Cisco Unified Communications MIBs, Syslogs, and Alerts/Alarms for Managed Service Providers

For Cisco Unity Release 7.0(2)

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-19789-01

Cisco Unified Communications MIBs, Syslogs, and Alerts/Alarms for Managed Service Providers

CONTENTS

Preface

This chapter describes the purpose, audience, and organization of this document and describes the conventions that convey instructions and other information. It contains the following sections:

Purpose

This document provides information for managing Cisco Unified Communications products, specifically Cisco Unity. This includes information for the products Management Information Base (MIB), and it also explains syslogs, alerts and alarms for the managed services that Service Providers implement in their networks. This document outlines basic concepts including Simple Network Management Protocol (SNMP) and the features of the management tools available for these products.

Audience

This document is for system and network administrators who install, upgrade, and maintain the Service Provider network.

You must have an understanding of Cisco Unified Communications Manager and the functionality of the related Cisco Unified Communications products included in this guide. See the "Related Documentation" section on page vi for Cisco Unified Commuications documents and other related technologies.

Organization

The following table provides an outline of the chapters in this document.

Chapter	Description
Chapter 1, "Overview"	Describes an overview of UCCX and Cisco Unity. Covers concepts with which you need to be familiar to implement SNMP, MIBs, and serviceability features.
Chapter 2, "Managing and Monitoring Cisco Unity"	Describes the management tools for managing UCCX and Cisco Unity. Provides information on product architecture
Chapter 3, "Cisco Unity MIBs"	Describes the function of all the MIBs in the UCCX and Cisco Unity products.
Chapter 4, "Cisco Unity PerfMon and Alerts"	Lists the perfmon counters and alerts for UCCX and Cisco Unity.

Related Documentation

This section lists documents that provide information on Cisco Unified Contact Center Express and Cisco Unity

- Cisco Unified Communications Contact Center Express Release 7.x—Go to http://www.cisco.com/en/US/products/sw/custcosw/ps1846/tsd_products_support_series_home.html for a complete set of user documentation for UCCX.
- Cisco Unity Release 7.x—Go to http://www.cisco.com/en/US/products/sw/voicesw/ps2237/tsd_products_support_series_home.html for a complete set of user documentation for Cisco Unity.

Conventions

This document uses the following conventions:

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in *italics*.
[ ]	Elements in square brackets are optional.
{ x | y | z }	Alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Convention	Description
`screen` font	Terminal sessions and information the system displays are in `screen` font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.
italic screen font	Arguments for which you supply values are in *italic screen* font.
→	This pointer highlights an important line of text in an example.
^	The symbol ^ represents the key labeled Control—for example, the key combination ^D in a screen display means hold down the Control key while you press the D key.
< >	Nonprinting characters, such as passwords are in angle brackets.

Notes use the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the publication.

Timesavers use the following conventions:

Timesaver Means *the described action saves time*. You can save time by performing the action described in the paragraph.

Tips use the following conventions:

Tip Means *the following are useful tips.*

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

Cisco Product Security Overview

This product contains cryptographic features and is subject to United States and local country laws governing import, export, transfer and use. Delivery of Cisco cryptographic products does not imply third-party authority to import, export, distribute or use encryption. Importers, exporters, distributors and users are responsible for compliance with U.S. and local country laws. By using this product you agree to comply with applicable laws and regulations. If you are unable to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found at—http://www.cisco.com/wwl/export/crypto/tool/stqrg.html.

If you require further assistance please contact us by sending email to export@cisco.com.

CHAPTER 1

Overview

This chapter gives a conceptual overview of Cisco Unified Communications Contact Center Express and Cisco Unity. It also includes some basic information about Simple Network Management Protocol (SNMP) including traps, Management Information Bases (MIBs), syslogs, and alerts/alarms. It contains the following sections:

Cisco Unity Architecture Overview

Note These sections contain information related to the 4.x release of Cisco Unity. While some of the information may be dated, much of it is still relevant. Work is being done with the Unity architecture team to update this information for Unity 7.x.

This section is a high level overview of the Unity 4.x architecture including several “walk throughs” that cover typical system usage scenarios and how individual services and components work within those scenarios. The reader should take away an understanding of what the various services and components are responsible for in the Unity system, how Unity interacts with the directory and message stores, and how the client interfaces into Unity work at a high level.

This document assumes the reader has a basic knowledge of the Unity administration interfaces, it is not an introduction to the Unity system. If you are unfamiliar with terms such as “call handlers”, “call routing rules”, “media master control” or do not know your way around the web based system administration (SA) interface, you will want to first review the Unity System Administration Guide and Unity User Guides.

The Cisco Unity server is a complex and powerful application that is made up of numerous individual components and services installed both on the local Unity server itself and, in some cases, on other servers in the network. Further, Cisco Unity interacts with a number of other external applications including Microsoft SQL 2000 or Microsoft MSDE, Microsoft Exchange 2000, Microsoft Exchange 5.5, Active Directory, Microsoft Internet Information Services, Lotus Domino, and also integrates with various phone switches including Cisco Call Manager and a number of legacy analog PBX systems on the market today.

On a typical Unity 4.0 installation you will see in excess of 10 separate services added by the setup process in the Windows Service Control Manager. Understanding all the components in each of these services, how they interact with each other and with all the external components Unity integrates with can, at first glance, be a daunting task. A quick look at figure 2.1 below shows there's a lot of boxes and lines on that picture, but trust me, it's not as bad as it looks. Hang with me through this document and you'll have a much better grasp of how the Unity product works.

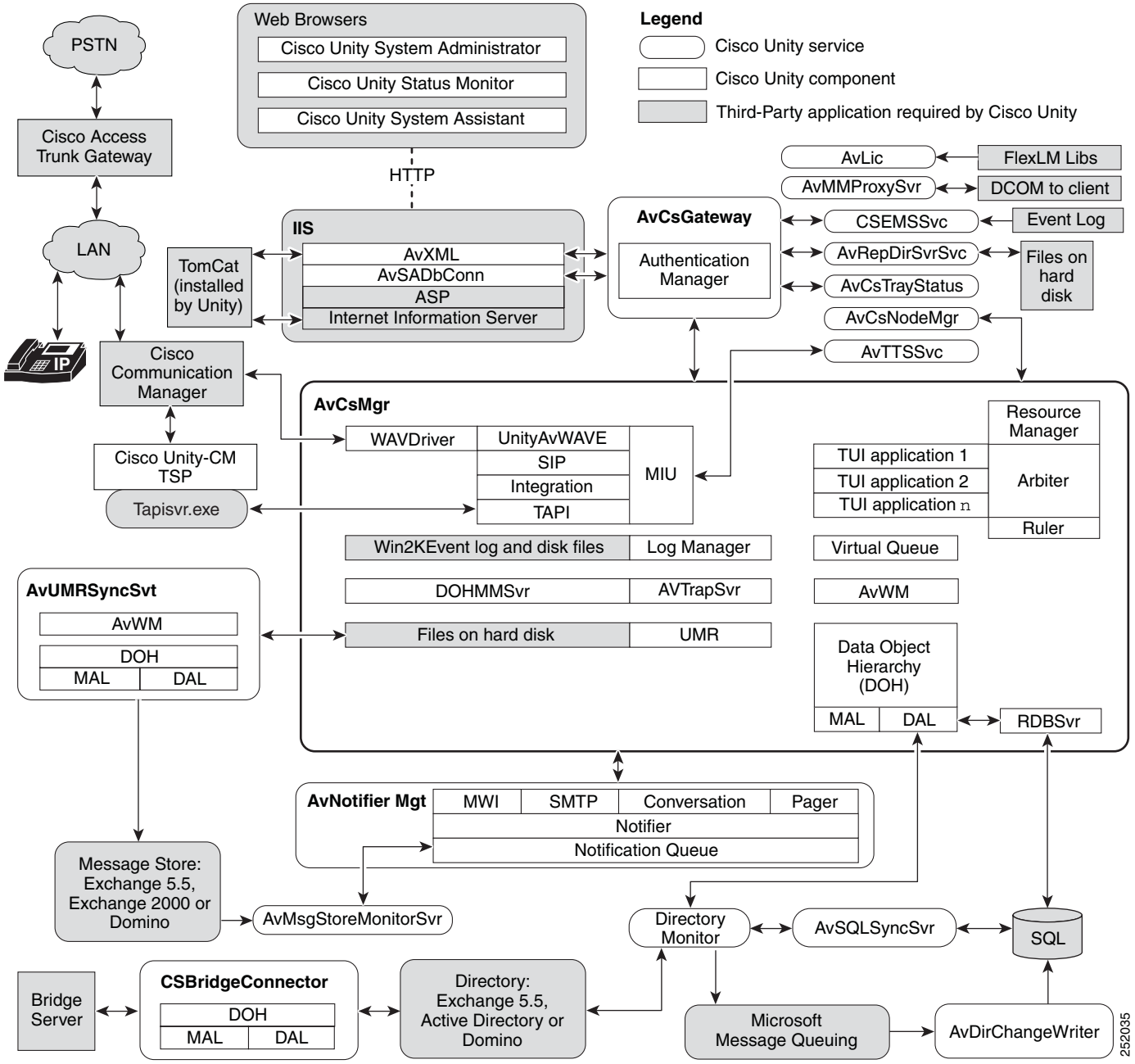
First we'll take a look at the architectural "big picture", run through all the high level components and discuss what function they serve. After that we'll do a couple of soup-to-nuts walk throughs and talk about which components are serving what role when a caller leaves a message, when the MWI is turned on for that message, when an administrator makes a change to a subscriber's record via the system administration console and when a change takes place for a subscriber in the directory.

Architectural Big Picture

Figure 1-1 below is a 10,000 foot shot of all the services and primary components installed on a Unity 4.x server. In the figure, all of the rounded rectangles represent a service or set of services. The ones with a dark background represent items the Unity install did not add to the system but are required to be there for Unity to operate properly.

One of the goals of this document is to describe which process does what and which service it runs under. If you look at the "micro traces" section of the Unity Diagnostic Tool (which can be found in the Tools Depot on the desktop) you'll find most of the high level sections offered are represented either as a service or a component running under a service in Figure 1-1. While this document is not intended to be a troubleshooting guide, some sections will show how to turn on traces and review their output to help better understand what's going on under the covers.

Figure 1-1 Unity Architecture Overview



Before we run through a couple of scenarios, let's cover all the services depicted above and broadly discuss their purpose in life. You'll notice most of the Unity related services start with "AV". This is because Cisco purchased Active Voice for the Unity product line in 2000 and such designations in the services, registry keys, message classes and other places not directly visible to end users were left as is. The diagram above also shows Unity connected to a Call Manager, remember that Unity can also hook up to legacy PBX switches, SIP gateways or combinations of any two of the three. Just the Call Manager interface is shown here for simplicity.

AvCsGateway

The gateway service does what its name implies, it acts as a gateway into the main Unity components that run under the AvCsMgr process. All outside applications that want to connect to the DOH (Data Object Hierarchy – a wrapper around the directory and messaging interfaces, see below) or any other component running under AvCsMgr has to authenticate with the gateway first. The AvSADBConn component running under IIS (Internet Information Services) which is used to produce the SA web pages has to go through the gateway to get to the DOH where it pulls all the data necessary to produce the various administration web pages. While it's possible to connect to most components running under AvCsMgr, the usual target for external applications is the DOH. As discussed in the Administering Unity Programmatically document available on www.CiscoUnityTools.com, the need to go through the proprietary DOH interfaces for administration adds, moves and deletes is no longer a requirement as it has been for earlier versions of Unity.

The gateway service also keeps an eye on the status of the AvCSMgr service and can report if it's up, down, starting up, going down, etc. to external clients that are interested in such things. The tray status application, for instance, polls the gateway periodically and uses this information to decide which icon it'll show. The tray status application also uses interfaces exposed on the gateway service for starting and stopping Unity which involves shutting down the AvCsMgr, AvRepDirSvr, AVTTSSvr, AvMsgStoreMonitorSvr, AvNotifierMgr and AvUMRSyncSvr services. The rest of the services, most notably the directory monitors and, of course, the gateway service itself, stay up and running even if Unity is off line and not taking calls.

AvCsMgr

The AvCsMgr (Active Voice CommServer Manager) is the main Unity process in the system. In older Unity versions, in fact, only the AvCsMgr and AvCsGateway services were present on a new installation, literally everything ran under the single AvCsMgr process. As Unity has scaled up and added features, various components have been taken "out of process" into their own services. Eventually as Unity moves towards a true clustering model some of these services will actually be able to run off box entirely. At some point, for instance, it'll be possible to simply add another media server to provide more TTS and/or voice port capacity to a Unity installation. While this is the direction Unity is headed, it's not there now and everything runs on a single Windows 2000 server at this point. Sites needing to scale beyond the capacity of a single Unity server for ports and/or subscriber resources will need to leverage Unity's digital networking capabilities and use separate Unity servers to balance the load. You can find more information about Unity networking features for all versions of Unity 4.0(x) here: http://www.cisco.com/univercd/cc/td/doc/product/voice/c_unity/unity40/net/index.htm

There are a number of important processes that run under the AvCsMgr service. The following is a list of the more important components.

DOH

The Data Object Hierarchy is the abstraction layer over the directory and messaging back ends supported by Unity. The idea here is simple: provide a common interface for all client applications such as the phone conversations, administration interface, reports and the like for getting at directory and messaging information from Exchange 5.5, Exchange 2000, Exchange 2003, Active Directory and Domino.

The DOH has two main sub components: The Directory Access Layer (DAL) and the Message Access Layer (MAL). The original idea was to provide different MAL and DAL "plug ins" for each back end Unity supported. Clients would be blissfully unaware of the wildly different access protocols going on underneath the DOH. The first incarnation of the DOH was an abstraction layer over the Exchange 5.5 directory and mailbox data for Unity 2.x when all Unity directory information and messages were stored right in Exchange. The LDAP and MAPI interfaces used to get at this data in Exchange 5.5 is more of an art than a science and having a fully functional wrapper around this complexity was an absolute must.

