MWI Normalization with Centralized Cisco Unity Connection

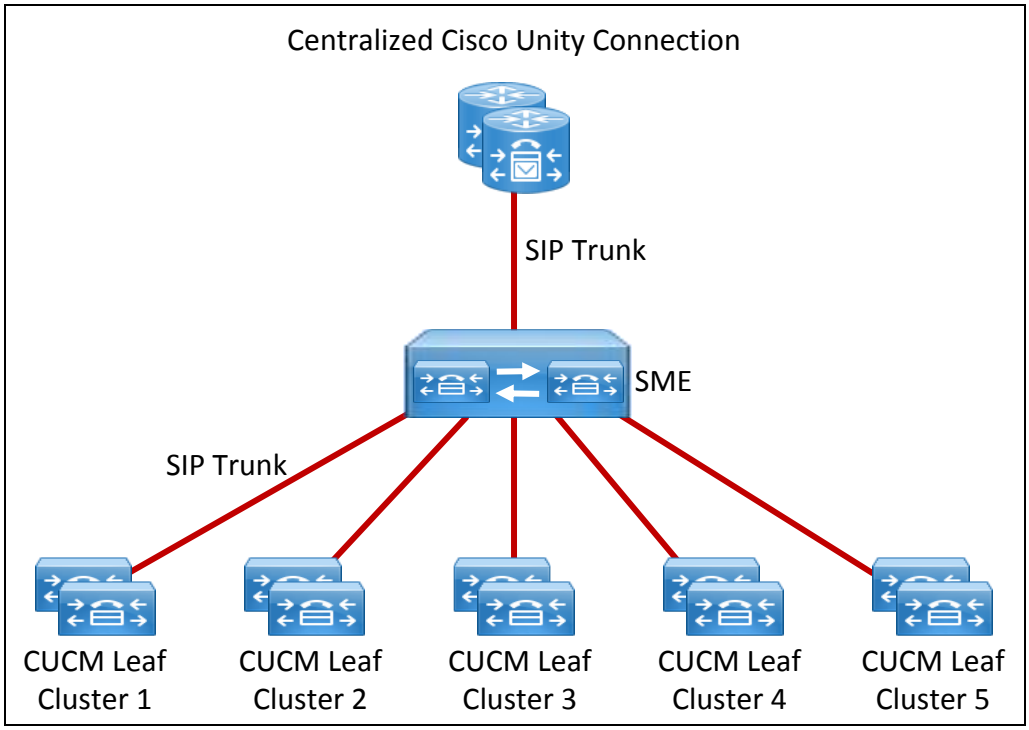
Written by: Chris Ward, Technical Marketing Engineer, Unity Connection

Table of Contents

Disclaimer

This script is not supported by the BU… Blah blah blah.

The Problem



The purpose of this document is to address a potential problem in deploying a centralized Cisco Unity Connection (CUC) deployment with a single SIP-based Phone System integration to a Cisco Unified Communications Manager (CUCM) Session Management Edition (SME) cluster that will be servicing more than one leaf clusters where the extensions configured on the user devices may not match the extensions configured in Cisco Unity Connection

We will leverage SIP Normalization scripts which were first added in CUCM version 8.0. These scripts will allow us to modify SIP MWI messages so that we can ensure that MWI functions as expected even with an over-lapping dial plan whilst still maintaining a single, centralized CUC-to-SME Phone System integration.

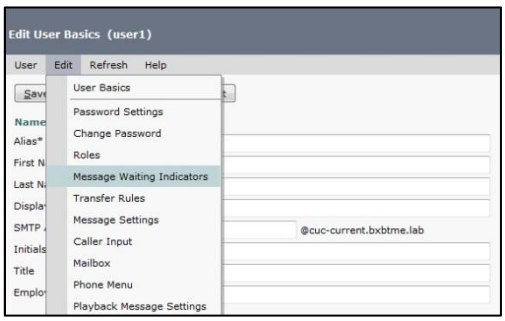
In this document we will touch on the basic requirements for making this type of centralized voicemail deployment function. For specific details on configuring SME for this type of deployment, please consult the SME Deployment Guide at:

http://www.cisco.com/en/US/docs/voice_ip_comm/cucm/session_mgmt/deploy/8_x/SME_Deployment_Guide.html#wp67376

Before we get into the proposed solution, let's talk about why this solution is necessary. In the following sub-sections, we will discuss how you could deploy CUC in a centralized manner and why this solution might be necessary.

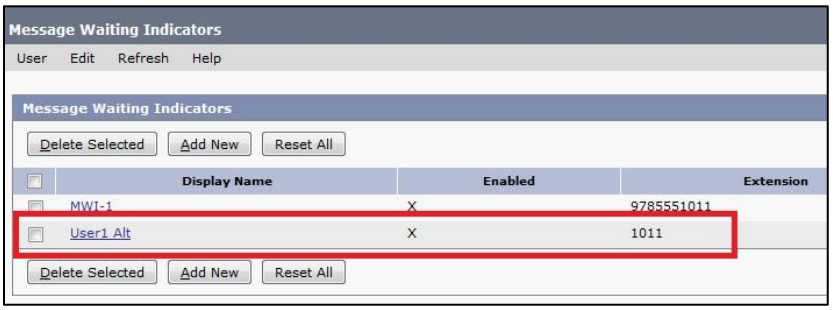
Avoiding Alternate MWI Numbers in Cisco Unity Connection

Within Unity Connection, there is the ability to specify alternate MWI numbers. This feature allows administrators to specify additional numbers that CUC will send MWI notifications to when CUC needs to change the MWI status of a device. While this is not a very involved process, as the number of users scales up, so will the administrative overhead. An administrator would need to add an alternate MWI number for each user and then any extension change would also require a change to their alternate MWI settings as well.



Alternate MWI menu location

Configure an Alternate MWI number



New Alternate MWI number

Using the method described in the next section, we don't need to configure any alternate MWI extensions for users. We can simply allow CUC to send MWI requests to the long-form or E.164 extension that we assigned to the user account and SME will take care of modifying the extension number within the SIP MWI request.

