered down to the matching name or names: wild cards are supported in the name. If no server parameter is specified, this function runs against the local server.

Choose-OCSPool

Optional Parameters: *-Server*

This function returns a single *MSFT_SIPPoolSetting* object from the list returned by *Get-OCSPoolSetting*. If there is only one object in this list, it is returned; otherwise, a list of objects is presented to allow one to be chosen. If no server parameter is specified, this function runs against the local server.

Start-OCSReplication

Required Parameter: *-Pool*

This function starts the replication of Active Directory information to the specified pool. It is not designed to run remotely.

SIP Domains

The following functions can be used to manage the list of SIP domains for which an Office Communications Server 2007 R2 environment is authoritative.

Get-OCSSIPDomain

Optional Parameters: *-Name, Server*

This function returns SIP domains supported in an Office Communications Server deployment (from the WMI class *MSFT_SIPDomainData*). If no server parameter is specified, this function runs against the local server.

If a name is specified, the function returns the matching SIP domain objects, otherwise it returns all known SIP domains. Wild cards are supported in the name.

Examples

```
Get-OCSSIPDomain
Get-OCSSipDomain –Server SE.litwareinc.com
Get-OcsSipDomain litware*
```

New-OCSSIPDomain

Required Parameter: *-Domain*

OptionalParameter: - *Server*

This function adds a SIP domain to the list of SIP domains supported on an Office Communications Server deployment. If you need a new domain to be the default domain, this should be set through the Office Communications Server Global Properties, General Tab. If no server parameter is specified, this function runs against the local server.

Examples

```
New-OCSSIPDomain “LitwareInc.Co.Uk”

New-OCSSIPDomain –domain “LitwareInc.Co.Uk” –server se.litwareInc.com
```

Remove-OCSSIPDomain

Required Parameter: *-Domain*

OptionalParameter: - *Server*

This function deletes a SIP domain from the list of SIP domains supported on an Office Communications Server2007 R2 deployment. If you need to delete the default domain, you must set a new domain to be the default through the Office Communications Server Global Properties, General Tab first. If no server parameter is specified, this function runs against the local server.

Examples

```
Remove-OCSSIPDomain “LitwareInc.Co.Uk”
Remove-OCSSIPDomain –domain “LitwareInc.Co.Uk” –server se.litwareInc.com
```

Services

The following functions can be used for managing services in an Office Communications Server 2007 R2 environment.

Get-OCSInstalledService

Optional Parameter: *-Server*

This function returns the Office Communications Server 2007 R2 Service components (from the WMI class *MSFT_SIPServerInstalledComponentData*). If no server parameter is specified, this function runs against the local server.

Get-OCSWindowsService

Optional Parameter: *-Server*

This function returns a list of *win32_service* objects for the Real-Time Communications (RTC) services. If no server parameter is specified, this function runs against the local server.

Start-OCSWindowsService

Optional Parameters: *-Server*

This function attempts to start all the services returned by *Get-OCSWindowService*. If no server parameter is specified, this function runs against the local server.

Stop-OCSwindowsService

Optional Parameter: *-Server*

This function attempts to stop all the services returned by *Get-OCSWindowService*. If no server parameter is specified, this function runs against the local server.

Get-OCSTrustedService

Optional Parameter: *-Server*

This function returns a list of trusted services (from the WMI class *MSFT_SIPTrustedServiceSetting*). If no server parameter is specified, this function runs against the local server.

Choose-OCSMediationServer

Optional Parameter: *-Server*

This function returns a single *MSFT_SIPTrustedServiceSetting* object from the list returned by *Get-OCSTrustedService*. The list is filtered down to objects with a type of Mediation Server. If there is only one object in this list, it is returned; otherwise, a list of objects is presented to allow one to be chosen. If no server parameter is specified, this function runs against the local server.