When Unity 3.0 hit the streets and introduced SQL as the primary directory repository, the necessity of the DOH wrapper around database access was not nearly as critical. Unity now employs directory monitors to pull in data from the directory to a local SQL database and to write necessary changes back to the directory. This would enable clients to simply use standard SQL access mechanisms to retrieve and update data instead of using the proprietary DOH interfaces. However time to market considerations deemed it necessary to leave the client interfaces into the DOH alone and simply re plumb the DOH back ends to talk to both the directory monitors and the SQL database.

While the DOH is still in use in Unity 4.x, its days as the primary back end wrapper for the directory are numbered. As discussed in the Administering Unity Programmatically document noted above it is possible to interface directly with SQL for all your basic Unity administration needs and never talk to the DOH at all. The long term model is to have all clients use interfaces right into the Unity SQL database (via XML/SOAP/ADO/ODBC/JDBC/Other interfaces) and rely on the directory monitors to do the heavy lifting for the various back ends supported. See the Directory Monitor section below for more on how that works.

While the DAL component under the DOH is being phased out, the MAL wrapper around the messaging tasks will likely live on for a while in one form or another. In the case of Exchange the MAL uses MAPI client access methods very similar to what an Outlook client does to gain access to a mailbox. During the configuration setup a special mailbox named "Unity_(server name)" is created on the Exchange server selected as the "companion server" for Unity (which could be on the same box with Unity). This mailbox is referred to as the "Unity System Mailbox" in the documentation. A MAPI profile is created on the local Unity server that "points" to that mailbox. This message profile is very similar to a profile you'd create for your Outlook client that points back to the Exchange home server your mailbox is stored on. The Unity system mailbox profile, however, is optimized in a couple of ways such that it can support many hundreds of simultaneous MAPI connections which is not possible with a standard profile. In short this profile is Unity's ticket to the MAPI ball and we use it to gain access to all mailboxes for subscribers serviced on the local Unity server. This profile and the Unity system mailbox account are critical to the proper functioning of the system and, as such, they are created on the fly the first time the DOH tries to log into a mailbox if they are missing in the system. In the past this was checked at startup and the profile and account would be created then but in 4.0(3) this check was moved to the first time a mailbox was accessed. This turned out to be necessary since so many folks gave into the natural human instinct to destroy that which they don't understand and would delete one or both of these items when they encountered them without realizing what they were doing. If something gets out of whack with the account and/or the profile, it's many times as easy as deleting them and restarting the server.

In the case of Domino, the DOH uses the Notes client interface (which must be installed on each Unity server) to get at messages for subscribers and send messages of its own. The Domino Server will see Unity as a client, however the user ID we use to log in has rights to the server's address book and the administrative request database. This allows the Unity system to log into subscribers mailbox and get at the directory (address book) information.

Resource Manager

The resource manager is used to help avoid contention for limited resources on the server. For instance the notification MWI device manager (see the notifier section below) will "ask" the resource manager is there's an available port for doing an MWI dialout. If there is, the resource manager will reserve that port for the device manager which can then ask the arbiter to initiate the dialout. Similarly if a conversation would like to use the Text to Speech (TTS) engine to play the body of an email to a subscriber, it will reserve a TTS session from the resource manager. If there's a session available, things proceed as normal, if not the conversation tells the subscriber to try again later. In this way no more than the licensed number of TTS sessions can ever be active at the same time.

Arbiter

The arbiter is the call "traffic cop" of the Unity system. All processes needing access to a voice port for whatever reason has to first go through the arbiter to get one. It's the arbiter process that reads the port capability settings on the "Ports" page in the SA and enforces them. It's also the arbiter that uses the Ruler (see next section) to handle processing the routing rules configured on the "Call Routing" page in the SA that are used for deciding where new inbound calls go in the system. The processing and debugging of these rules is covered in more detail in the Audiotext Applications In Unity 4.x document which can be found on www.CiscoUnityTools.com.

Ruler

The ruler component is used by the arbiter for evaluating the routing rules when deciding where a new inbound call goes in the system. The ruler pulls the call routing rules out of an SQL table in 3.1(3) and later, however in earlier versions of Unity these rules were stored in the Routing.RUL flat file in the \Commserver\Support directory. The arbiter passes the call information in (calling number, forwarding number, dialed number, if it's a direct or forwarded call) and the ruler evaluates all the rules in order until it finds one that matches which it then passes back to the arbiter to take action on.

UMR

The Unity Messaging Repository interface is part of the mechanism that helps insulate Unity from volatility in the external network. Since Unity uses the Exchange or Domino message stores directly instead of using a proprietary message store and synchronizing, the availability of those mailstores affects users' access to messages over the phone interface. The UMR is used to provide callers the ability to continue leaving messages and retrieving some voice mail messages while access to remote Exchange servers is interrupted.

Outside caller messages are delivered via the UMR process. When the message is recorded by one of the phone conversations (or "TUI Applications" in the figure above – see below) the UMR process under AvCsMgr deposits the message as a pair of files (the message itself and a routing file indicating where it needs to go) in a directory on the local hard drive. The AvUMRSyncSvr service (see below) picks

these messages up and delivers them to the mailstore back end via the MAL interface of the DOH. If for whatever reason the ability to hand the message off to the Exchange or Domino back ends is interrupted, the messages remain on the local hard drive until such time as they can be delivered. While messages wait on the hard drive for delivery, the subscriber conversation can access them and let users get at messages left while connectivity to the mailstore is interrupted using a limited "UMR conversation".

TUI Applications

Telephone User Interface Applications (or simply "conversations") make up the user experience when interacting with Unity over the telephone. Not every conversation, however, is made up of traditional recorded prompt, voice names and greetings played to a caller. There are dozens of separate conversations used for doing everything from finding subscribers in the directory to recording messages to logging subscribers into their mailboxes.

The most visible conversations, and the ones you should care about, are the top level "routable" conversations that show up as options in the SA and other administration applications such as the Audio Text manager. For instance when you select an after greeting action that sends the caller to the operator call handler and tell it to "send to greeting" you are actually telling Unity to invoke the PHGreeting conversation with the operator call handler as a parameter. As a call moves through Unity it carries with it a handle back to the MIU (Media Interface Unit, see below) and conversations get user inputs (i.e. touch tones pressed or the call hanging up) and can request the MIU play out greetings, voice names and prompts that make up the Unity phone conversation. Each conversation that handles a call uses this same handle to check for events and request actions via the MIU. Conversations also have access to the call information via this handle which includes the calling number, forwarding number,that the MIU got from the phone system when the call first arrived.

The following is a list of all the routable conversations visible externally in Unity:

- PHGreeting. The Phone Handler Greeting conversation needs a call handler as an input. This conversation processes the greeting rules on a call handler and based on which rules are active, which schedule the handler is associated with, the origin of the call and the reason the call came to Unity it will select a greeting rule and execute it. Every subscriber is associated with a special call handler called a "primary call handler" and, as such, this same conversation is used to send callers to a subscriber or an audio text application call handler. See the "Unity Data Object Model" document available on the Documents page of www.CiscoUnityTools.com for more on call handlers and subscriber structures in Unity.
- PHTransfer. The Phone Handler Transfer conversation also expects a call handler, however this conversation processes the transfer, or contact rules on the handler. Depending on which transfer rules are active and the schedule the call handler is associated with this conversation will execute a transfer rule. If the transfer rule is set to not transfer at all but to go straight to the greeting for the call handler, the PHTransfer conversation will turn around and invoke the PHGreeting conversation for the same call handler. One of the more common things folks stumble on is the "entry point" into the call handler when they expect it to try and ring the phone and, instead, it goes to the greeting. This is covered in more detail in the "Audiotext Applications in Unity" document noted above.
- SubSignIn. The Subscriber Sign In conversation collects the ID and the password (if configured) of a subscriber wanting to log into their mailbox and check their messages. This conversation is similar, but not the same as the "AttemptSignIn" conversation below. Once the user is authenticated the call is then passed to the subscriber inbox conversation which allows users to access their inbox and other options configurable over the phone.

- PHInterview. The Phone Handler Interview conversation expects an interview handler to be passed to it. It will then play all the questions recorded for an interview handler and record the caller's responses.
- AD. The Alpha Directory conversation expects a reference to a Directory Handler to be passed to it. This conversation has been around since the first version of Unity however in versions prior to 4.x there has only ever been a single, hard coded directory handler. In 4.0(1) administrators can create any number of directory handlers with different sets of users associated with each.
- AttemptSignIn. The Attempt Sign In conversation is referenced by the default routing rules for processing new incoming direct calls. This conversation will search for the calling number in the directory and if a match is found for a subscriber it will prompt the caller to enter their password if one is configured. If no match is found the conversation kicks the call back out to the arbiter which will continue down the list of routing rules looking for a match. Routing rules and how they work are covered in the "Audiotext Applications in Unity" document noted above.
- AttemptForwardToGreeting. The Attempt Forward To Greeting is also used by the default routing rules for handling new incoming calls but this rule is used for calls that forward into Unity. The forwarding number is searched in the directory for a subscriber or call handler that matches. If a match is found the call is handed to the PHGreeting conversation with the matching call handler as a parameter. Note that the PHTransfer rule is specifically not used here to prevent a "looping call" scenario where Unity transfers to a number which forwards to Unity and Unity then transfers to that number again etc. There are times when you want Unity to transfer to another number when a call forwards in, however, and techniques for dealing with that problem are covered in the "Audiotext Applications in Unity" document noted above.
- GreetingsAdministrator. This is a new routable conversation that was added in Unity 4.0. It allows administrators to setup a one key map such that they can enter a special conversation that allows them to record the greetings for call handlers over the phone interface. They have to log in as a subscriber in this conversation and it checks to make sure they are the owner (or are a member of the distribution list that's the owner) of the call handler before letting them get at the greetings for a handler.

Log Manager

The log manager is responsible for all diagnostic and data (for reports) logging to flat files on the local Unity server as well as handling information bound for the event logs. It's important to note that each Unity service fires up its own instance of the log manager, not just the AvCsMgr service. If there were only one instance of the log manager running under AvCsMgr all other services would have to authenticate against the gateway to gain access to it and use it which would be really inefficient. As such each service has its own log manager which I didn't add to the 2.1 diagram to try and keep the clutter to a minimum.

At first blush this might seem like just a nerdy detail, however it has one very important ramification when you're troubleshooting a Unity server or perhaps trying to pull information to generate a report or the like. Each service logs into its own files under \Commserver\Logs. If you go look in this directory you'll see a series of files in there named "data_AvCsMgr_YYYYMMDD_HHMMSS.txt", "data_AvCsDirChangeWriter_YYYYMMDD_HHMMSS.txt", "data_AvTTSSvr_YYYYMMDD_HHMMSS.txt", etc. Each service represented in figure 2.1 will have a data file associated with it in the logs directory. If you have traces turned on for any components you will also see "diag_*.txt" files in the same name format which includes the service name, date in [year][month][day] format and time in [hour][minute][second] format to ensure uniqueness.

This puts the onus on the technician to know, for instance, that when they turn on an MIU trace that information will show up in the "diag_AvCsMgr*.txt" file since that's the service it runs under. On the other hand the TTS traces, which are closely related to the MIU, actually show up in the "diag_AvTTSSvr*.txt" file. This can, to put it mildly, be a bit frustrating to someone new to Unity trying to figure out what they're looking at. The Unity Diagnostic Tool found under the Tools Depot on the desktop is designed to help pull the information you need out of the various log files for you which helps somewhat but nothing beats understanding the process.

One last note to make about the log manager before moving on. The keen observer will note that some components show up under more than one service such as the DOH running under AVCsMgr, CSBridgeConnector and AvUMRSyncScr. So what happens, then, when you go turn on DOH traces in the Unity Diagnostic Tool? Output from these DOH traces will show up in each of the diagnostic logs for each separate service. A little tricky but it makes sense if you consider that each service also has its own instance of the log manager so every process running under it will be logging to the file opened by that service using the log manager.

Tip The raw Diag*.txt file stamps its entries with a time from the local Unity server as you'd expect, however the Data*.txt file stamps its entries with the "system time" which is one hour earlier than GMT. There are Windows API calls to convert these times to the local time which is necessary when generating reports off these files. It was done this way to facilitate, at some point, being able to "merge" report data from multiple Unity servers into single reports.

Note When viewing diagnostic files, it's best to use the Unity Diagnostic Tool to gather the logs instead of viewing them "raw" off the hard drive. The UDT tool formats the logs and makes them more readable by adding insertion strings for hex codes that produce more human understandable output. We'll walk through this process later when we run through a few flows and look at some diagnostic output.

Tip When Unity runs reports it's pulling information out of the ReportDB database (see the "Unity Data Object Model" document noted above for more on this). The curious observer may ask how information gets from the DATA log files generated by the log manager into the ReportDB database. There's a process called the "scavenger" that kicks of every 30 minutes and pulls relevant report data from the DATA files and stuffs them into the ReportDB tables where the reports modules can get at them. This is one of the reasons why when you make some test calls and then immediately run a report your data doesn't show up. I've taken a number of questions along these lines in the past. There's no way to force the scavenger to run immediately, you just have to hang tight until it does it's job.

MIU

The MIU (Media Interface Unit) is responsible for interacting with the telephone switch(es) Unity is connected to. All audio, integration information, digit presses, dialouts, etc. go through this component. The MIU is a sophisticated component that can handle communicating with numerous phone systems using many different integration methods. It can also handle talking to two different phone systems at the same time on a single Unity server which is a capability unique in the industry.

TAPI (Telephone Application Programming Interface)

Prior to Unity 4.0(1) all the switch integrations Unity supported went through the TAPI interface in one way or another. Moving ahead Unity will be shifting to alternative interface mechanisms such as SIP that allow more robust and scalable solutions for switch interaction.