Overlapping Dial Plans

An overlapping dial plan is when you have the same extensions between two or more different clusters. This can also occur within the same cluster but for the purpose of this document, assume we are talking about different clusters. The concepts that we use to address different clusters can also be applied to over-lapping extensions within the same cluster.

For example you may have 2 DIDs from different area codes on different leaf clusters that share the same last 4 digits, such as 408-555-1011 (Leaf Cluster 1) and 978-555-1011 (Leaf Cluster 2). If you use full E.164 extensions or 10-digit extensions on the devices, then there is no problem. However, If you use the last 4 digits of these DIDs as your phones extensions then you would have two 1011 extensions,

one in Leaf Cluster 1 and one in Leaf Cluster 2. Site codes or DID dialing would make these extensions reachable, but there are considerations to make for voicemail.

Multiple Phone System Integrations

Within Unity Connection, we have the ability to handle duplicate extensions however, it would require that the extensions exist in different partitions and also that they belong to different phone system integrations within CUC. This would mean adding a second SIP trunk from CUCM/SME to CUC (using a different port) and dividing the CUC ports into two integrations. This would allow CUC to determine which phone system integration the call is coming in on and since each of our 1011 extensions would need to belong to separate phone system integrations, we could route calls to the appropriate account. MWI would also function properly in this environment.

If all you had were two sets of overlapping patterns like I previously mentioned, then all you would need is two phone system integrations, however, if you have rolled out 4 digit extensions to an entire enterprise and have tens or hundreds of overlapping extension ranges (even if their full DID or E.164 number is unique), then you would need more partitions in CUC and divide the pool of CUC ports into even smaller groups. This would become a functional and administrative challenge as the number of ports per phone system integration drops and the number of phone system integrations increase. It would probably also offset the gains you could get from centralizing your CUC off of an SME cluster.

Using Full DID or E.164 in CUC

We can use full DID or E.164 numbers in CUC which would eliminate the need for multiple phone system integrations and dividing the CUC ports into the different integrations. We would need to use Voice Mail Box Masks (as described previously) to route calls that are forwarded from short-form extension (i.e. 1011) to the appropriate account in CUC (i.e. 4085551011). This would allow for voicemail to function correctly, but MWI is a different story.

MWI, in this case, would be using the DID or E.164 number to trigger MWI, which in the case of overlapping extensions, wouldn't allow us to use alternate MWI numbers. The reason for this is because if we put alternate MWI numbers on both the 4085551011 and 9785551011 accounts in CUC of 1011, how will SME route the MWI request to the right leaf cluster with only a single phone system integration? The answer is the MWI request will go to the CUCM leaf cluster that is highest in priority in the incoming CSS of the SIP trunk from CUC to SME. So, if the pattern that matches 1011 in the incoming CSS of the SIP trunk is Cluster A (4085551011) and not Cluster B (9785551011) then only Cluster A's 1011 MWI could ever altered. Meaning Cluster B's 1011 would never be alerted to new VMs.

Using the configuration described in the following section, CUC would still use full DID or E.164 numbers for CUC accounts however, a SIP Normalization script will be placed on each of the "SME to Leaf Cluster" SIP trunks. This SIP Normalization Script will allow SME to modify SIP MWI NOTIFY messages to replace the full DID or E.164 extension with the local extension pattern on the destination leaf cluster.

The Solution – Normalizing MWI SIP NOTIFY Messages

How it Works

The basic operation of this solution can be broken down into a few steps. We assume that the centralized CUC has been deployed with 10-digit extensions in the user accounts and that the user's extensions on their devices are shorter which is why we require this solution.

1) A voicemail is left in a Cisco Unity Connection User's voicemail box
    a. Voicemail box was previously empty
    b. SIP Integration to CUC
2) CUC generates an SIP MWI notification
    a. SIP NOTIFY message with Request URI header of user's CUC extension
3) SME receives the MWI SIP NOTIFY
4) SME routes the MWI SIP NOTIFY message based on Request URI header (not the "To:" header)
5) SME modifies the MWI SIP NOTIFY using the SIP Normalization script
6) SME forwards the MWI SIP NOTIFY message to the leaf cluster (determined in step 4)
7) Leaf cluster CUCM receives the MWI SIP and processes the message
8) The user's device MWI light's up

In this document we centralize the SIP Normalization script on the SME and modify the MWI SIP NOTIFY messages after they are routed but before they are actually sent to the leaf cluster. The normalization script could also be run at the leaf cluster to incoming MWI SIP NOTIFY messages. You would just need to modify the script appropriately. The necessary modifications are outlined later in this section.

In the subsections that follow, we will go into how to actually implement the SIP Normalization script. These sections assume the following is already configured in your deployment:

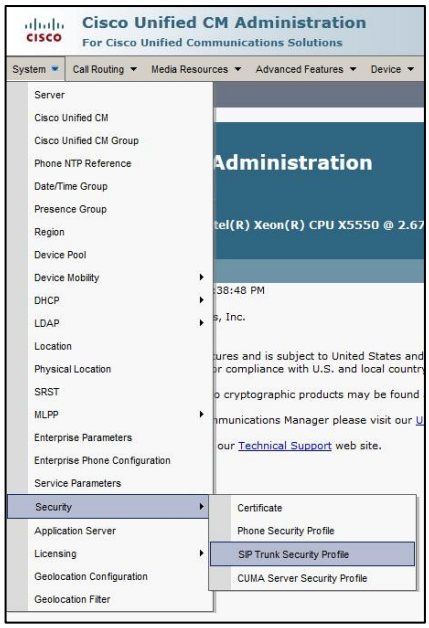
1) Cisco Unified Communications Manager Session Management Edition
    a. SIP Trunks configured to Leaf Clusters
    b. SIP Trunk configured to Cisco Unity Connection
2) Cisco Unity Connection with working SIP integration to SME
    a. Using 10-digit or E.164 extensions on user accounts
    b. Calls from all clusters should forward correctly to voicemail accounts
3) CUCM Leaf clusters with SIP trunks to SME
    a. All inter-cluster dialing should be functioning
    b. User's voicemail button should bring users to their login greeting