UC Policies

The following functions can be used for managing Unified Communications policies in an Office Communications Server 2007 R2 environment. (For more information, see the *Add-OCSPhoneRouteUsageToOCSPhoneRoute* function in the "Phone Route Usages" section, earlier in this appendix.)

Get-OCSUCPolicy

Optional Parameters: *-name, -Server*

This function returns a list of UC policies (from the WMI class *MSFT_SIPGlobalUCPolicyData*). If a name is specified, then the function returns the matching policy objects; otherwise it returns all policies. Wild cards are supported in the name.

If no server parameter is specified, this function runs against the local server.

Choose-OCSUCPolicy

Optional Parameter: *-Server*

This function returns *MSFT_SIPGlobalUCPolicyData* objects from the list returned by *Get-OCSUCPolicy*. If there is only one object in this list, it is returned; otherwise, a list of objects is presented to allow one to be chosen.

If no server parameter is specified, this function runs against the local server.

New-OCSUCPolicy

Required Parameter: *-Name*

Optional Parameters: *-AllowSimultaneousRinging, -Server*

This function creates a new UC policy object. If the *-AllowSimultaneousRinging* switch is used, the policy will allow call forking. No usages are linked to the policy. (For more information, see the *Add-OCSPhoneUsageToOCSUCPolicy* item shown earlier in this appendix.) The newly created policy is set not to be the default. If no server parameter is specified, this function runs against the local server.

Example

```
New-OCSUCPolicy “Admin Policy” – AllowSimultaneousRinging
```

Remove-OCSUCPolicy

Required Parameter: *-Name*

Optional Parameter: *Server*

This function deletes the named UC policy. If no server parameter is specified, this function runs against the local server.

Example

```
Remove-OCSUCPolicy “Admin Policy”
```

Update-OCSUCPolicy

Optional Parameters -Name, -AllowSimultaneousRinging, -defaultPolicy, -Server

This function updates a UC policy. Policies are either piped into the function or the name of a single policy is passed as a parameter (the name does not support the use of wild cards). The policy can be set as the Default or to allow Simultaneous ringing by passing either $true or $false as values for the parameters. If no server parameter is specified, this function runs against the local server.

Get-OCSUCPolicyForOCSUser

Optional parameters: *-User, -Server*

This function returns the UC policy object associated with one or more users. It checks, using *Get-OCSGlobalUCSetting*, to see whether a global policy is in force. If a global policy is in force or no policy has been set for the user, it will return the global policy; otherwise, it will return the one specified for the user. One or more users may be passed into the function using the pipe or using the parameter, using either the WMI object representing the user or the users SIP URI. (which is passed to Get-OCSUser). The parameter supports single or multiple objects and wild cards. If no server parameter is specified, this function runs against the local server.

Example

```
Get-OCSUCPolicyForOCSUser bart@litwareinc.com
```

Get-OCSGlobalUCSetting

Optional Parameter: *-Server*

This function returns the information from the object *MSFT_sipGlobalUCSetting*.

If no server parameter is specified, this function runs against the local server.

Users

The following functions can be used for managing users in an Office Communications Server 2007 R2 environment.

Get-OCSUser

Optional Parameters: -URI, -Condition, -Server

This function returns a list of presence-enabled users (from the WMI classMsft_sipesusersetting). If no condition is specified, all enabled users are listed. The condition is enclosed in quotes and written in WMI syntax, not PowerShell syntax. To avoid the need to write a condition in the form “PrimaryURI=sip:user@domain”, the function will also accept a URI parameter. This parameter can include a wild card, and if the SIP: is omitted, it will be inserted automatically. If no server parameter is specified, this function runs against the local server.

Examples

```
Get-OCSUser –condition “PublicNetworkEnabled = 'TRUE' "
```

```
Get-OCSUser –URI 'SIP:bart@litwareonc.com'
```

Get-OCSUserDetail

Optional Parameters: *-Condition, -Server*

This function displays a formatted list of UC users, showing their *DisplayName*, *URI*, and whether they are enabled. If a user is enabled, the function shows whether they are enabled for enhanced presence, enabled for remote access, enabled for federation, enabled for public IM connectivity, or enabled for voice. It also displays the Unified Communications Policy and Meeting Policy applied to them.