Integrations with traditional PBXs use Dialogic voice cards which come with a TAPI driver that sends information to and from the MIU via the Microsoft TAPI driver. Unfortunately serial data and inband DTMF information for call integrations with many PBXs is not supported by the TAPI protocol so the MIU has to do an "end run" around TAPI to get this information when necessary. This is the "integration" block under the MIU on the 2.1 diagram. Unity did, at one point, also support Natural Microsystems and, briefly, Brootrout voice cards which provided their own TSPs, however that equipment is no longer supported.

Even though Call Manager does support a native TAPI interface, Unity does not use it. Instead, we wrote a TSP (TAPI Service Provider) that talks directly to the "Skinny" interface into Call Manager which, in turn, communicates through Microsoft's TAPI driver to and from the MIU. This was done since the Skinny interface is considerably lighter and faster than the direct TAPI interface into Call Manager since it only supports a subset of the full TAPI functionality that Unity needs.

Microsoft pushed TAPI as the greatest thing since sliced bread and pitched that it would unify the telecommunications industry into one big happy family. Short story, TAPI is dying and support for it is quickly drying up as new more flexible, faster and interesting interfaces such as SIP hit the streets. TAPI support in Unity will continue for a while, however work with Dialogic on a SIP based interface to their equipment is already well under way. For more information on the TAPI interfaces and the Microsoft telephony model it's a part of, visit the MSDN "Telephony Overview" site here:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tapi/intro_2khj.asp

Integration

For legacy PBXs that require serial or analog integration information, the MIU has to go around TAPI since the specification does not account for much in the way of custom integration information. Unity has flexible switch definition and integration instruction files that allow fairly easy and robust integrations with numerous switch types and models. Cisco TAC only supports a limited subset of the larger PBXs on the market but the thing to remember is that Unity was built on top of an integration engine that was designed to attach to just about anything that could spit out dial tone. If you have some of that Lewis and Clark spirit in you it's possible to configure switch integration files to talk to just about anything, however this is not supported by TAC as you might suspect.

SIP (Session Initiation Protocol)

As of version 4.0(1) Unity supports SIP for communicating with phone systems. Session Initiation Protocol (SIP) is a multimedia control protocol that was designed to initiate, modify and tear-down IP sessions. Standards based, it is defined by the IETF in RFC3261, ratified in 1999 which I'm sure you are all rushing out to go read right now. A little SIP hype here:

SIP is a peer-to-peer protocol that utilizes the architecture of the WWW. This means that end-devices, called SIP User-Agents (UA), can have various levels of intelligence as well as utilize the services available within the network. SIP UAs can take various formats, including SIP Phones, SIP-based PC clients, SIP VoIP Gateways or any other device that might be contacted for a SIP session. SIP services might include device registration, call routing, call redirection, feature invocation, device notification, or various other aspects that effect a session. These services might be provided by a SIP Proxy server, SIP Redirect server, SIP Registration server, or other SIP UAs.

In addition to providing a WWW-like architecture, SIP only specifies how sessions are initiated, modified and torn-down. For additional functional, SIP utilizes existing IETF protocols such as RTP/RTCP for media, SDP for session description, MIME for message/application encoding, HTTP for message formatting and URLs for addressing. Because of this abstraction between signaling, media and application, SIP is application independent and very extensible for applications that utilize things like HTML, Email, or Instant Messaging and Presence.

The Unity Sip component talks Sip natively via a Sip Stack that was developed within the BU, based on the original Sip RFC254bis-04. It does *not* go through TAPI for call control. The Unity Sip component requires a Sip proxy for registration and call routing/forward calls into voice mail. Currently, we are only using Sip for traditional voice mail services but are pursuing other interesting applications such as presence and video messaging and a lot of other things that make sales folks' toes curl.

Both the Skinny TSP and the Unity Sip component use AvAudio.sys (Unity wave driver) to communicate RTP for the voice media stream.

UnityAvWAV

This component sits over the WAV driver. Currently Unity supports wave drivers for the Dialogic voice boards and the VoIP protocols (Skinny and SIP). UnityAvWav provides extended services not handled by standard WAV drivers. The AGC (Automatic Gain Control) is handled here for systems not using Dialogic cards (the Dialogic drivers has it's own AGC capabilities built in). Other features such as speed control and codec conversion (i.e. switching from 711 to 729a or back) are also done here.

Virtual Queue

The Virtual Queue is used for call holding functionality within Unity. All calls currently on hold waiting for an extension to free up are represented in the queue.

RDBSvr

The RDBSvr (Remote Database Server) component is a thin layer over the SQL data access. In short this is just a set of libraries that make finding, adding and updating information into the SQL tables easier. This served as a shim between the DOH back end and the SQL databases and as the DAL component of the DOH is phased out this, too, will retire.

TRAP Connection Server

The TRAP (Telephone Record and Playback) connection server is used to satisfy connection requests from clients wanting to use their phone to playback and record messages, greetings and voice names. The Media Master control will try and make a direct connection to the both the AvTRAPConnectionServer and the DOHMMSvr component (below) in order to playback the requested WAV file at the client desktop.

The client (SA, AA or VMO form) uses the media master ActiveX control to connect directly to the TRAP connection server to establish a phone connection to the client's desktop. The TRAP Connection server will then take care of talking to the resource manager and the arbiter components to see if a port is available to do a media master dialout. If there are no ports available to handle this request, the client is sent an error message and the user is told at the desktop that there are not ports available and to try again later. If there is a port available, the MIU is instructed to dial out to the extension number specified

in the media master configuration at the desktop and once the user picks up the phone, a media connection is established and all playback and recording of WAV files takes place over that phone connection.

One thing to keep in mind, however, is the media master ActiveX control on the clients uses DCOM (Distributed Component Object Model) to connect directly back to the Unity server from the desktop. Since DCOM does not work properly through fire walls or proxy servers, it's difficult at best to establish media master connection (either using TRAP or just your speakers and soundcard on the local client) from outside the local intranet. Further, when using NTLM (NT LAN Management) authentication for the SA, the ActiveX control will need to authenticate a 2nd time when the client and the Unity server are in separate domains since it's not able to "piggy back" on the NTLM challenge and response authentication the user had to supply when first accessing the page, it needs its own security token. Unity 4.0 offers an alternative to NTLM to help with sites that are not using NT as their primary network authentication mechanism or want to work around the firewall/proxy server/domain boundary issues.

DOHMMSvr

The DOH Media Master Server is used with the media master on the web client pages and the VMO form in Outlook to stream voice files. If the client is using TRAP or just their local speakers and soundcards to playback/record greetings, voice names, messages via VMO or the SA, the client needs to make a connection back to the DOH media master server to establish a stream for the WAV file in question. The client passes in the identifier of the WAV file they wish to playback or record to the media master server and the server establishes a stream handle and passes it back to the client which can then play or record the WAV file.

In the case the client is wanting to use TRAP to playback a greeting, for instance, the media master control on the SA connects to both the AVTrapSVR and the DOHMMSvr components directly via DCOM. Once a port has established a connection to the local phone the media master server is then asked to establish a stream connection to the WAV file desired over the phone connection. At this point the message is then streamed over the network to the client box and played out there.

AvWM

The Windows Monitor is responsible for determining the up or down status of all Exchange servers the local Unity system cares about. In short any Exchange server that is housing one or more subscribers on the local Unity server is added to the list of systems that the windows manager "pings" every 15 seconds (in versions 3.1(3) and earlier this was every 30 seconds). If the server is off line or the Exchange directory (in the case of Exchange 5.5) or mailstore services (for both Exchange 5.5 and 2000) are not responsive then this server is marked internally as being down. When a subscriber signs into the Unity server over the phone to gain access to their messages, the conversation first checks to see if that Exchange server is up or down before attempting to log into that user's mailbox. If it's down, they are not allowed to attempt the log in since this can result is very long MAPI timeouts and will result in a call being stuck or "wedged" for a long period of time. If the Exchange server that you selected during the configuration setup (the "companion Exchange server" as it's called in the documentation) is off line, then Unity goes into full "UMR Mode" and will store voice messages locally until connectivity can be reestablished.

In the case of Domino, the monitoring service here can't assume the server is running on top of a Windows platform so things get a little trickier. It simply pings the server by name and checks for a response to make sure it's up. It does not, however, know if the Domino services on that box are functioning properly.

Whenever a server is marked as being down or back up again, an event log message written through to the application event log. The Exchange Monitor Tool introduced for Unity 4.0 can be found in the Tools Depot and is designed to give the administrator a look at the current up/down status of all Exchange servers Unity cares about and can also be used to reconstruct the up/down status history by pulling these events from the log and putting them together into a time line. For sites having trouble with their external Exchange servers or that are experiencing severe network latency issues, this can be helpful in reconstructing Unity's view of the network over a period of time.

AvNotificationMgr

For the Unity 4.0(3) release, the notification processes that are responsible for lightning MWIs, sending out notification emails, pages etc… based on inbox activity have been pulled out of the AvCsMgr group and into their own process here.

Notifier

What's referred to generally as "the notifier" is actually a set of three separate components. The AvMsgStoreMonitorSvr (see below) watches all inboxes of subscribers on the local Unity server for changes and then pushes that information onto the notification queue. The notifier engine in the AvNotificationMgr process watches the queue for items it needs to react to and it, in turn, hands the request off to one of 4 "device managers" for MWIs, SMTP (text pagers), Conversations (phone notification device), or Pagers (tone pagers). Each of these device manager is responsible for sending the emails, lighting the message lamps or making the phone calls to satisfy the delivery requirements for the subscriber in question. The notifier is also responsible for kicking off the AMIS message delivery process which is covered in the Digital Networking documentation noted above.

The MWI, Conversations and Pager device managers all work with the resource manager and the arbiter components in the AvCsMgr process to initiate the dialout/MWI requests. The SMTP device manager talks directly to the DOH component to send out the emails for text pagers and the VMI notification devices. When they have failed or succeeded to carry out their notification duties they will push a message to this effect onto the notification queue to let the notifier know the request is complete one way or the other. Each notification device maintains a local queue of tasks and is responsible for handling the retries configured for that device. For instance deliveries to the home phone may be set to retry up to 4 times if the line is busy or MWIs may be configured to retry several times in the case of a serial integration. It will only mark a notification attempt as failed after exhausting all the retries configured for that device.

Later in this document when we do a call flow walk through we'll discuss a little more how this process works and take a look at some trace output from the various components to show what's going on under the covers.

Notification Queue

The Notification queue is, as the name implies, a simple queue of events that the notification engine watches. The AvMsgStoreMonitorSvr service (see below) is responsible for actually "watching" subscriber mailboxes for events that require the services of the notification engine and pushing those events onto the queue. The individual notification device managers mentioned above also push events onto this queue to let the notifier know if it was successful or not in satisfying the notification request.

AVMsgStoreMonitorSvr

The message monitor server service is responsible for watching all the mailboxes of all the subscribers on the local Unity server. Whenever a change of any kind happens in an inbox it's monitoring it will push the event and the mailbox identifier on the Notification Queue in the AvNotificationMgr process (see above). The notifier, in turn, pulls that information off the queue and will then go out and filter the inbox to get a message count and decides if a notification action such as an MWI on/off or an email page or the like needs to happen.

In the case of Exchange 5.5 or Exchange 2000, the message store monitor logs into the mailbox for all subscribers homed on the local Unity server. In the case where this mailbox is off the Unity box (i.e. the Exchange server is not installed on the same box) there's a security warning (more specifically a "success audit") logged in the application event log since Unity is logging in using the account associated with the AvMsgStoreMonitorSvr. Exchange will log this anytime another account logs into a mailbox that's not its own, even if it has rights to do so, this is not a message to be concerned about:

Windows 2000 User LINDBORGLABS\Administrator logged on to EAdmin@lindborglabs.thinkpad.com mailbox, and is not the primary Windows 2000 account on this mailbox.

Note that at times when Exchange is busy or too many changes take place on an inbox in too short a time (i.e. the user deletes many messages at once) Exchange can return an event that indicates we simply need to resynch the entire mailbox. In this case the notifier must go out and filter the entire inbox to know what to do. In most cases, however, it's not necessary to filter the entire inbox, it's possible for the notifier to determine what needs to be done specifically from the change notice the message store monitor pushes onto the queue. This makes it a little difficult to look at the diagnostics kicked out by the notifier and message store monitor components since there's no hard and fast rule for when Exchange decides it's "too busy" to issue a specific change request or a general "resync" notice for an inbox.

In the case of Domino, the message store monitor actually registers for changes on any inboxes it's interested in via the DUCS (Domino Unified Communications Service) interface on each Domino server that houses one or more Unity subscribers. When any message is added, removed or modified the message store monitor is notified of this change and pushes the appropriate event onto the notification queue in the same way the Exchange message store monitor does. At startup Unity logs into each user's mailbox and synchs, after that it's simply a matter of adjusting as necessary based on change events for each user's mailbox.

It's important to note that the message store monitor itself does not determine if something needs to happen, just that a change has occurred on a mailbox it's watching. It knows what type of action happened such as a new message was added, a message was modified (i.e. marked read) or deleted, however it doesn't know what needs to be done for the subscriber when than action takes place. The notifier running under AvNotifierMgr must do the work to determine if anything needs to be done such as lighting an MWI light or initiating a pager dialout or the like.

Later in this document when we do a call walk-through of an outside caller leaving a message for a subscriber we'll cover the notification process in a bit more detail and look at some of the trace output from these components.

Directory Monitor

The Directory monitor actually goes under different names depending on which back end Unity is configured for, however the functionality is the same. The directory monitor is responsible for all adds/moves/changes for all objects Unity writes to in the directory. This includes mail users (subscribers), public distribution lists, contacts (custom recipients in Exchange 5.5) and location objects. The monitor is also responsible for finding users in the directory that are not yet subscribers. The SA

import and the stand alone import tools in Unity use this interface provided by the monitor called the Import Directory Connector (IDC) for allowing administrators to make standing mail users in the directory a subscriber. The directory monitor is also responsible for finding all subscribers and location objects added to the directory by other Unity servers on the network. These objects are added to the global subscriber and global locations tables in the local SQL database and are used for digital networking functionality.