SIP Trunk Security Profiles

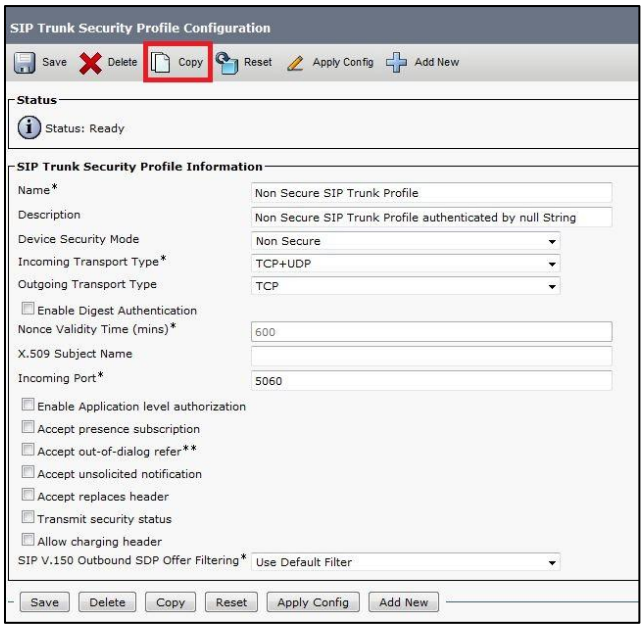
In order for SIP trunks to accept SIP MWI NOTIFY messages, we need to enable the "Accept unsolicited notification" option in the SIP Trunk Security Profile. Changing this setting allows a CUCM/SME to process the MWI SIP NOTIFY messages that it receives on its SIP trunks. The SME will receive these MWI SIP NOTIFY messages from CUC and the leaf CUCM clusters will be receiving these messages from the

SME. So this will need to be changed on all the leaf clusters' SIP trunks to SME and on the SIP trunk on SME that connects to CUC.

If you haven't already configured a separate SIP Trunk Security Profile for your SIP trunks, just open the "Non Secure SIP Trunk Profile" and then press the Copy button at the top of the configuration page.

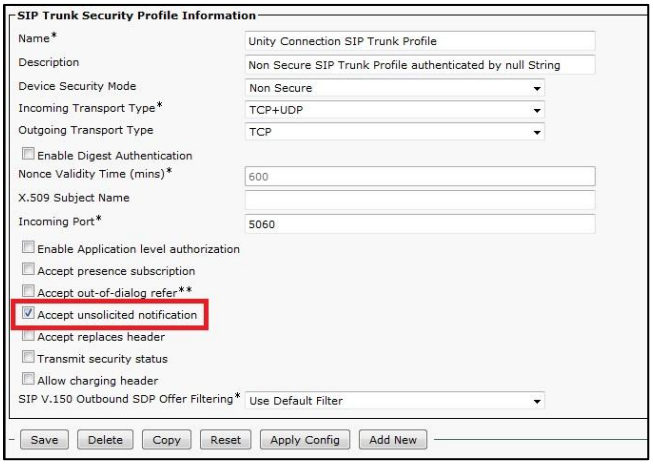


SIP Trunk Security Profile Location



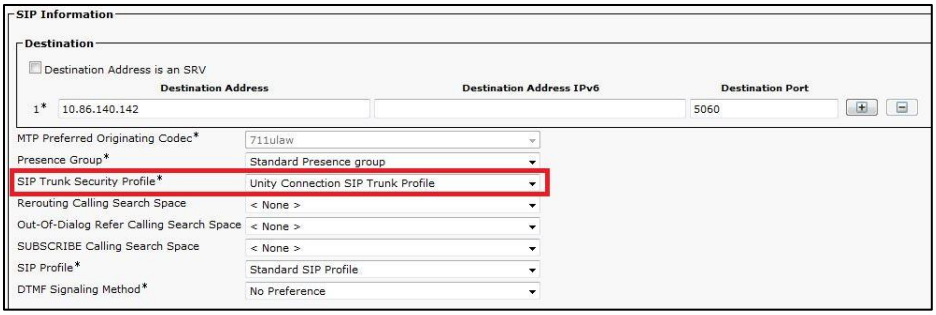
Copy the existing Non Secure SIP Trunk Profile

After you hit the Copy button, make a new name for the copied profile, check the "Accept unsolicited notification", and save the new profile.



Enable "Accept unsolicited notification"

Once you have a SIP Trunk Security Profile that has the "Accept unsolicited notification" enabled, you need to assign it to the SIP trunk you want to receive MWI SIP NOTIFY messages.



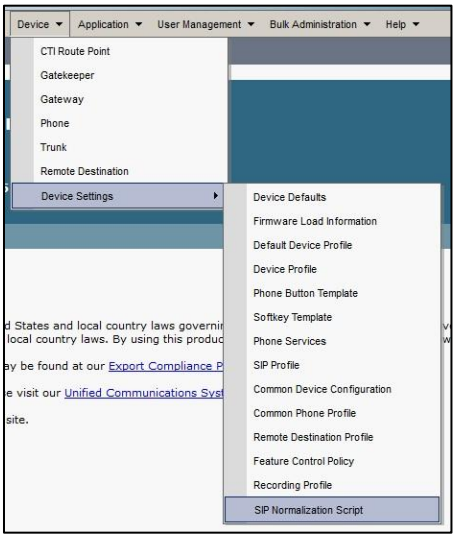
Assign the new SIP Trunk Security Profile to the SIP Trunk

Duplicate these steps on the remaining SIP trunks that interconnect the leaf clusters and the Unity Connection SIP Trunk on SME. If you have already created new SIP Trunk Security Profiles for these interconnecting trunks, ensure that the "Accept unsolicited notification" option is enabled.