The data is returned by *Get-OCSUser*, and the *-Condition* parameter is passed straight through to it. (See the *Get-OCSUser* function earlier in this list for more details.) If no server parameter is specified, this function runs against the local server.

Get-OCSPICUsersCount

Optional Parameter: - *Server*

This function returns a count of users enabled for public IM connectivity. The data to be counted is returned by a call to *Get-OCSUser*. If no server parameter is specified, this function runs against the local server

New-OCSuser

Required Parameters: *-User*, *-HomeServer*
Optional parameters: *URI, Server*

This function returns one or more newly enabled Office Communications Server 2007 R2 user objects. It accepts either an Active Directory object representing the user or their Active Directory distinguished name, and a WMI object representing the requested home server or the Home Server's Active Directory distinguished name (not its FQDN) and the user's SIP URI. If no URI is specified, then the user's e-mail address is used as their URI. If the domain in the URI is not a supported SIP domain, a warning is issued, but the process continues. If no server parameter is specified, this function runs against the local server.

Examples

```
Get-adUser –filter “givenName=Jo*” | new-OcsUser –homeServer (choose-ocspool)
```

```
New-ocsUser –user "CN=Sidney Higa,CN=users,DC=litwareinc, DC=com" –homeServer "CN=LC
Service,CN=Microsoft,CN=SE,CN=Pools CN=RTCServices,CN=Configuration,DC=litwareinc,Dc=com"
-uri "sip:sidhig@Litwareinc.co.uk"
```

Remove-OCSUSer

Required Parameters: *-URI*
Optional parameters: *-Server*

This function removes the users specified by the URI. It fetches the user objects identified by the URI, and calls its delete method. You can build more complex selection processes and call the delete() method, although care is needed because **the delete process is not easily reversible**. Remove-OcsUser * will remove all users without warning. If no server parameter is specified, this function runs against the local server.

Update-OCSUser

Optional Parameters: Condition, -URI –userDN , Server, -AllowOrganizeMeetingWithAnonymousParticipants, -ArchiveFederatedCommunications, -ArchiveInternalCommunications, -EnabledForEnhancedPresence, -EnabledForFederation, -EnabledForInternetAccess, -HomeServerDN, -LineURI, -MeetingPolicy, -PublicNetworkEnabled, -RemoteCallControlTelephonyEnabled, -UCPolicy

This function accepts input from the pipe command, which can either be an OCSUser object or an Active Directory object. It will also accept the Active Directory distinguished name for the user in the userDN parameter or a SIP URI in the URI parameter. Finally, it can retrieve users matching the condition parameter. (See the Get-OCSUser function earlier in this appendix for more details.) The other parameter names correspond to the WMI field names shown in Get-OCSUser and are used as column headings in Import-OCSUser and Export-OCSUser. If no server parameter is specified, this function runs against the local server.

Examples

```
Get-adUser –filter “givenName=Jo*” |
Update-OcsUser –EnabledForFederation $true
```

```
Get-OcsUser –Condition "PublicNetworkEnabled=true" |
Update-OcsUser –EnabledForFederation $true
```

(this example can be shortened, as the following one shows)

```
Update-OcsUser –condition "enabledForFederation=true" -EnabledForInternetAccessTrue
```

```
Update-OcsUser –URI Sip:sidHig@litware.co.uk  –enabledForInternetAccess $true
```

Import-OCSUser

Required Parameter: *-Path*

This function imports Office Communications Server 2007 R2 user data from a CSV file specified by path. For each line in the CSV, the logic is as follows:

1. Check to see if the user already exists. If not, and if the enabled parameter is true, the *New-OCSUser* function is called to create the user.

2. Check to see if the enabled parameter in the file is set to false. If so, the user is deleted; otherwise, all the settings are passed to *Update-OCSUser*.