Changes or deletions of subscribers and distribution lists in the directory are noted by the directory monitor and their local representations in SQL are updated by the directory change writer (see below). Unity knows an object in the directory has been changed by checking the Change ID value against its local copy in SQL. If the number is higher in the directory, a change has occurred on that record external to Unity and the directory monitor pulls in information about that object into SQL again. When a change or an add comes along in the directory, the monitor pushes that information into the Microsoft Messaging Queue (MSMQ) which is watched by the DirChangeWriter service. The DirChangeWriter then writes the change through to the local SQL tables as necessary. Later in this document when we walk through what happens when an administrator makes a change in the SA we'll look at some of the directory monitor, directory change writer and MSMQ traces for a system using Active Directory (Exchange 2000).

It's important to note that the directory monitor is not responsible for knowing when an item in SQL is updated and then writing that through to the directory. Currently clients that use the DOH don't have to worry about this since the DOH writes changes through to the directory via the monitor and then updates SQL once the directory update is completed successfully. For folks that wish to be able to update SQL directly and have those changes propagate through to the directory without dealing with the DOH or updating the directory manually you need to use the SQLSynchSvr service discussed below.

The directory monitor is actually one or two services named differently depending on which directory back end Unity is connected to.

Exchange 5.5

The directory monitor service is called "AVDSEx55". By default the AvDSEx55 service is watching all containers from the site level down in the Exchange 5.5 site it's installed in, however this scope can be changed to the organization level if necessary by editing the registry or going through the DOHPropTest application. This is covered in the Digital Networking documentation noted above.

Exchange 2000

For Exchange 2000 there are actually two services with make up the directory monitor functionality. The "AvDSAD" and "AVDSGlobalCatalog".

The AvDSAD service watches for updates to subscribers that are homed on the local Unity server. It will only monitor domains and containers that it absolutely has to since there's no way to have AD trigger on changes to the service has to poll the directory for updates. By default this is done every 2 minutes but it's adjustable in the registry if necessary. If a subscriber is added that's in a new domain the service is not already watching, that domain is added to the list it is keeping an eye on. You can see a list of the domains it's looking at in the registry under HKLM\Software\Active Voice\Directory Connectors\DirSyncAD\1.00\Domains\. Each domain will have a top level container for subscriber and distribution lists that the directory monitor is watching. This container and all its subcontainers are being polled for changes to mail user or distribution list objects. It's possible for this root container to actually be the domain root itself. In the registry the container is actually referenced by a directory ID

which is not going to tell you much, however you can also see this information in DOHPropTest under the "AD Monitor" button. It will list all the domains and for each domain a human readable container name for the root containers for mail users and distribution lists in that domain.

The AvDSGlobalCatalog service, on the other hand, watches for subscribers, distribution lists and location objects added to the directory by other Unity servers in the forest. It pulls this information in and writes it to MSMQ where the Directory Change Writer updates the global subscriber, global locations and the distribution list tables in the local SQL tables. Since it can't know which containers or domains those Unity servers may be installed in, it has to watch the entire forest. Again, it must poll the directory for adds/moves/deletes, but since this information is much less critical than updates to local subscribers this polling interval defaults to 15 minutes. The AvDSGlobalCatalog service is always watching all containers in all domains in the entire forest, there is no way to limit its scope to a smaller subset. This service also provides the IDC interface mentioned above which is used by the import utilities to find mail users in AD that can be imported as a subscriber.

Domino

In the case of Domino, the directory service name is AvDSDomino and it works in much the same way as the other directory monitors for Exchange do. In Domino land, however, the "directory" is actually the address books that contain user information. Each Domino server has an address book and a Unity server can be configured to watch one or more address books for changes and allow imports of users in those address books. If a Unity server is handling more than one Domino server it's necessary for the administrator to include each and every address book containing users they want to service with the Unity box in the SA.

Every minute or so the monitor will checks for changes made to the address books Unity is configured to watch since the last time it looked. Updates to users we care about are then pulled into SQL using the same mechanism. Again, any changes to a user in the address book results in a "bulk write" of all the properties in the address book Unity cares about into the local SQL subscribers table.

AVDirChangeWriter

The AVDirChangeWriter service is the same regardless of which back end Unity is connected to. All the directory monitor services write the adds/moves/deletes that need to be updated in the local SQL database to the MSMQ which the directory change writer watches. When updates are pushed onto the MSMQ, the directory change writer pulls them off and executes the update in SQL.

Note that blocks of properties for subscribers are written through to SQL even if you only change a single property on the user or distribution list in the directory. As such if you change the display name of a subscriber in AD, all properties Unity cares about for that mail user in the directory are pushed into SQL again, including the display name. Since each object in the directory only has a single change ID value the directory monitors cannot know what property changed on an object, only that it's been updated. This also means, of course, that if you change a property Unity doesn't care about such as the manager of a mail user, the directory monitors will still note the object has changed and will pull in all the properties for that user into MSMQ and the change writer will update SQL.

Later in this document when we do a walk through of what happens for SA and directory changes we'll cover some of the directory change writer, directory monitor and MSMQ traces to show in more detail how this process works.

AVSQLSynchSvr

The SQL Syncronization Server service is new to Unity 4.0. This service has one job in life and that's to walk through the mail user, locations and distribution list tables in SQL and ask the directory monitors to update the directory if necessary.

This service's primary job in life is for the bulk import utilities including the CSV import, the FullDB Import/Export tools used to migrate from 2.x to 3.x/4.x systems and the DiRT (Disaster Recovery Tools). It is also used by the configuration setup to add the default objects to the directory (Example administrator, example subscriber, etc.). This allows applications to add users and distribution lists to SQL and then kick off the syncher to either find the corresponding user/DL in the directory and bind to it or create a new subscriber/DL in the directory as necessary.

It can also be used to update subscriber properties in SQL directly and then have the syncher talk to the directory monitor to "push" the changes for that subscriber into the directory. This allows applications to talk directly to SQL for doing the vast majority of their Unity administration tasks and then ask the syncher to make sure the directory is up to date. As such it is no longer strictly necessary to interface with the DOH components or update the various directories supported by Unity when writing administration applications for Unity.

The SQLSynchServer DLL has been around since Unity 3.0(1), however it has not been possible to call it from an off-box application. To use the SQLSyncher in 3.x it was necessary for the administration application using it to run on the same server as Unity and create an instance of the SQLSynchServer to use it. This is why applications such as DiRT, ATM (Audio Text Manager), the Import tools and others only ran on box in 3.x. In 4.0 the SQLSynchSvr service can be safely "kicked off" remotely by using an SQL stored procedure which then talks to the SQLSynchSvr service which, in turn, fires up an instance of the syncher itself to execute the updates to the directory through the directory monitors. This is a big jump forward for folks wanting to do their own interfaces into Unity using standard SQL connectivity. This process is covered in more detail in the Administering Unity Programmatically document available on the Documents page of www.CiscoUnityTools.com.

AvLic

The Licensing service has two primary rolls. It is used by the SA to determine if there are enough of a specific license to allow it to be activated on a class of service object (i.e. TTS users). When users are added to the system or moved between classes of service the totals for all the licensed features are updated and if there's not enough of a particular license to accommodate the add/move the action is rejected by the SA. The AVLic service is also responsible for keeping the license counts up to date for the new "pooled licensing" feature in 4.x. Multiple Unity servers can share pool of licenses instead of each one having to have their own set. This is handy for large sites that require multiple Unity servers to handle the number of subscribers they need. A specific Unity server can "opt out" of the license pool if necessary and not contribute to or pull from the shared pool of licenses.

The AvLic service writes these changes into the primary location object which has new fields used to keep track of the total number of subscribers, ports and feature license counts for each Unity server. It writes these changes into SQL directly and then uses the SQLSynchSvr service noted above to have those changes written through to the directory. Other Unity servers in the directory pick up the changes to the location object and pull the updated license information into their local SQL databases. If the Unity server is participating in the pooled licenses with the Unity server that made the change it will update its local license counts accordingly.

AVRepDirSvrSvc

The Report Director Server service is responsible for launching all the reports requested via the SA and providing status updates for queued, running and completed reports that the status monitor web pages display.

The first step in the reports process actually has noting to do with the AvRepDirSvrSvc. A “scavnger” service running under AvCsMgr kicks off every 30 minutes and “scrapes” report related information from the data files in the \commserver\logs directory and stuffs it into appropriate tables in the ReportDB database. The reports engine pulls it’s data from here, not directly from the DATA files as it did in earlier versions of Unity.

Each report offered in the SA is actually a set of DLLs that generate the report in three steps. First, a DLL called the report “pump” extracts the raw information from the tables in the ReportDB database or from the UnityDB database as appropriate. Another DLL called the report “crunch” is then responsible for transforming that raw data into a temporary MDB file that the crystal reports engine can use to generate the final report. Finally a 3rd DLL uses the Crystal Reports engine to generate an HTML or CSV report from that SQL table depending on what the administrator requested when launching the report. The Report Directory Server will launch the appropriate set of DLLs for each report queued up one at a time and then keep an eye on their status which is reported in the status monitor web page. Only one report can be running at any given time, all other reports wait in line to be serviced.

The report director service runs at the lowest possible priority to avoid cutting into the main AvCsMgr service’s ability to process calls in real time. This means reports can sometimes take a while to complete, even if Unity isn’t particularly busy. Originally we tried to check how busy Unity was and based on a value generated for that adjust the priority of the reports process, however this wasn’t real smooth so it was decided to force the process to the lowest priority all the time. To help with this problem, once the report is completed the report director server sends an email to the administrator that requested the report which contains a link to the final report under the \commserver\reports share. This is handy since administrators can queue up numerous reports to run and the output will be waiting for them in their inbox when they come in the next day.

It’s important to note that any of the “pump” report DLLs that need access to the directory information, such as the subscriber report that shows the mailbox size of each mail user, goes through the gateway service and “talks” to the DOH running under AvCsMgr to get at this information. Even though reports are hard coded to the lowest possible processing priority if a subscriber report is run during heavy call traffic it will add to the load on the DOH component under AvCsMgr which is, of course, also busy processing calls on the system. The obvious solution to this is to provide a mechanism to schedule a report to run in the off hours, however Unity currently doesn’t offer this. The reports engine is slated for a major overhaul and this will be revisited then.

CSEMSSvc

The Event Monitoring Service is new for 4.0(3) and it replaces the AVGAENSvc service used for notifying administrators of event log activity.

The CSEMSSvc (Cisco Systems Event Monitoring Service) monitors the event logs for specified entries and then executes the notification rule set up by the administrators which can leave a voice mail or email in the subscriber’s inbox or directly emailing out an SMTP port to an external email server in the case that Unity is no longer able to properly communicate with the mail backend it’s connected to. The updated EMS tool has the ability to trigger notification events on specific events, entire classes of events or on any Unity related event as opposed to the old AVGAEN service which could only trigger on a list of specific events entered into its database.

Unity ships with the CSEMSSvc service not installed. Once you run the administration tool for the EMS (found in the tools depot) it will automatically install and activate the CSEMSSvc service on the fly.

If the set action when a "watched" event shows up in the event log is to send a voice mail or an email to a subscriber or a public distribution list, the CSEMSSvc uses the DOH to find the user in the directory and drop the selected message into their mailbox. To get at the DOH it authenticates with the gateway service and uses the running DOH in AvCsMgr.

This was intended as a mechanism for administrators to use for an "early warning" system when something in Unity was having a problem without requiring a full blown SNMP client or the like. In later versions of Unity SNMP triggers will be an option as a notification mechanism in the EMS tool itself among other items.

AvMMProxySvr

The AVMMProxyServer (Active Voice Media Master Proxy Server) service is used in conjunction with the new authentication method offered in Unity 4.0 that allows for media master connections across domains without requiring a 2nd authentication from the client.

As noted above in the discussion about the AVTRAPSvr and DOHMMSvr components running under AvCsMgr, clients wishing to use the media master to playback or record a greeting or voice name via a web client use the Media Master ActiveX control to connect back to the Unity server to do this. When NTLM authentication is being used the ActiveX control can connect directly to the DOHMMSvr component and, if using the phone for a media device via TRAP, also to the AvTRAPSvr component. If the user is coming from a domain different than the one Unity is running in, this will cause another authentication dialog to pop up even though the user may have already authenticated when they first fired up the SA.

Using the Unity authentication mechanism it is not required to use NTLM. However since the media master controls on the client side use DCOM which requires a security token from NT before it will allow a connection to the client, this presents a problem. To get around that problem, once Unity has authenticated the user and decide they're OK, it uses the AvMMProxSvr service to act as a proxy which establishes a DCOM connection to the client and, in turn, gets TRAP and WAV stream handles from the DOH running under AvCsMgr which is then passed through to the client. In this way the client only has to authenticate once and does not need to be logged into an NT domain at all (i.e. Domino users) to use the SA or the new Cisco Unity Personal Assistant pages in 4.0.

AvTTSSvc

Prior to Unity 4.0(1), the AVTTSSvc service interfaced with two TTS engines: TTS3000 and the Real Speak products offered by L&H. Post 4.0(1), only RealSpeak is supported. When conversations request a TTS session to playback the text of an email to a caller over the phone, it passes the MIU a text file to play which, in turn passes it through to the TTS service which, finally, uses the 3rd party TTS vendor APIs to convert that into a WAV stream and plays it out the voice port the conversation is using.

As other TTS vendors and engines are tested and qualified, it should only be necessary to install their engines and update the TTS service to accommodate them. At the time of this writing there are other TTS engines being investigated but none are slated to be released in Unity 4.0.

The CSBridgeConnector service is always installed in Unity 3.1(3) and later however it's only used if the system is using the Cisco Unity Bridge server to talk to other foreign voice mail servers via the analog OctelNet protocol.

The CSBridgeConnector is responsible for updating mail user information on the bridge server itself for both the local Unity server this service is running on and other Unity servers in the directory. Similarly it gets user information about subscriber residing on the foreign voice mail servers using OctelNet. In this way Unity subscribers can address messages to remote users via OctelNet by name or ID and get voice name confirmation back just as if they were addressing another Unity subscriber on the network.