MWI Normalization Script

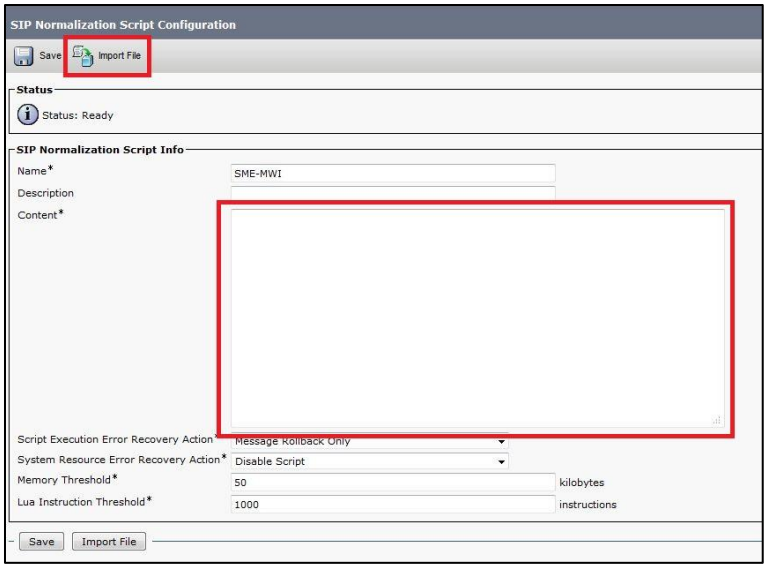
Loading the Script

The script provided for MWI normalization will only alter MWI SIP NOTIFY messages. The script is located at the end of this document for your reference and usage. This script will not affect any SIP NOTIFY messages that are part of a normal SIP conversation/call. This script can be loaded onto the SME cluster and then applied to all leaf cluster trunks and modify the MWI SIP NOTIFY messages as they leave the SME; or you can load the script on each of the leaf clusters and modify the MWI messages as they are received by the leaf clusters. From and administrative standpoint, it will be easier to load the script in one place, the SME, and just apply to the required trunks.



SIP Normalization Script menu location

You will want to select new, and give the script a name. You then have two methods of loading the script. You can either copy the script out this document or a .lua file that you copied it to and then paste it into the "Content" portion of the script page or you can load the .lua files using the "Import File" option.



Import the LUA file or copy the contents into the Content field

Either way, once the file is loaded, press "Save". You will notice after the script is saved that there is an option to reset. You will not need to immediately use this as the script isn't assigned to any SIP trunks yet. Once you assign the script to one or more SIP trunks, if you make a change to the script, you will need to reset the trunk in order for the changes to take effect. Also, if you decide you want to modify the script, the "Content" portion of the above page is where you will make changes. Common modifications are listed in the next sub-section.

Now that we have loaded the script, we need to assign it to the SIP trunks that will be passing MWI SIP Notify messages to the leaf clusters. Browse to the SIP Trunk that you are going to apply the script to and scroll all the way down to the "Normalization Script" section and select the script that was just created. After the script is selected we need to give a value for the "extensionMask" parameter. This parameter is the transformation mask that will be applied to the MWI messages as they leave SME cluster towards the leaf cluster.

Apply the normalization script to the SIP trunk and then add the extensionMask parameter

If we do not provide a value for the "extensionMask" parameter, then it defaults to "XXXX". In the screenshot above, if an MWI request came in for 4085551011, the output would be an MWI request for 601011. If there was no "extensionMask" parameter specified, then the output would have been a MWI request for "1011" as the default mask would have been used.

Modifying the Script

Within the script there are some options that you can configure, including whether you want to enable inbound interception of the MWI SIP NOTIFY or outbound interception. If you load the script on the SME

you will need to make sure the script is set to modify outbound SIP NOTIFY messages. Locate the function declaration at the top of the script by searching for “function M” and change accordingly:

Modify outgoing MWI SIP NOTIFY Messages (Script loaded on SME)

```
function M.outbound_NOTIFY(msg)
```

Modify incoming MWI SIP NOTIFY Messages (Script loaded on leaf cluster)

```
function M.inbound_NOTIFY(msg)
```

When referencing the script, please note that “--” represents a comment line and will not be processed as part of the script. Comments have been placed throughout the script to help users understand how the script works.

If you are experiencing issues with the script and want to troubleshoot, you will need to enable the trace output. Locate the current trace settings by searching for the “trace.disable()” function below and change accordingly:

Traces Disabled

```
trace.disable()
```

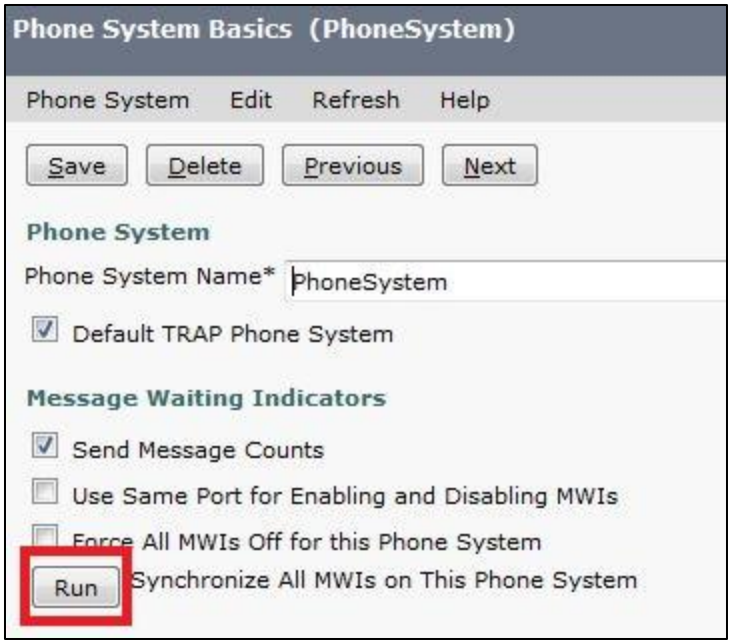
Traces Enabled

```
trace.enable()
```

With traces enabled, you will see outputs in the CCM SDI trace files that will help you validate that MWI SIP NOTIFY message is being intercepted and that the mask that you specified is being applied as expect. In the “CCM Trace Analysis” sub-section below, we will analyze some potential outputs from the CCM traces.