Export-OCSUser

Required Parameter: *-Path*

Optional Parameters: *-Condition, -Server*

This function exports Office Communications Server user data to a CSV file specified by path. The CSV file can be modified and re-imported.

The data is returned by *Get-OCSUser*, and the *-Condition* parameter is passed straight through to it. If no condition is specified, all enabled users are exported. If no server parameter is specified, this function runs against the local server.

Enable-ExchUmForOCSUser

Optional Parameters -filter, -exchUmPolicy, -runNow

This command is different from the other functions in this script set. It's designed to run on Exchange 2007, not Office Communications Server 2007 R2. It retrieves users from Active Directory who are enabled for Enterprise Voice in Office Communications Server, but who are not enabled for Exchange Unified Messaging. It generates the commands to run in Exchange PowerShell to enable them.

The users selected from Active Directory can be restricted with the filter switch specifying an LDAP term (which must be enclosed in brackets). If no filter is specified, all users will be processed. An Exchange UM Mailbox policy name can be specified, but if none is provided, the function will look for an Exchange dial plan that matches the name of the Office Communications Server Location profile. If the dial plan is found, and it has a single UM Mailbox policy associated with it, the user is enrolled into UM using that policy. If multiple policies are associated with a dialplan, a choice will be displayed (in this case it is more effective to filter the users by Office Communications Server Location Profile and specify the Exchange UM policy). By default, Exchange will generate a random PIN of the required length and mail it to the user.

This command outputs the commands to be run in Exchange PowerShell. If it is run from within Exchange PowerShell, the –RunNow switch invokes the commands;, however if you decide to run this on Office Communications Server 2007 R2, the output can be redirected to a .PS1 file to be run as a script from within Exchange 2007's PowerShell in a two step process

PowerShell Commands Used by the Functions

The following is a list describing different PowerShell commands used by the functions contained in the OCS-R2.PS1 script.

Variables

PowerShell variables are prefixed with a dollar sign ($), and values are assigned with the equals sign (=). For example:

```
$users = Get-OCSUser
```

There are some special variables, notably $_, which represents the current pipeline object. In addition, $( ) is also used as a command expansion operator.

param

In a function declaration, *param* defines the parameters to be passed. It can define their type, what their value should default to or an action to take if they are not passed. For example, in the function name *get-OCSPhoneRouteForNumber*, we have the following:

```
Param([String]$Number=$(throw "You must specify a Number"),
       $routes=$(get-ocsphoneRoute ))
```

Get-WMIObject

Get-WMIObject returns either a single management object or an array of objects. For example, the command

```
get-wmiobject –list
```

returns a list of available WMI objects on the local machine. All those that are related to Office Communications Server, or the earlier Live Communications Server versions , have class names that begin with *MSFT_SIP*. So you can fetch a list of the interesting objects with the following:

```
get-wmiobject -list | where-object {$_.__CLASS -like "MSFT_SIP*"}
```

There are three ways to get information with *Get-WMIObject*.

The first example is to use the *-query* switch, which allows WMI to be queried with SQL syntax:

```
Get-WmiObject -query "select * from MSFT_SIPDomainData where
    address='$domain' "
```

Note that in PowerShell, when a string is enclosed in double quotation marks, variables will be substituted into the string. So if *$domain* contains Microsoft.com, the query is executed as

```
select * from MSFT_SIPDomainData where address='Microsoft.com'
```

Secondly, you can select all the objects of a given class by running a query without a where condition or use the *-class* switch. For example:

```
Get-WmiObject -class MSFT_SIPDomainData
```

Finally, you can explore WMI objects with the PowerShell *Get-Member* commands (aliased as GM). For example:

```
Get-WmiObject -class MSFT_SIPDomainData | GM
```

Typically, objects for working with Office Communications Server have a *DELETE()* method to remove them and a *PUT()* method to save them when properties have been changed; the underlying code checks for validity when running a *PUT()* method and generates an error if the values being set break any consistency rules.