The CsBridgeConnector gets information about adds/moves/changes to Unity subscriber by monitoring the global subscriber table in SQL. When it notices a change it pushes a small subset of information about this user (including their voice name) to the bridge server by sending it an email by going through the DOH's MAL interface. All communication between the bridge server and the CsBridgeConnector service is done via SMTP messages like this. When the bridge server sends an SMTP message indicating an add/move/change, the CSBridgeConnector service uses the DOH to create new mail enabled contacts representing the subscribers on the foreign voice mail server or update existing ones. Going through the DOH to do this ensures that both SQL and Active Directory records are created/updated automatically. However, since the DOH does not provide an interface for deleting objects in the directory (only adding or modifying is allowed for security reasons) it has to go right to Active Directory to delete contacts when a user is removed. Since these are only contacts that cannot have message stores, the risk for deleting objects in this case is low so this work around is allowed.

The reference to Active Directory is deliberate in this case. The Bridge server only works with Unity servers running with Exchange 2000 or mixed Exchange 2000 and Exchange 5.5 systems which use Active Directory. This is currently not supported for pure Exchange 5.5 or Domino systems.

The Cisco Unity Bridge server functionality is covered in some more detail in the Digital Networking documentation for Unity 4.0(x) noted above.

AvNodeMgr

The Node Manager service is responsible for keeping a primary and secondary Unity servers up to date with one another for failover configurations. In a failover configuration the secondary Unity server is dormant but on "hot standby" ready to take over call processing and notification duties if necessary. For more information on the failover configuration and administration process, see the "Failover Guide" in the Unity documentation.

The node manager services on both the primary and the secondary Unity servers in this configuration send each other messages about the states of their services and components on their respective local servers. When the primary server fails for whatever reason (loss of connectivity to the secondary Unity, services/components becoming unresponsive, calls overflowing to the secondary server, administrator manually forcing a failover for maintenance reasons, etc.) the secondary server will take over handling calls and notification duties until the primary comes back on line and a "fail back" event is executed by the administrator.

Besides communicating service and component status back and forth between the primary and secondary servers, the node manager is also responsible for keeping selected data in the registry, the greetings and voice name WAV files synchronized between the two servers. When a new greeting is recorded for a call handler on the primary Unity server, for instance, it's important that the updated greeting for that handler also be copied over to the secondary Unity server as quickly as possible so the systems are not out of sequence should the primary fail over. SQL Synchronization is configured between the two servers to keep the Unity databases identical between the boxes as well, however the node manager is not involved in this process directly.

TomCat

The TomCat service is responsible for generating the HTML pages for the new Cisco Unity Personal Assistant pages offered in Unity 4.0. The TomCat service uses JSP and Java Scriptlets to kick out web pages based on information pulled from the directory and mailstore via the DOH.

Currently only the Cisco Unity Personal Communication Assistant pages use this service, the SA and status monitor pages still use ASP code to generate the HTML pages as they have in previous versions of Unity. If the system administration group decides to follow the client team down the Java road is not known at this time (at least by myself). However the processes used for the SA/SM and the CUPCA page generation from a high level is actually fairly similar even if they do use different code to generate the final HTML sent back to the client.

When a client requests a CUPCA page from their desktop, the request hits IIS and, based on the URL requested, redirects it to the TomCat server running on the box. The TomCat server connects to the DOH by going through the AvXML component running under IIS and authenticating through the gateway service much like the SA uses the SADbConn component to do the same thing. It uses the DOH to get at both directory information about the requesting subscribers' account as well as to get at the messages in their mailbox. Using this information it constructs the HTML page and sends it back to IIS which, in turn, feeds it to the requesting client's browser.

One big difference between the CUPCA and the SA, however, is that the CUPCA will not support NTLM at all, only the new Unity authentication offered in Unity 4.0. The SA, on the other hand, can optionally support either NTLM or the new Unity authentication mechanism depending on what the customer is more comfortable with.

Process Walkthroughs

Now that we've covered the services Unity adds to the server and the basic components running under those services, lets walk through a couple of scenarios and see those components in action.

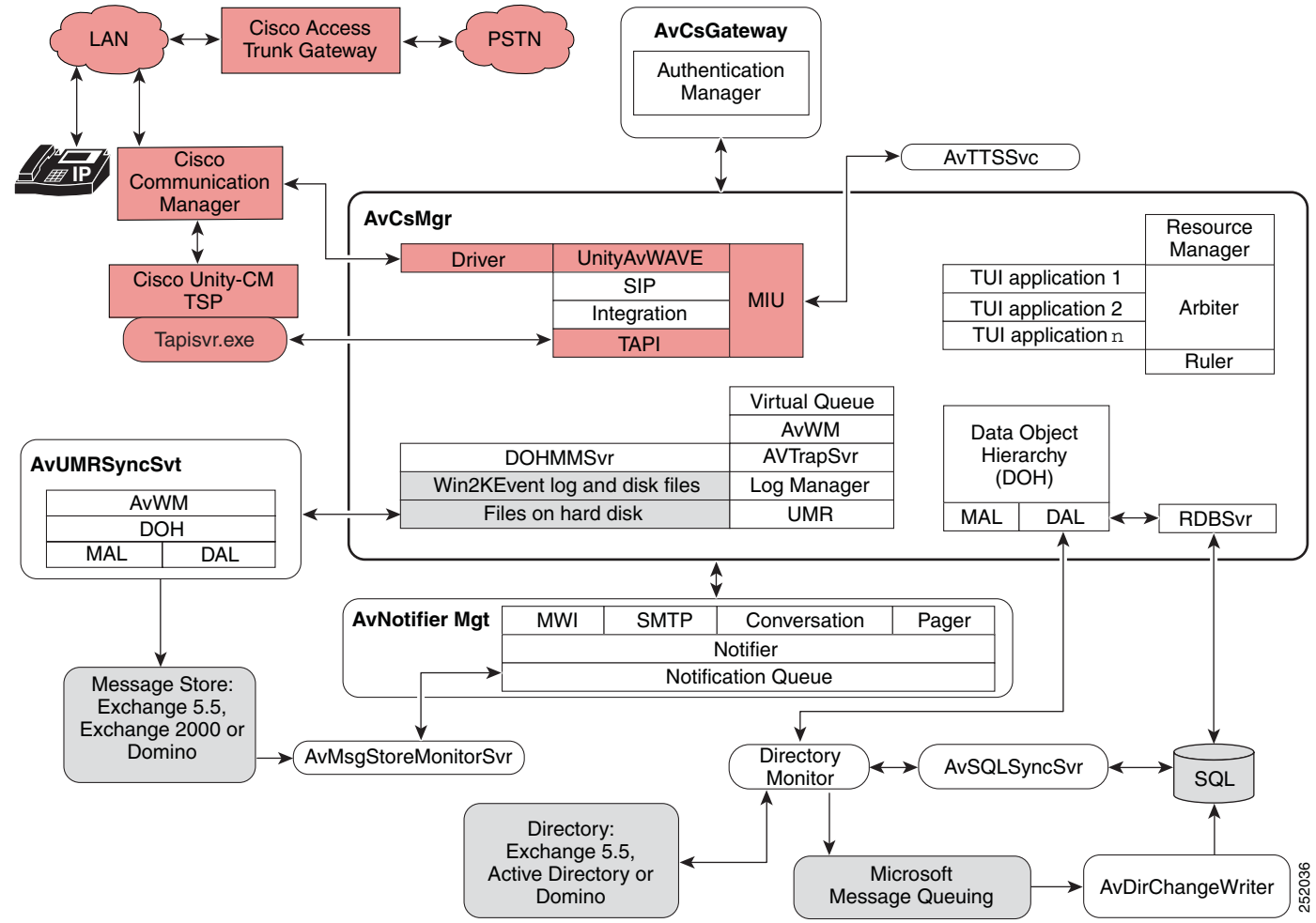
Outside Caller Leaves a Message

Unity uses the selected mailstore (Exchange 5.5, Exchange 2000 or Domino) as its message repository as opposed to storing messages in a local database and pushing/pulling updates in and out by polling the remote store. I'm often asked how this is done and how Unity knows to turn the MWI on or trigger a notification dialout when a new message arrives in the inbox of a subscriber either via the phone or from another subscriber from the desktop. In this section we'll walk through the details of what happens when an outside caller forwards into Unity after dialing the extension of a subscriber's phone and leaves a voice mail message for a subscriber which then results in a new MWI event being triggered.

For this walk through we'll have an outside caller at 206-555-5555 call the subscriber at extension 1234 via DID. We'll assume that their extension rang 4 times and then forwarded ring-no-answer into Unity.

Figure 1-2 shows the first step in a new inbound call where the call forwards ring no answer from extension 1234 to a hunt group of lines which includes the voice ports assigned to Unity. The Call Manager sends information about the call via the Skinny to the AvSkinnyTSP which talks to the MIU via TAPI. The MIU establishes an audio channel via the VoIP WAV driver.

Figure 1-2 Unity Call Flow



The MIU now has all the information about the call including the calling number, forwarding number, the dialed number, which Unity port ID was used to handle the incoming call and the fact that the call forwarded RNA to Unity. This is all associated with the "call object" that clients further up stream have access to, including conversations (TUI applications). For some switch integrations, at this point the call has been answered already (i.e. we've gone off hook on the call) and an audio channel is established. For analog integrations the MIU does not get details about the call until it goes off hook and receives a stream of DTMF digits that indicate the same data the Call Manager sends via its Skinny interface. As such it's not possible for Unity to now decide based on the call information that it's not going to take the call and redirect, or "deflect", the call to another number. The cow, as they say, is out of the barn and we're going to have to handle the incoming call. In the case of a Call Manager integration, however, all the data arrives via the TSP.

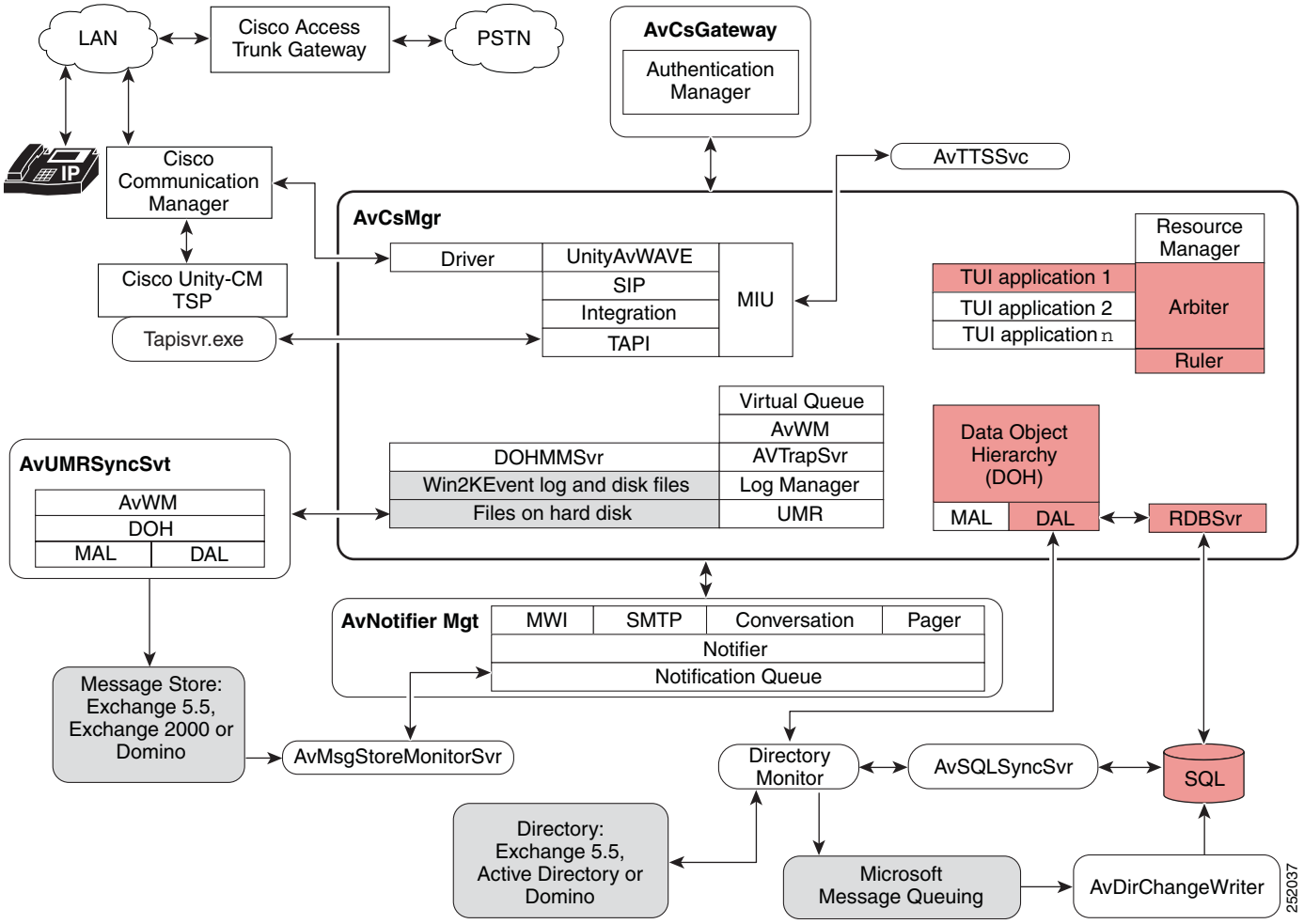
This may seem like a strange detail but it can actually bite you if you're not careful. For instance if you're using an Analog integration (or a dual switch with Call Manager and an analog switch) and have all subscriber phones forwarding to a hunt group that includes all the Unity ports and yet some of those ports are not set to allow incoming calls, you can easily get into a situation where an inbound call is answered but cannot be handled and the user simply gets hung up on. For this reason if you are going to dedicate some ports to MWI or notification dialouts only, for instance, be sure those ports are not

included in the hunt group subscriber phones are forwarded to. This also helps avoid “collisions”, a situation where Unity is attempting to go off hook to do a dialout at the same time an incoming call is trying to use that same port.