Verifying Operation

Now that the script is loaded and applied to the SIP trunk(s) any MWI SIP NOTIFY message that leave one of those SIP trunks will be modified appropriately. At this point you should be able to test by leaving a voicemail or if you have a mailbox with an existing voicemail, but MWI failed to light, simply “Run” and MWI sync in the phone system configuration page and if the MWI lights up on the expected device, then the MWI Normalization script is working.



Synchronize MWI on the phone system integration

If the MWI Normalization is not working as expected, please proceed to the "CCM Trace Analysis" sub-section below to learn how to troubleshoot the script.

SIP Transparency and Normalization References

SIP Normalization was added as part of the CUCM 8.0 release. It uses a combination of the LUA programming language with some built-in CUCM APIs. For information on writing new scripts or modifying this script, please use the following documentation:

LUA Programming Language documentation:
http://www.lua.org/

Developer Guide for SIP Transparency and Normalization:
http://www.cisco.com/en/US/partner/docs/voice_ip_comm/cucm/sip_tn/8_5_1/sip_t_n.html

For information on how to load the SIP Normalization Script that is included in this document please follow the guide located at the following link:
http://www.cisco.com/en/US/docs/voice_ip_comm/cucm/admin/8_5_1/ccmcfg/b06scrpt.html

CCM Trace Analysis

In this section we will look at different events dealing with SIP Normalization scripts. Using this information you should be able to collect detailed traces from your system and understand what is or is not working. In the "Modifying the Script" sub-section above, the method for enabling traces within the script was described. Please follow those instructions to enable traces within the script. You will also want to enable traces for the loading of the SIP normalization script. This is done on the SIP trunk configuration page.

Enable SIP Normalization Script traces

Once all the tracing is enabled you will be able see the following events to verify operation of your SIP normalization script. Please note these traces were taken from a CUCM version 9.0 system. SDI and SDL traces in version 9.0 have been combined. The outputs we will be looking at are mostly SDI traces so if you were reading traces on an 8.6 or previous version, you should still be able to follow along.

Reset SIP Trunk (Applying SIP Normalization Script to SIP Trunk)

Successful loading of MWI normalization script:

05216617.003 |09:39:53.995 |AppInfo |//SIPLua/Info/sipNormalizationNewScript: Store new device VM-SME script SME-MWI
05216617.004 |09:39:53.995 |AppInfo |//SIPLua/Info/Load and Init successful: device VM-SME script SME-MWI module M max mem 51200 warning mem 40960
05216617.005 |09:39:53.995 |AppInfo |//SIPLua/Info/Device:VM-SME Script:SME-MWI -- base script memory 19542 user script memory 1384

Unsuccessful loading of MWI normalization script:

05218562.004 |09:55:12.744 |AppInfo |//SIPLua/Error/Load error -- device VM-SME script SME-MWI error "SME-MWI at line 53: '=' expected near 'ver'"
05218562.005 |09:55:12.745 |AppInfo |SIPNormalizationScriptError - A script error occurred Device Name:VM-SME Script Name:SME-MWI Script Function:.Initial Chunk Script Type:Custom Error Code:1 Error Code Text:Load Error Error Message:SME-MWI at line 53: '=' expected near 'ver' Configured Action:Not Applicable Resulting Action:Disable Script In Use Memory:2867 Memory Threshold:51200 In Use Lua Instructions:0 Lua Instruction Threshold:1000 App ID:Cisco CallManager Cluster ID:US-SOUTH Node ID:sjtme-pub-4a

The highlighted section tells the administrator where the loading failed so that they know where to inspect the script and resolve the issue.

Modifying MWI SIP NOTIFY Message

Successful Modification of MWI SIP NOTIFY Message:

05233353.003 |11:22:09.054 |AppInfo |SIPTcp - wait_SdlReadRsp: Incoming SIP TCP message from 10.86.140.141 on port 36298 index 1217 with 609 bytes:
[300535,NET]
NOTIFY sip:4085551011@172.27.26.60:5060 SIP/2.0
Via: SIP/2.0/TCP 10.86.140.141:5060;branch=z9hG4bK129118f593e
From: <sip:voicemail@10.86.140.141>;tag=1728527123
To: <sip:4085551011@172.27.26.60>
Call-ID: ac9bcd80-fe01c350-7e-8d8c560a@10.86.140.141
CSeq: 101 NOTIFY
Max-Forwards: 70
Date: Tue, 19 Jun 2012 18:22:08 GMT
User-Agent: Cisco-CUCM8.6
Event: message-summary
Subscription-State: active
Contact: <sip:voicemail@10.86.140.141:5060;transport=tcp>
Content-Type: application/simple-message-summary
Content-Length: 73

Messages-Waiting: yes
Voice-Message: 1/0 (0/0)
Fax-Message: 0/0 (0/0)

05233353.004 |11:22:09.054 |AppInfo |SIPTcp - wait_SdlReadRsp: SignalCounter = 130278
05233354.000 |11:22:09.054 |SdlSig |SIPNormalizeReq |wait |SIPNormalization(1,100,71,1) |SIPHandler(1,100,72,1) |1,100,13,1750.2^10.86.140.141^* |*TraceFlagOverrode

```
05233354.001 |11:22:09.054 |AppInfo  |//SIPLua/Script/trace_output: Extension Mask: 60XXXX
05233354.002 |11:22:09.054 |AppInfo  |//SIPLua/Script/trace_output: MWI Message found
05233354.003 |11:22:09.054 |AppInfo  |//SIPLua/Script/trace_output: URI: sip:4085551011@172.27.26.60:5060
05233354.004 |11:22:09.054 |AppInfo  |//SIPLua/Script/trace_output: User Host: 4085551011 172.27.26.60
05233354.005 |11:22:09.054 |AppInfo  |//SIPLua/Script/trace_output: New uri: sip:601011@172.27.26.60:5060
05233354.006 |11:22:09.054 |AppInfo  |//SIP/SIPNormalization/trace_sip_message: After inbound SIP Normalization msg is:
[300535,INT]
NOTIFY sip:601011@172.27.26.60:5060 SIP/2.0
Date: Tue, 19 Jun 2012 18:22:08 GMT
From: <sip:voicemail@10.86.140.141>;tag=1728527123
Event: message-summary
Content-Length: 73
User-Agent: Cisco-CUCM8.6
To: <sip:4085551011@172.27.26.60>
Contact: <sip:voicemail@10.86.140.141:5060;transport=tcp>
Content-Type: application/simple-message-summary
Call-ID: ac9bcd80-fe01c350-7e-8d8c560a@10.86.140.141
Subscription-State: active
Via: SIP/2.0/TCP 10.86.140.141:5060;branch=z9hG4bK129118f593e
CSeq: 101 NOTIFY
Max-Forwards: 70

Messages-Waiting: yes
Voice-Message: 1/0 (0/0)
Fax-Message: 0/0 (0/0)
```