To access WMI objects on a remote computer, *Get-WMIObject* can be used with the *-computername* and *-credential* switches.

To create a new instance of a WMI object, there are two syntaxes that work in the same way:

```
$oInstance=([wmiclass]"\\.\root\cimv2:MSFT_SIPLocationProfileData")
    .createinstance()
```

and

```
new-Object System.Management.ManagementClass("MSFT_SIPDomainData"))
    .CreateInstance()
```

The \\.\ in the first example identifies the local server. This can be replaced with any server name. In the second example, the local server is the only choice available.

New-Object

New-object is used to create a new .NET or COM object. Only two object classes are used in the scripts we have provided: *System.management.managementClass*, for working with WMI,

and *directoryServices.DirectorySearcher*, for working with the Active Directory Services Interface (ADSI).

The class *System.management.managementClass* is passed the path to a WMI object class, while *directoryServices.DirectorySearcher* is passed an Active Directory path.

Get-EventLog

This function takes a *-LogName* switch to select the log required and then returns the events in the specified log.

If...else

PowerShell uses the following syntax:

```
 if (condition) {code block}

ElseIf (condition) {Code block}

Else {code block}
```

One thing that might confuse new users of PowerShell is the way conditions are written. For example:

```
($x –eq $y) rather than ($x =$ y)

($a –gt $b) rather than ($a > $b)

$x –eq $null rather than (isnull($x) or ($x is null)
```

In addition to the usual equal (*-eq*), not equal (*-ne*), greater than (*-gt*), and less than (*-lt*) functions, PowerShell supports *-like* (for wild cards), *-match* (for comparison with regular expressions), and *-is* for type checking. The types *$a*, *-is*, *[Strring]* (or any other type in square brackets) are useful for checking what type of variable you have been passed. Here is an example:

```
if { $pools –is [array]) {select the right element}

else {return the only element}
```

Where-object

The *where-object* function is used to filter an array of objects. For example:

```
$routes | where-object {$number -match $_.TargetPhoneNumber}
```

Unlike the IF statement, where the condition is enclosed in normal brackets and the code block to run is enclosed in braces, *where* accepts a code block in braces. If it evaluates to true,

the object being examined is passed on to the next processing stage. If it evaluates to false, the object is discarded.

Note also that *where-object* has an alias of WHERE, but this can lead the new PowerShell user into writing SQL style syntax. PowerShell users use an asterisk (*) for a wild card, not a percent sign (%. They also use *-eq* to mean "is equal to, rather than using an equals sign (=).

Measure-object

This function is used for counting the number of objects and producing aggregate results. For example, the following syntax counts the number of functions with "OCS" in their name:

```
dir function:*ocs*| measure-object
```

Select-object

This function selects the first or last *n* objects or isolates individual properties. For example, the *export-CSV* cmdlet outputs all the properties of the objects it is passed. So the redundant ones can be removed, as in the following example:

```
Get-OCSFederationPartner -| Select-object Domain, EdgeProxyAddress |

    export-csv $path
```

Format-table

One of the big advantages of using PowerShell is the ease of outputting information from arrays of objects in table or list format.

The *-autosize* switch allows *format-table* to size columns to fit the data, and the *-wrap* switch allows the data to spill over onto multiple lines.

The *-property* switch selects the information to display, and custom properties can be set up, as in the following example:

```
@{label='Members' ;Expression={$_.PoolMemberList}}
```

The *expression* code block can be pretty complex. Look at the code used in the *choose* functions shown earlier in this appendix.

ForEach-object

Most PowerShell cmdlets automatically carry out their task with an array of objects. However, some tasks need to be looped through. For example, if *$users* is a group of users to be

deleted, there is no *.delete* method for the whole array. So *foreach-object* is used instead, and *$_* is a special variable meaning "the current object," as shown in the following example:

```
$users| foreach-object {$_.Delete()}
```