There are a very large number of MIU related traces you can turn on that show a huge array of information coming from the switch, the WAV driver, TAPI etc., far beyond the scope of document. If you’re interested in getting more details about what’s happening under the covers with the MIU, however, there is a very good and highly detailed MIU troubleshooting document that talks about the various traces you can turn on and how to read their output. If you’re interested, you can get a copy of this document here: http://www.ciscounitytools.com/Applications/DiagnosingMiu.doc

The next step shown in Figure 1-3 is the MIU handing the new call off to the Arbiter component for initial processing. The arbiter in turn uses the ruler component to process the routing rules configured for the system to decide where this call should go. In this case we’ll assume default settings here and the “attemptForwardToGreeting” conversation gets called with the forwarding number of 1234 passed in. This extension corresponds to a subscriber so the PHGreeting conversation is evoked with the call handler for the subscriber “1234”.

Figure 1-3 Arbiter Component of Call Flow



In the "Audiotext Applications In Unity document" noted we take a look at the arbiter traces here that shows which routing rules are evaluated and how the inbound calls are routed. The document also take a look at the CallViewer output for troubleshooting where calls are ending up in the system and why, which is a real common challenge in the field especially for sites creating numerous complex routing rules.

Once the PHGreeting conversation is invoked with the call handler for the subscriber at extension 1234, it evaluates the greeting rules for that handler. In this case we'll assume the subscriber has a standard greeting recorded and that the after greeting action is set to take a message which is the default behavior. The Data Object Model document noted above goes into more detail on all the parts of a subscriber, call handlers and other objects the make up the Unity directory including greeting rules, contact rules etc., mentioned throughout this document.

The PHGreeting conversation works through the MIU to play out the appropriate greeting WAV file stored under the \Commserver\StreamFiles directory. Assuming the user enters no digits during this process, the conversation then proceeds to the after message action which is set to take a message.

A quick note about full mailboxes and the Unity messaging process here. In Unity 3.1(5) and later the PHGreeting conversation can be configured to make a quick check to ensure the subscriber the message will be sent to does not have a full inbox before taking a message for that user. Since running out and checking the status of a mailbox across the network can sometimes be slow this check is off by default and can be enabled via a registry edit (see the Advanced Settings tool for this). If Unity takes a message

from an outside caller for a subscriber that has a full mailbox, the UMR will end up delivering it to the mailstore which, of course, will end up rejecting its delivery to that box and will bounce the mail back to the Unity Messaging System account. The message ends up getting forwarded to the Unaddressed Messages public distribution list for handling in that case.

Note The check for a full mailbox is ONLY made when messages are being left for individual subscribers by an outside caller. Subscriber to Subscriber messages are always sent via the mailstore back end regardless. If, for instance, you log in as a subscriber and send a message to another subscriber that has a full mailbox, it will be sent just as if you delivered it via an email client. The mail server will then send it back to you NDR (Non Delivery Receipt) indicating the user's mailbox is full.

Note You may wish to look at the Message Store Manager tool available for download off www.CiscoUnityTools.com. This tool can help you manage your user's message storage usage, run reports, delete messages based on various rules such as the read status, age, urgency etc. You can read the help file and check out the training videos for the tool on it's home page for more detail

Note Assuming the subscriber at extension 1234 has been good and kept their inbox to a reasonable level, the record request is sent by the PHGreeting conversation to the MIU and the caller records a message. Assuming the recording is at least 1 second long it's considered a valid message. The MIU takes care of trimming leading and trailing silence so if the caller said nothing the silence is trimmed and the resulting recording is discarded. In some cases with PBX integrations if a user hangs up instead of recording a message and the disconnect event doesn't come down right away, some "clicking" noise gets recorded as a message and left for the subscriber. To help with this scenario it's possible to edit this minimum recording time on the SA under the Configuration\Recordings page, however you should exercise extreme caution if you opt to bump this value up from 1 second since it's entirely possible for folks to leave legitimate messages that are 2 or 3 seconds long.

Once the MIU has completed recording the message by timing out on silence or the user enters a DTMF to terminate the record session, the handle to the resulting file is passed back to the conversation. In this case since the message is from an unidentified caller, the PHGreeting conversation goes through the UMR component to leave the message for the target user. If, instead, the caller had either first signed into Unity as a subscriber and then left a message or their extension had corresponded to a subscriber on the local Unity server, the PHGreeting conversation would have instead submitted the message to the DOH which would have delivered it from the sender's mailbox directly. In this way the message is actually from the sender so the person getting the message, either over the phone or via in inbox client, can simply reply to it directly.

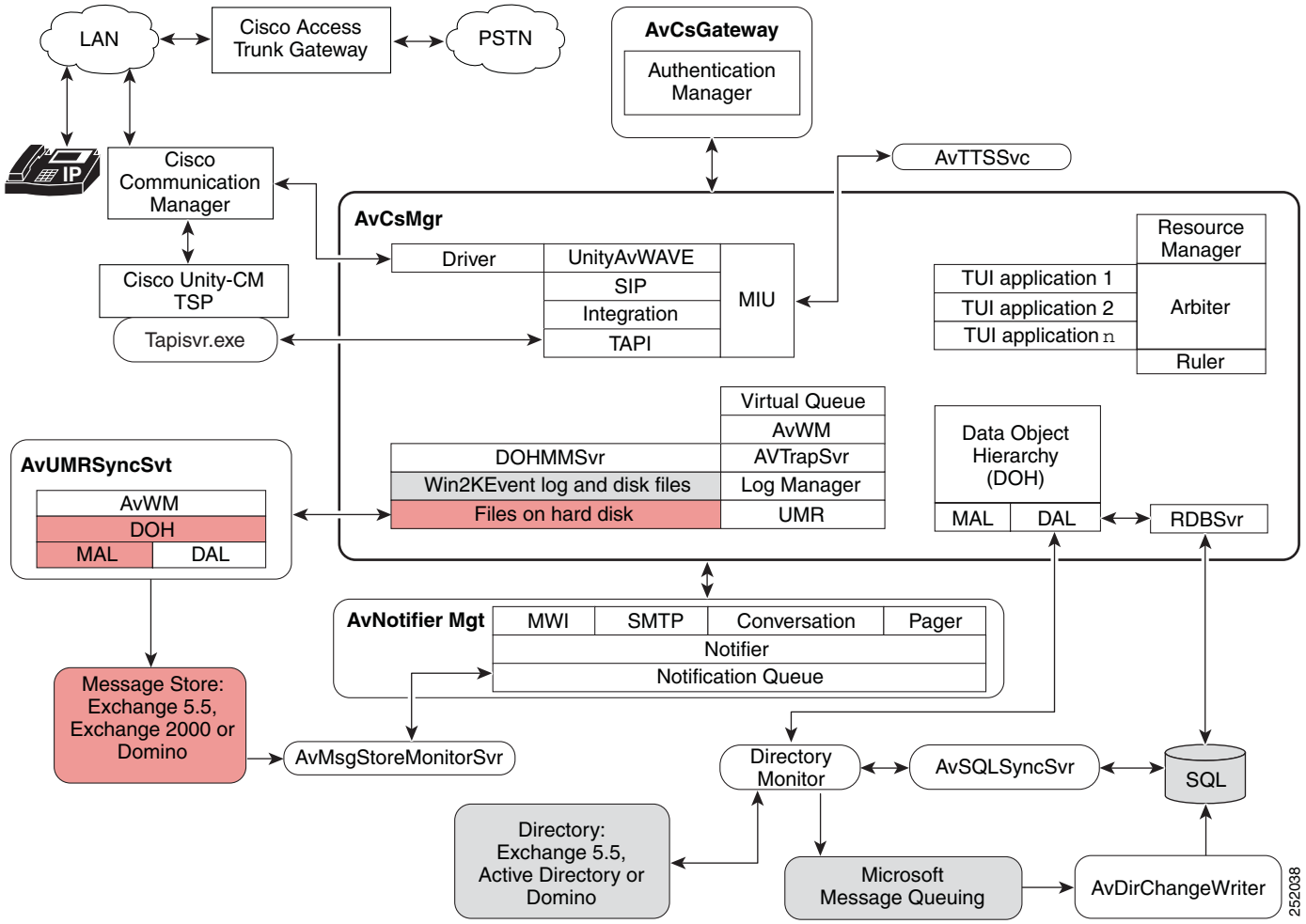
When the UMR process running under AvCsMgr gets the message it writes two files out to the \Commserver\UnityMTA directory. One file is the WAV file for the recorded message itself and another file is the "routing" file that indicates to the AvUMRSyncSvr service where it should ask the mail store to deliver the message when it picks it up. We'll cover that in the next step.

By default the after message action for a call handler is to go to the "say goodbye" call handler which plays a greeting ending the call and then hangs up. Assuming this is the case here, the call then terminates and the line is freed up and is ready to take another call.

The message is not yet in the mailbox of the target subscriber at this point, its sitting as a pair of files in the \Commserver\UnityMTA directory still. When the conversation deposits the message in the UnityMTA directory it sets an event the UMRSyncSvr responds watches that has it run out and check for messages in that folder right away. If the UMRSyncSvr doesn't get such an event within 60 seconds

it'll "wake up" and go check the folder anyway to look for messages that may have synched over from a failover server for instance. The UMRSyncSvr process uses it's own copy of the DOH to deliver the message to the mailbox of the target subscriber using the MAL interface. In the case of Exchange the message is sent from the Unity Messaging System account which resides on the Exchange server selected during the configuration setup.

Figure 1-4 AuUMRSyncSvr



You'll notice in the architecture diagrams that the AvUMRSyncSvr has an instance of the AvWM component running as well. It uses this service to specifically find out if the Exchange server the Unity Messaging System account resides on is currently available or not. If not, it does not attempt to deliver the messages in the UnityMTA directory, instead Unity switches into "UMR mode" and will allow subscribers to get at voice mail messages residing in the UnityMTA directory until connectivity is restored. Note that if the recipient subscriber's home Exchange server is considered off line by the Windows Monitoring service (see above) that message will stay in the local UMR directory until that server is back on line and functioning properly. In versions of Unity prior to 3.1(5) as long as the companion Exchange server was up and running the message would be handed off to Exchange and, in this case, it would sit in the Exchange MTA until the target Exchange server came back on line. By default the MTA attempted to deliver the message every 10 minutes for 24 hours. If after that time it still could not deliver the message it returned it to the sending party NDR which, in this case, is the Unity Messaging System account we send all outside caller message from. These messages then got forwarded

to the Unaddressed Messages public distribution list for handling by an administrator. Since it's not easy to see what messages are currently stacked up in the MTA these messages would appear to end users as being "lost". As such in Unity 3.1(5) and later the message isn't handed off to Exchange unless both the companion Exchange server and the home Exchange server of the target subscriber are both known to be up and functioning.

To see the outside message delivery process a little better, it's instructive to take a look at some diagnostics. For those following along at home, go ahead and open the Unity Diagnostic Tool which can be found in the Tools Depot or the Unity program group. Select "Configure Micro Traces" and go down to the "UMR" section and expand it. You'll see a number of trace options under there, in this case we just want to see when messages are added to the UnityMTA directory and when they are picked up and delivered. Turn on trace #10 – "General UMR Sync Thread" and #12 – "MTA Walk Trough". MTA in this case stands for Message Transfer Agent which is technically an Exchange 5.5 specific term but applies to all the mail store back ends Unity supports. Trace #12 is actually the UMR component that runs under AvCsMgr that the PHGreeting conversation talks to when submitting the outside caller message. Since it runs under the AvCsMgr process it will, of course, log its diagnostic output to the diag_AvCsMgrxxx.txt file. Once you finish the Micro Trace wizard you might want to select the "Start New Log Files" option to cycles the diagnostic trace logs which makes it easier to see what you were looking for.

Once you leave an outside caller message you can open the "diag_AvCsMgrxxx.txt" and "diag_UMRSyncSvrxxx.txt" files in the \Commserver\Logs directory directly and view the output. However the traces are more readable when they're formatted. To do this, again use the UDT and select the "Gather Traces" option to collect the files noted above and save them in formatted output. Once formatted the file is appended with a "_fmtd.txt" extension. All trace output shown in this document is formatted.

The AvCSMgr trace file should contain the following messages:

```
08:14:17:375 (AvUMR_MC,152,UMR,12) MTA Directory: =C:\CommServer\unityMta\ on line 330 of
e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\UnityMTA\AvMtaSession.cpp
08:14:17:376 (AvUMR_MC,152,UMR,12) New MTA Message file
Name=C:\CommServer\unityMta\JLindborg_20020728_081417375_cbe96ed3-5245-47d0-b1ea-f67da7279
c2c on line 101 of e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\UnityMTA\AvMtaSession.cpp
08:14:17:375 (AvUMR_MC,153,UMR,12) CAvMtaMessage::GetAudioStream() : New Message Stream on
line 343 of e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\UnityMTA\AvMtaMessage.cpp
```

Couple things to note about trace output in general before moving on. The lines of code noted by components will always reference drive letters like "e:\…". I'm often asked why since the Unity server may not have an E drive. This is actually the drive configuration of the server that stores and builds the Unity code and has nothing to do with the local Unity install. This information is really only useful to development folks that may be tracking down a problem. The first column will always contain the time/date stamp and the 6th and 7th columns will contain the trace category (UMR in this case) and the trace number that generated the output (12 in this case). This holds true for all diagnostic trace output and can be handy if you're wanting to extract information for a specific trace from, say, the diag_AvCsMgrxxx.txt file which can contain a very large amount of output if you have any number of traces turned on.