The output in red above represents the traces that are built into the MWI Normalization script that was implemented. The outputs show the progress of the script as it reads in and modifies the information within the MWI SIP NOTIFY message. The output in green is meant to highlight the change in the NOTIFY Request URI header; the user information has been changed to match the mask that was applied in the script.

Unsuccessful Modification of MWI SIP NOTIFY Message:

```
05227895.003 |10:49:07.491 |AppInfo  |SIPTcp - wait_SdlReadRsp: Incoming SIP TCP message from 10.86.140.141 on port 36504 index 1209
with 606 bytes:
[300180,NET]
NOTIFY sip:4085551011@172.27.26.60:5060 SIP/2.0
Via: SIP/2.0/TCP 10.86.140.141:5060;branch=z9hG4bK10b1e03b9ac
From: <sip:voicemail@10.86.140.141>;tag=860301227
To: <sip:4085551011@172.27.26.60>
Call-ID: fd73100-fe01bb93-69-8d8c560a@10.86.140.141
CSeq: 101 NOTIFY
Max-Forwards: 70
Date: Tue, 19 Jun 2012 17:49:07 GMT
User-Agent: Cisco-CUCM8.6
Event: message-summary
Subscription-State: active
Contact: <sip:voicemail@10.86.140.141:5060;transport=tcp>
Content-Type: application/simple-message-summary
Content-Length: 72

Messages-Waiting: no
Voice-Message: 0/0 (0/0)
Fax-Message: 0/0 (0/0)

05227895.004 |10:49:07.491 |AppInfo  |SIPTcp - wait_SdlReadRsp: SignalCounter = 130109
05227896.000 |10:49:07.491 |SdlSig   |SIPNormalizeReq              |wait                 |SIPNormalization(1,100,71,1)
|SIPHandler(1,100,72,1)        |1,100,13,1742.8^10.86.140.141^*       |*TraceFlagOverrode
05227896.001 |10:49:07.491 |AppInfo  |//SIPLua/Script/trace_output: Extension Mask: 60XXXX
05227896.002 |10:49:07.491 |AppInfo  |//SIPLua/Script/trace_output: MWI Message found
05227896.003 |10:49:07.491 |AppInfo  |//SIPLua/Script/trace_output: URI: sip:4085551011@172.27.26.60:5060
```

05227896.004 |10:49:07.491 |AppInfo |//SIPLua/Error/Execution error -- device VM-SME script SME-MWI callid fd73100-fe01bb93-69-8d8c560a@10.86.140.141 error "SME-MWI at line 62: attempt to call method 'endcode' (a nil value)"
05227896.005 |10:49:07.491 |AppInfo |//SIPLua/Error/Traceback: in SME-MWI::M.inbound_NOTIFY at line 62
05227896.006 |10:49:07.491 |AppInfo |SIPNormalizationScriptError - A script error occurred Device Name:VM-SME Script Name:SME-MWI Script Function:M.inbound_NOTIFY Script Type:Custom Error Code:3 Error Code Text:Execution Error Error Message:SME-MWI at line 62: attempt to call method 'endcode' (a nil value) Configured Action:Rollback Only Resulting Action:Rollback Only In Use Memory:3105 Memory Threshold:51200 In Use Lua Instructions:0 Lua Instruction Threshold:1000 App ID:Cisco CallManager Cluster ID:US-SOUTH Node ID:sjtme-pub-4a

The highlighted section above shows what happens when there is a failure executing the script. Note that this script loaded ok during the SIP trunk reset, and only failed during attempted execution. Searching on “SIPNormalizationScriptError” will yield any errors related to the SIP normalization scripts whether encountered at loading or during execution. As before, the error message tells the admin where the failure occurred and where to look for the scripting error.

Non-MWI Related SIP NOTIFY Messages

With tracing enabled, any SIP NOTIFY message that is NOT carrying MWI information will print the following output in the CCM traces:

05221218.003 |10:07:42.939 |AppInfo |//SIPLua/Script/trace_output: Not an MWI Notify Message

Once the script is working as desired, the administrator can disable tracing.

Appendix

The following sections will explain how basic voicemail function work within a Cisco UC deployment. This will provide a base knowledge necessary for implementing the solution outlined in this document.

Routing Calls to Voicemail

Leaving a Voicemail

It is important to understand how calls route to voicemail. The most common way calls are directed to voicemail boxes is via a call forward event. These events commonly take place when an extension is set to either Call Forward All (all calls sent directly to VM without ever ringing the extension), Call Forward No Answer (call rings an extension but after a set time with no answer, we send the call to VM), and Call Forward Busy (an extension already has a specified maximum number of calls on it so any new calls are forwarded to VM). There are other ways to direct calls to VM, but these are the most popular.

When the forward event happens and a call is sent to VM, the original call is altered and forwarded. The calling party will stay the same, the called party will be changed to the VM pilot number, and a new field will be added which contains something called the redirecting party. In SIP this is known as a Diversion Header that you can see in the SIP message. In SCCP, this is known as the Original Called Party. For the purposes of this document, I will refer to it as the redirecting party. Here is an example:

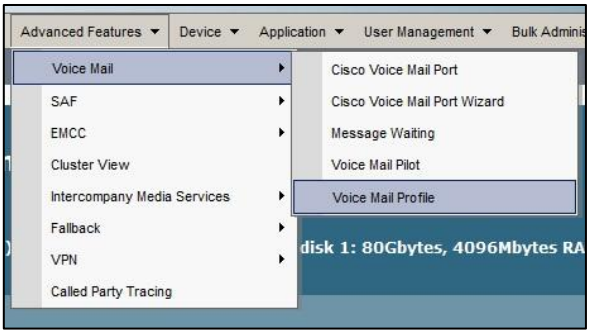
Original call to extension:
Calling Party: 1002
Called Party: 1011

After call is forwarded (for whichever reason):
Calling Party: 1002
Called Party: 2000 (VM Pilot)
Redirecting Party: 1011

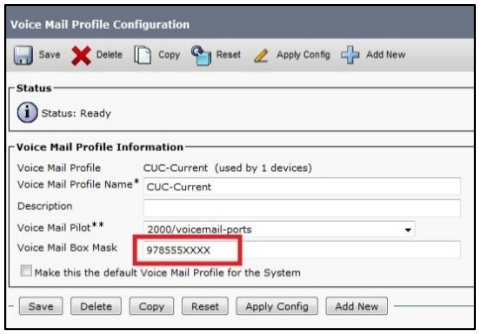
Calls can actually track two different redirecting parties, the first redirecting party, and the last redirecting party. If there were three forwards done to a call, the middle forward operation is not typically tracked. CUC can be configured to use either the first or last redirecting party. The redirecting party is used by CUC to decide which mailbox to direct the calling party towards. This is how calls get directed to the appropriate CUC account.

Within CUCM/SME we can apply what is called a “Voice Mail Box Mask” to calls that are forwarded to voicemail. This allows the users extension in CUCM to differ from the extension that is associated with the user in their CUC account. Here is an example:

User1 Configuration:
Phone Extension: 1011
Mailbox Extension 4085551011
Required Mask: 408555XXXX



Voice Mail Profile menu location



Voice Mail Box Mask

This allows us to eliminate overlapping, shorter extensions in CUC however; it presents a problem with MWI which you will read about in the following section.

Checking Voicemails

The other type of voicemail interaction most users experience on a day-to-day basis is checking their voicemail. This is not nearly as complicated as leaving a voicemail. When a user hits their voicemail button on their phone, a very simple call is placed. This will be a call from the user's extension to the voicemail pilot number. The voicemail button on the user's device derives the pilot number from the user's "Voicemail Profile" which is assigned either as a default or on a per-user basis.

So, if we leverage the information from our previous examples, pressing User1's voicemail button will yield a call with the following details:

Calling Party: 1011
Called Party: 2000

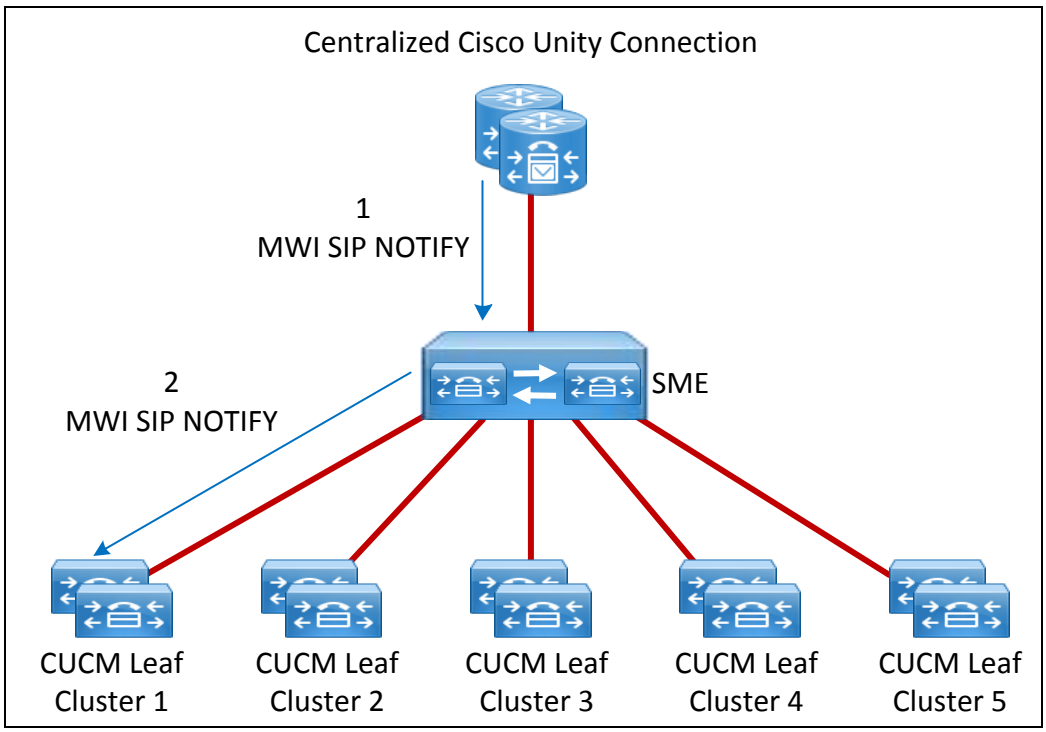
There will be no redirecting party information since this is a direct call and when CUC receives this call, it will route it to the login greeting for User1 at extension 1011. As previously discussed, if the User1's voicemail account were actually 4085551011, then we would need to apply a transformation mask of 408555XXXX to the calling party as this call left the cluster so that when the call reach CUC, the calling party would be 4085551011.