In this case you can see the new message that arrived resulted in a file being created in the C:\Commserver\UnityMTA\ directory named "JLindborg_20020710_1819_d702ea8b-5412-4ea2-bc33-9e6c821c51ed". There will be two files with that name in the directory, one with a .WAV file extension which is the voice message itself and one with a .TXT extension that is the routing file that the AvUMRSyncSvr service will use when addressing the message. If you want to see this you can actually just stop the AvUMRSyncSvr service and leave an outside caller message. This wont prevent Unity from running normally other than the fact that outside caller messages wont be delivered while it's off line. If you do this and open the .TXT file:

```
RecipientAlias:JL
To:
Subject:Message from an unidentified caller
Priority:1
Sensitivity:0
Date:1026351166
X-AvMailHandlerId:=?unicode?b?AQAHADAAMwA6AHsANgAxAEIAMAA2ADMANwBFAC0AQQAyADQANwAtADQAQQBF
AEUALQBCADkANQAyAC0ANgAzADAAOABEADkARgA1AEIAQQA1ADUAfQAAAA==?=
```

The RecipientAlias field is what the UMRSyncSvr uses to address the message. If the calling number was collected from the switch and passed on the subject field would actually contain that information, in this case the calling number did not come through in the integration and so the generic "Message from an unidentified caller" is what is used by default. Currently this is not configurable (I get this question often). The priority and sensitivity fields are the same ones you have access to when sending an email using the desktop client. The AvMailHandlerID is the Unicode format for the ObjectID of the primary call handler for the subscriber (if you're confused, hang tight until you cover the Data Object Model document available on www.CiscoUnityTools.com). When the UMR goes to send the message it actually uses this ID to find the primary call handler, then find the mail user associated with it and grabs the data from there to fill in the message send information it passes to Exchange. You can copy and paste this string into DOHPropTest's "Find By ObjectID" dialog and it'll translate it into a regular object ID and find the call handler for you which can be handy for running down problems here.

If you fire up the AvUMRSyncSvr service again (or you left it alone to begin with) it'll pick up the pair of files and deliver the message to the mail store back end. As noted in the diagram above, it does this by spinning up its own instance of the DOH and delivering the message from the Unity Messaging System Account created by Unity setup. If you take a look at the diag_AvUMRSyncSvrxxx.txt file:

```
08:14:26:953 (AvUMR_MC,150,UMR,10) [Thread 0x00000E28] Proceeding to deliver messages from
UMR on line 413 of e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\AvUMRSyncSvr\UMRThread.cpp
08:14:27:031 (AvUMR_MC,150,UMR,10) [Thread 0x00000E28] Message for [JLindborg] in the UMR
on line 995 of e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\AvUMRSyncSvr\UMRThread.cpp
08:14:27:125 (AvUMR_MC,150,UMR,10) [Thread 0x00000E28] Message delivered on line 514 of
e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\AvUMRSyncSvr\UMRThread.cpp
08:14:27:126 (AvUMR_MC,150,UMR,10) [Thread 0x00000E28] No more messages in the UMR on line
615 of e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\AvUMRSyncSvr\UMRThread.cpp
08:14:27:125 (AvUMR_MC,150,UMR,10) [Thread 0x00000E28] No messages in the UMR wait till we
have more messages on line 339 of
e:\views\Unity3.1.3.23\un_Conv3\UnityUMR\AvUMRSyncSvr\UMRThread.cpp
```

You can see it picked up the message bound for the subscriber with the alias of Jlindborg, then noted it delivered it and then notes that there are no more messages in the UnityMTA directory to be delivered. The last line there is just a note that it's going to take a nap until more files arrive in the UnityMTA directory.

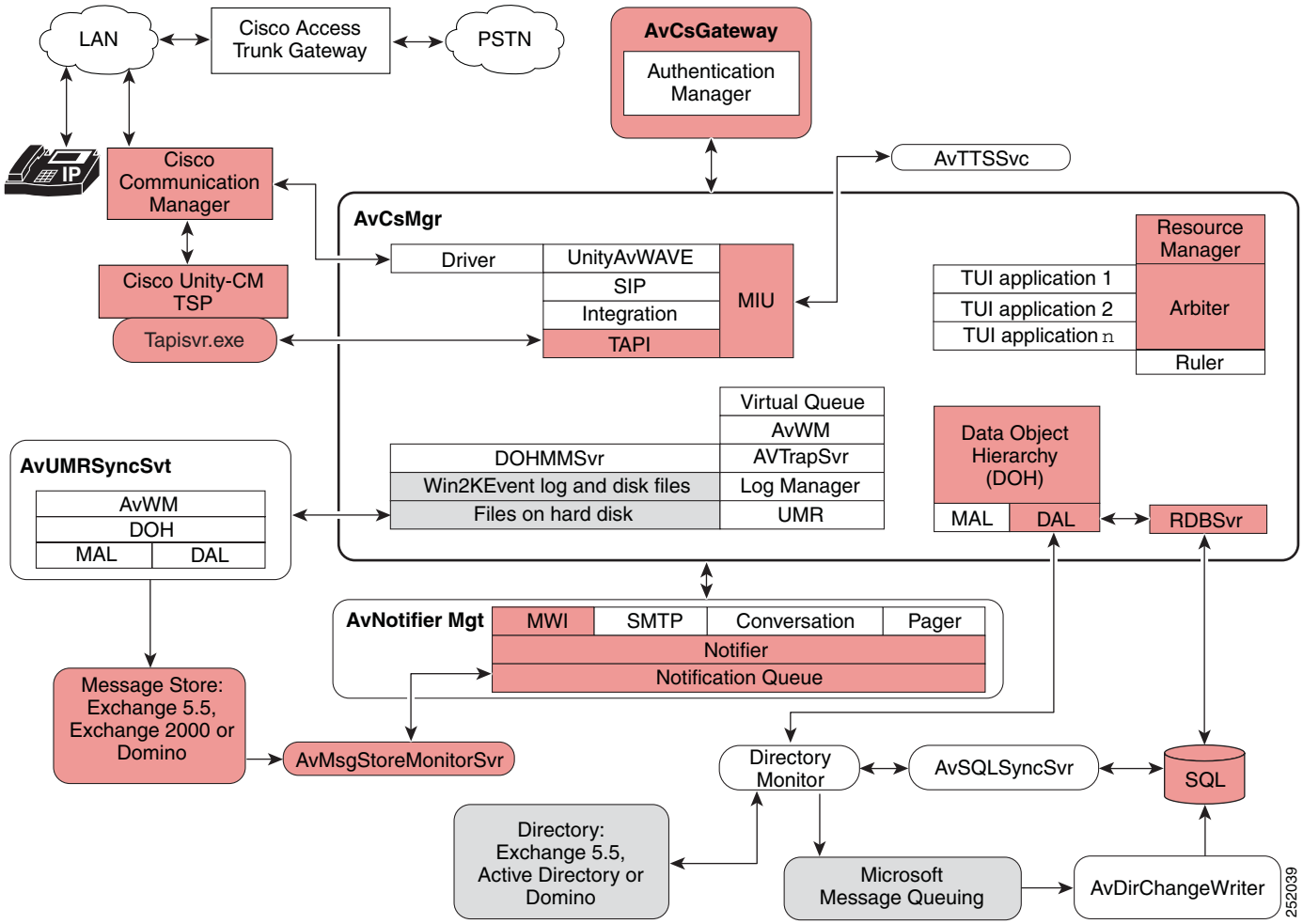
If for whatever reason the UMRSyncSvr service gets an error back from the mail server (for instance the RecipientAlias does not exist in its directory) when it attempts to deliver the message, the pair of files will be moved to the \Commserver\UnityMTA\Failed\ directory. It will attempt to deliver these failed messages periodically and log errors to the event log to let you know there's a problem. In earlier versions of Unity a failed message delivery would "back up" the UnityMTA directory and nothing behind the bad message would get through. Starting in 3.1(4) the failed directory was used as a mechanism to get around this. The first step in troubleshooting such a problem is to pop open the routing .TXT file and see where it's attempting to send the message and making sure that destination exists and is accessible. As noted above, Exchange will not issue an error if the target mailbox is full, it will simply NDR the message back to the sender. The conversations can be set to check the full mailbox condition for outside caller's wanting to leave messages to help prevent problems here.

In the case of Domino, the full mailbox state is actually not enforced by the messaging back end as it is in Exchange. As such, Unity does not specifically warn users or prevent sending/receiving messages based on specific mailbox size restrictions. With Exchange we have no choice since the mail store will enforce it regardless. Later versions of Domino may include stricter mailbox quota enforcement and at that point Unity will need to check those limits and act accordingly.

Assuming the message is handed off to the mailstore properly in the last step and that it then delivers the message to the mailbox of the target subscriber, this is where the notifier process kicks in.

First, the AvMsgStoreMonitorSvr service is notified as soon as the message arrives with a 'New Message" event and a message Id. It pushes this information onto the Notification queue which is running in the AvCsMgr process which requires that the AvMsgStoreMonitorSvr first authenticate with the gateway to get at the queue. The notifier then picks this information out of the queue and decides what to do with it. In this case we'll assume subscriber 1234's lamp is not on and there are no notification devices other than the MWI device itself active. As such, the notifier adds the MWI on request to the MWI device manager. The MWI device manager then asks the resource manager if there's a port available to do an MWI request – it will make this same request even if the MWIs are done via the serial port or via a voice port. Assuming a port is available, the resource manager reserves it for the MWI device manager and passes the reserved port ID back which the MWI device manager then sends to the Arbiter when it requests the actual dialout to turn the lamp on. The MWI device manager will then wait until the Arbiter returns success or failure for the MWI request and if it fails, it will take care of requeing the request to try again at a specified interval for a set number of tries. It will only consider the MWI attempt failed after the total number of retries has been exhausted. Once the MWI request has either succeeded or failed it will push this result back onto the Notification Queue to let the notifier know it's been completed. The MWI device manager also takes care of updating the subscriber's record to indicate we think the MWI lamp is on which is what shows up on the messages page in the SA for the subscriber as the current "Indicator Lamps" status. It uses the DOH to write this through to the local SQL record, the lamp status of the subscriber is not written through to the directory so the directory monitor is not involved here.

Figure 1-5 AvMsgStoreMonitorSvr



This step covers a lot of ground, lets again turn some traces on for the components in question and walk through the output you'd see from the example scenario above. In this case we want to turn on traces for the notifier and the MWI device manager running under the AvCsMgr service. In the Unity Diagnostic Tool return to the micro traces section and in the Notifier section turn on trace #12 (MWI Device) and #21 (NotifyQ). Both these traces, of course, write out into the AvCsMgr diagnostic file since these processes run under that service. Popping open that diagnostic file you'll see something like this:

```
08:14:27:937 (AvNotifier_MC,1146,Notifier,21) [Thread 0x00000DA8] NotifyQ popped
eNOTIFYQ_ACTION_MSG_NEW [2], mailbox='cn=JLindborg cn=Recipients ou=First Administrative
Group o=E2KDomain', arg1=2, arg2=3, arg3=1, varMessageData=
(UnexpectedVariantType:8209!!).
08:14:27:938 (AvNotifier_MC,1158,Notifier,12) [Thread 0x00000DA8] JL:MWI-1(1189), 1
messages (message just Added), current status Off, current attempt None
08:14:27:937 (AvNotifier_MC,1113,Notifier,12) [Thread 0x00000DA8] Queued MWI task for
mailuser=JL, extension=1189, status=On
08:14:27:984 (AvNotifier_MC,1108,Notifier,12) [Thread 0x00000D7C] MWI Device - MWI Entry
AV_MWI_ON Received: Task JL 1189 timestamp 1027869267, Port 4
08:14:30:812 (AvNotifier_MC,1146,Notifier,21) [Thread 0x00000DA8] NotifyQ popped
eNOTIFYQ_ACTION_MWION_COMPLETE [7], mailbox='cn=JL cn=Recipients ou=First Administrative
Group o=E2KDomain', arg1=4, arg2=0, arg3=0, varMessageData='MWI-1' (VT_BSTR).
08:14:30:843 (AvNotifier_MC,1113,Notifier,12) [Thread 0x00000D7C] Completed MWI task for
mailuser=JL, extension=1189, status=On
```

The first entry is the notifer process under AvCsMgr picking up the message that the AvMsgStoreMonitor service wrote to the notifier queue when the new voice mail message arrived in the mailbox. The Notifier then takes action on that message by adding an item to the MWI device manager's queue (the 2nd message in the trace). The MWI Device manager then issues an MWI ON event which completes and it then pushes an event back onto the notifier queue to that effect (the eNOTIFYQ_ACTION_MWION_COMPLETE message). What's not shown in the above traces is the notifier acting on the MWI ON complete message and using the DOH to update the subscriber's record to indicate their lamp is now on (at least so far as Unity knows).

One thing to note is that you won't always get the specific mail store event from the message store monitor (the eNOTIFYQ_ACTION_MSG_NEW event in the first line of the diagnostic output). Sometimes if Exchange is busy or several events happen at once (i.e. multiple messages are deleted or marked read at the same time) a simple event that indicating "something changed" will come across and the notifier will need to filter the entire inbox instead of acting specifically on the message state change received. Under normal circumstances, however, the message store monitor will get a specific new\updated\deleted message notification which includes the message type.