How MWI Works

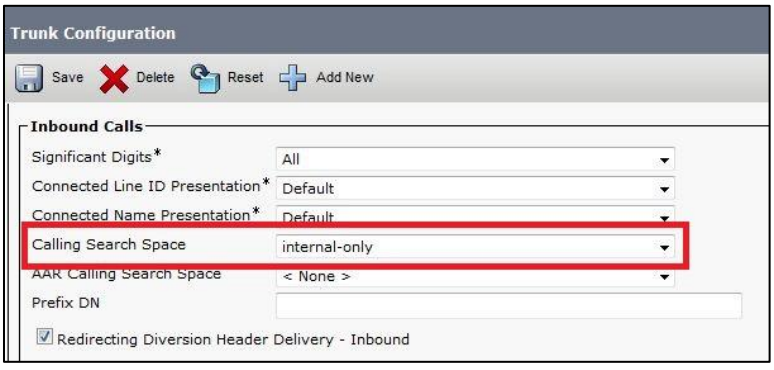
Originally, CUC (or Unity) with a SCCP integration to CUCM/SME handled MWI by placing a call into CUCM with a called party of the MWI on or off extension and a calling party of the DN of the device we wanted to alter MWI status. With these MWI extensions, we could place calls directly from end user devices to the MWI extensions to activate or deactivate the MWI light on the phone (feel free to test if you know your MWI extensions). With SIP integrations, the MWI process is different. We no longer require MWI on/off extensions and in their place, CUCM/SME use special unsolicited SIP NOTIFY messages to alter MWI status on devices.

We can still route based on the SIP NOTIFY Request URI Header (the first line of the SIP NOTIFY message), however we cannot apply translation patterns to these messages as Digit Analysis is not run

on unsolicited SIP NOTIFY messages. So in the case of SME with centralized CUC servicing multiple leaf clusters, we use this routing capability to send MWI SIP NOTIFY messages to the appropriate cluster.



One requirement for MWI functionality, in the case of SIP, is the Incoming CSS of the SIP trunk where MWI messages (SIP NOTIFY) are coming in from be capable of reaching the extension we are trying to alter MWI status. In the case of SCCP, the device's CSS must be able to reach the MWI on and off extensions. So for either type of VM integration, CSS is a critical part of making sure MWI can function correctly.



Calling Search Space of inbound calls to a SIP trunk

Another requirement for MWI to function is that the MWI number in the MWI message (NOTIFY Request URI Header for SIP or calling party for SCCP) must match the actual extension of the device whose MWI you are trying to alter. Here are a couple examples:

Successful MWI event:

Extension in MWI Message: 1011
Extension on device: 1011

Unsuccessful MWI event:
Extension in MWI Message: 9785551011
Extension on device: 1011

So if the mailbox extension doesn't match the actual DN, then MWI will be unsuccessful unless we either modify the MWI request or have CUC send a MWI request to an alternate extension (which we will cover in the next section). For SCCP integrations, we can use a translation pattern to match the MWI on and off numbers and then apply a calling transform mask to the call which can alter the calling party and allow the MWI event to properly alter the MWI status. But as mentioned earlier, for SIP integrations, we cannot use translation patterns to alter the SIP NOTIFY messages.

The MWI Normalization Script

Copy the text below line and paste it into either a .lua file or paste it directly into script window on the CCMAdmin page.

```
--[[
6/20/2012
THIS IS A PROOF OF CONCEPT SCRIPT AND SHOULD BE TESTED PRIOR TO DEPLOYMENT INTO A PRODUCTION ENVIRONMENT.
THIS SCRIPT IS NOT SUPPORTED BY TAC, CISCO SYSTEMS OR ANY OF ITS AFFILIATES.

If this script does not behave as expected,
1) check "Enable Trace" checkbox in the Normalization section in the SIP Trunk configuration page
2) reset the SIP trunk
3) place a couple of calls into the trunk to turn on/off MWI
4) collect a detailed the SDI trace file
5) mail the trace file, current SIP trunk script (if changes were made) and a description of the issue that includes
   the date, time, calling/called numbers and the result to Dan Keller and Chris Ward for review

Dan Keller
dakeller@cisco.com

Chris Ward
chrward@cisco.com

Purpose:
This script will modify the NOTIFY user number to a locally significant value to trigger MWI on/off

Parameter: extensionMask
Value: Desired extension mask to apply to MWI SIP NOTIFY messages.  Default value: XXXX
--]]

M={}

--change to trace.enable() to have trace output in SDI
trace.disable()
--trace.enable()

--Use M.outbound_NOTIFY(msg) if loading script on SME cluster and M.inbound_NOTIFY(msg) if loading on leaf cluster
function M.outbound_NOTIFY(msg)  --capture outbound NOTIFY messages

  -- Create the mask based on the extensionLength parameter
  local mask = scriptParameters.getValue("extensionMask")
```

```
  -- If "extensionMask" parameter was not specified in SIP Trunk config, we default the value to XXXX
  if mask == nil
  then
    mask = XXXX
  end

  -- Trace output: Outputs the current extension mask
  trace.format("Extension Mask: %s", mask)

  -- Get the from header to verify the message is MWI message
  if msg:getHeader("Event") == "message-summary"
  then
    -- Trace output: If we found a valid MWI message
              trace.format("MWI Message found")

    -- Since the change has to happen on NOTIFY line, we have to parse the request line
    local method, ruri, ver = msg:getRequestLine()
    -- Trace output: Displays the user@host portion of the RequestURI
              trace.format("URI: %s", ruri)

    --The requrest line is then parsed into the CUCM specific sipUtils format
    local uri = sipUtils.parseUri(ruri)

    -- Trace output: Displays the user and host after parsing
              trace.format("User Host: %s %s", uri:getUser(), uri:getHost())

    --if there is a valid URI
    if uri
    then
      --Create the new request line number.  Format will be sip:<masked number>@host
      local newuri = "sip\:" .. uri:applyNumberMask(mask) .. "@" .. uri:getHost()
                        -- Trace output: Displays the user and host after modifying with the extension mask
      trace.format("New uri: %s", newuri)

      --update the request URI
      msg:setRequestUri(newuri)
    end

  else
    --all non-MWI NTOFIY messages will be ignored
    trace.format("Not an MWI Notify Message")
  end
end
return M
```