If the above message were left on a Domino system the message monitoring traces would look a little different. The following traces are pulled from a Unity server hooked to a Domino back end. They show a new message arriving and then being read 10 minutes later:

```
15:45:50:094 (AvDominoMonitor_MC,1093,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received Notification packet from dna-test1:
<notification><handlerid>unitynotifier</handlerid><applicationdata></applicationdata><user
></user><database>bQBhAGkAbABcAGYAbABkADEALgBuAHMAZgA=</database><event>NewMessage</event>
<newmessagedata><type>V</type><priority>N</priority><sender>ZgAyAF8AbAAyAC8ARABuAGEAZABvAG
0AJQBjAG8AbQAlAGMAbwBtAAA=</sender><subject>TQBlAHMAcwBhAGcAZQAgAEYAcgBvAG0AIABVAG4AaQB0AH
kA</subject><id>2330</id></newmessagedata></notification>
15:45:50:093 (AvDominoMonitor_MC,1096,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received NewMessage Notification packet: Type:V; Prio:N; From:f2_l2/Dnadom%com%com;
Subj:Message From Unity; ID:2330
15:45:52:562 (AvDominoMonitor_MC,1093,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received Notification packet from dna-test1:
<notification><handlerid>unitynotifier</handlerid><applicationdata></applicationdata><user
></user><database>bQBhAGkAbABcAGYAbABkADEALgBuAHMAZgA=</database><event>CountChange</event
><countchangedata><count>2</count></countchangedata><id>2330</id><priority>N</priority><ty
pe>V</type><eventtype>1</eventtype></notification>
15:45:52:563 (AvDominoMonitor_MC,1094,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received CountChange Notification packet: Cnt:2; ID:2330; Prio:N; Type:V; EType:1;
16:10:18:074 (AvDominoMonitor_MC,1093,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received Notification packet from dna-test1:
<notification><handlerid>unitynotifier</handlerid><applicationdata></applicationdata><user
></user><database>bQBhAGkAbABcAGYAbABkADEALgBuAHMAZgA=</database><event>CountChange</event
><countchangedata><count>1</count></countchangedata><id>2330</id><priority>2</priority><ty
pe>V</type><eventtype>6</eventtype></notification>
16:10:18:075 (AvDominoMonitor_MC,1094,DominoMonitor,12) [Thread 1756] [Thread 0x000006DC]
Received CountChange Notification packet: Cnt:1; ID:2330; Prio:2; Type:V; EType:6;
```

Messages come in pairs – one showing the raw XML-like message coming in and a second showing what that message means to Unity. You can read through the first 4 messages there and reasonably easily tell that a new message has arrive for the user "f2_l2" (yes, those are bulk created users names… not real creative). You can see the type of "V" there which stands for voice, of course. You can also see F, E, I, R, O for Fax, E-Mail, Invitation, Report, and Other in this field. The message count noted in the 3rd and 4th entry comes after the new arrival and simply notes that there are now 2 unread voice message counts.

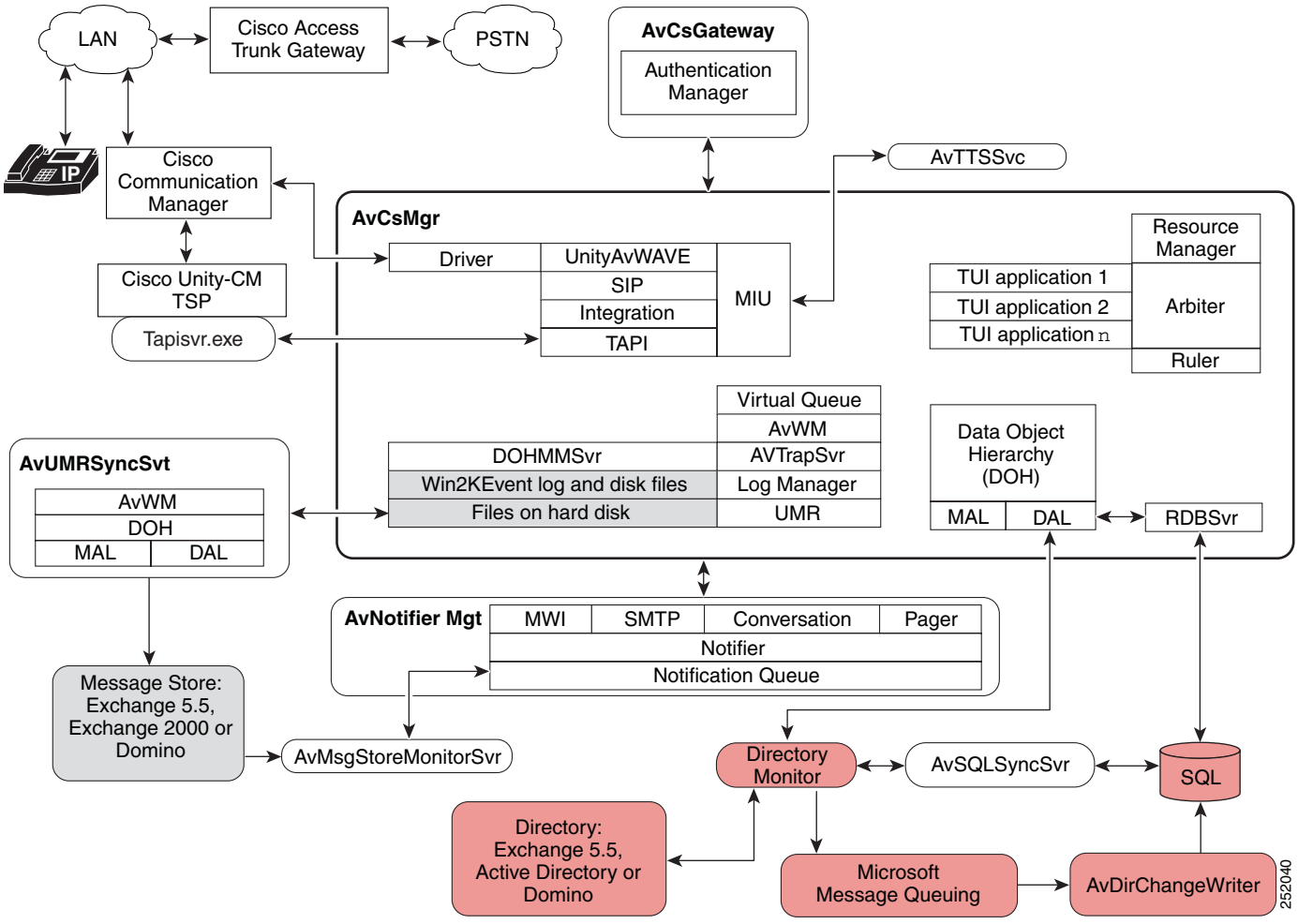
The deletion event (the last two entries) is simply noted as a message count state change. You can see that the count for voice mail messages is now 1. As such the notifier knows to leave the MWI on, it'll turn it off once that count gets to 0. The count returned in the packet indicates the number of unread

messages for the specified type so if the type had been “E” that would indicate the number of unread email messages which may be of interest for notification triggers setup to deliver a page, for instance, based on email messages in the inbox.

Change to Mail User In Directory

If an administrator changes, for instance, the display name of a subscriber in the directory (i.e. through Active Directory Users and Computers), how does Unity update the local SQL database to reflect that change? This is where the directory monitor services come into play. The directory monitors are watching objects in the directory for updates, additions and deletions and pushing that information into the MSMQ queue where the avDirChangeWriter service pulls from to update SQL. Lets run through a quick example.

Figure 1-6 Mail User Change



Lets say an administrator changes the display name of Jeff Lindborg who also happens to be a subscriber on the local Unity server. Lets turn on some traces and take a look at what the process looks like. In the case of Active Directory (Exchange 2000) we’ll want to watch the AvDSAD service and the AvDirChangeWriter service to see what’s changes are being noted in the directory and what the

AvDirChangeWriter is actually pushing into SQL. Again, in the Unity Diagnostic Tool go to the micro traces section and in the AvDirChangeWriter section turn on #15 (All queued requests) and in the DSAD section turn on #11 (changes queued). Go ahead and make a change to the display name of a subscriber and wait 2 minutes. Remember the synch time for the avDSAD service is 2 minutes, for the AvDSADGlobalCatalog service it's 15 minutes. You can force the sync to take place immediately using the DOHPropTest tool which is covered in more detail in the Unity Data Object Model document noted above.

What you'll see in the AvDSAD diagnostics is very simple:

```
12:24:40:953,AvDirSynch_MC,1127,3652,-1,DSAD,11,AVOBJECTTYPE_MAILUSER,CN=JL CN=Users
DC=LindborgLabs DC=com
```

This is just a notation that the record for Jl has changed and that the information in the directory is newer than what we have locally in SQL. This is determined through a change Id stored in the directory which we also keep track of locally. Whenever any change is made to an object in the directory, this number is incremented. If the AvDSAD service, in this case, notes that it's newer than the number we have stored, it pushes all the information for that user that Unity cares about into the MSMQ store so it'll be written through to SQL. The details of which values are actually being pulled from the directory can be see in the AvDirChangeWriter diagnostic output:

```
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,CHANGEREQ: [LogId] 392
[ReqType] AV_DIRCHANGE_MODIFY | AV_DIRCHANGE_DOMAIN_CONTROLLER [ObjType]
AVOBJECTTYPE_MAILUSER [DirLogId] 423 [LocLogId] 424 [PropIdsId] 425 [PropValsId] 426
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 423  [vt]
VT_BSTR [Data] 4DF880B4E68A4F4B96394F4D97B49698
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,OBJID: [LogId] 424 [Hdr]
0x00070001 [ObjType] AVOBJECTTYPE_LOCATION [ID] {FEC3952E-3119-4FAF-82E2-A6534E193635}
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425
[NumElems] 27
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_ADDRTYPE [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_ALIAS [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_ALTERNATE_DTMF_IDS [ExpType] VT_ARRAY | VT_VARIANT
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_AMIS_DISABLE_OUTBOUND [ExpType] VT_BOOL
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_DIRECTORY_ID [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_DISPLAY_NAME [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_DTMF_ACCESS_ID [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_RECIPIENT_EMAIL_ADDRESS [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_FIRST_NAME [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_LAST_NAME [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_LIST_IN_DIRECTORY [ExpType] VT_BOOL
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_LOCATION_OBJECT_ID [ExpType] VT_ARRAY | VT_UI1
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAIL_DATABASE [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAIL_SERVER [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAILBOX_ID [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAILBOX_SEND_LIMIT [ExpType] VT_I4
```

```
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAILBOX_SEND_RECEIVE_LIMIT [ExpType] VT_I4
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAILBOX_USE_DEFAULT_LIMITS [ExpType] VT_BOOL
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_MAILBOX_WARNING_LIMIT [ExpType] VT_I4
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_OBJECT_CHANGED_ID [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_PRIMARY_FAX_NUMBER [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_REMOTE_ADDRESS [ExpType] VT_BSTR
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_SID [ExpType] VT_ARRAY | VT_UI1
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_SID_HISTORY [ExpType] VT_ARRAY | VT_VARIANT
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_SMTP_ADDRESS [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_VOICE_NAME_DATA [ExpType] VT_ARRAY | VT_UI1
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,IDLIST: [LogId] 425 [PropId]
AVP_XFER_STRING [ExpType] VT_BSTR
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_ARRAY | VT_VARIANT [Dim] 1 [ElemSize] 16 [NumElems] 27 [LBound] 0 [Data(first 32
bytes)] 08 00 00 00 00 00 00 00 84 A9 16 00 00 00 00 00 08 00 00 00 00 00 00 00 B4 3E 18
00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] EX
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] JLindborg
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] 4df880b4e68a4f4b96394f4d97b49698
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] Jeff Lindborg
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] 1189
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] cn=JLindborg cn=Recipients ou=First Administrative Group o=Lindborg Labs
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] Jeff
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] Lindborg
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BOOL [Data(entire union)] FF FF 00 00 00 00 00 00
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_ARRAY | VT_UI1 [Dim] 1 [ElemSize] 1 [NumElems] 88 [LBound] 0 [Data(first 32 bytes)] 01
00 07 00 30 00 39 00 3A 00 7B 00 46 00 45 00 43 00 33 00 39 00 35 00 32 00 45 00 2D 00 33
00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data(first 128 chars)] CN=Mailbox Store (BIGBOY) CN=First Storage Group
CN=InformationStore CN=BIGBOY CN=Servers CN=First Administrative Group CN=Admin
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] BIGBOY
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] cn=JLindborg cn=Recipients ou=First Administrative Group o=Lindborg Labs
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
```

```
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BOOL [Data(entire union)] FF FF 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] ?
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_ARRAY | VT_UI1 [Dim] 1 [ElemSize] 1 [NumElems] 28 [LBound] 0 [Data] 01 05 00 00 00 00
00 05 15 00 00 00 57 29 02 4C E7 CB DD 7D 23 5F 63 6B 5A 0E 00 00
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] JLindborg@LindborgLabs.com
12:24:41:266,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_NULL [Data(entire union)] 00 00 00 00 00 00 00 00
12:24:41:265,AvDcw_MC,1049,3748,-1,Directory Change Writer,16,VARIANT: [LogId] 426  [vt]
VT_BSTR [Data] 1189
```

This is a rather long section of diagnostic to include here but it's instructive to see it in its entirety. What you're seeing here is a list of 27 properties that are pulled in from the directory on the object that changed in the directory. They are listed twice: once with the [propID] tag which indicates which property name in the Unity object model the value maps through to and again, in the same order, with the [vt] tag which shows the actual value pulled from the directory. To find, for instance, the display name value you'll have to look at the propID tag which is the 6th item listed. Then go to the 6th item in the vt tags and notice that the actual value is "Jeff Lindborg". The values, meanings and locations of the constants in the propID tag list (i.e. AVP_DISPLAY_NAME) are covered more in the "Unity Data Object Model" document noted above.

Although a total of 27 properties are pulled from the directory, only a small handful of them are visible using normal administrative tools for viewing the directory. The first name, last name, display name and alias round out the list of obvious ones. Most of the information, however, is specific to Unity and is written through "hidden" values in the directory. You can see this with tools such as the ADSI editor but normally they don't show up. Also notice that all 27 values are updated even though in this case only the display name was changed. The directory monitor has no way of knowing which property was changed in the directory, only that the object has changed in some way. It's possible that none of the values we care about were updated, however since the change Id has rolled, the monitor will push the values through anyway.

What happens if a user is deleted in the directory entirely? In previous versions of Unity this used to present a problem because some subscriber data would still hang around unless the administrator first removed them as a subscriber in Unity before removing their record in the directory. This, of course, didn't always happen. In Unity 3.1(x) and later, however, the directory monitor handles this. If you went ahead and then deleted the Jlindborg account out of AD, in this case, you'd see this line in the AvDSAD trace output:

```
12:30:42:671,AvDirSynch_MC,1127,3652,-1,DSAD,11,AVOBJECTTYPE_MAILUSER,CN=JLindborg\
DEL:b480f84d-8ae6-4b4f-9639-4f4d97b49698 CN=Deleted Objects DC=LindborgLabs DC=com
```

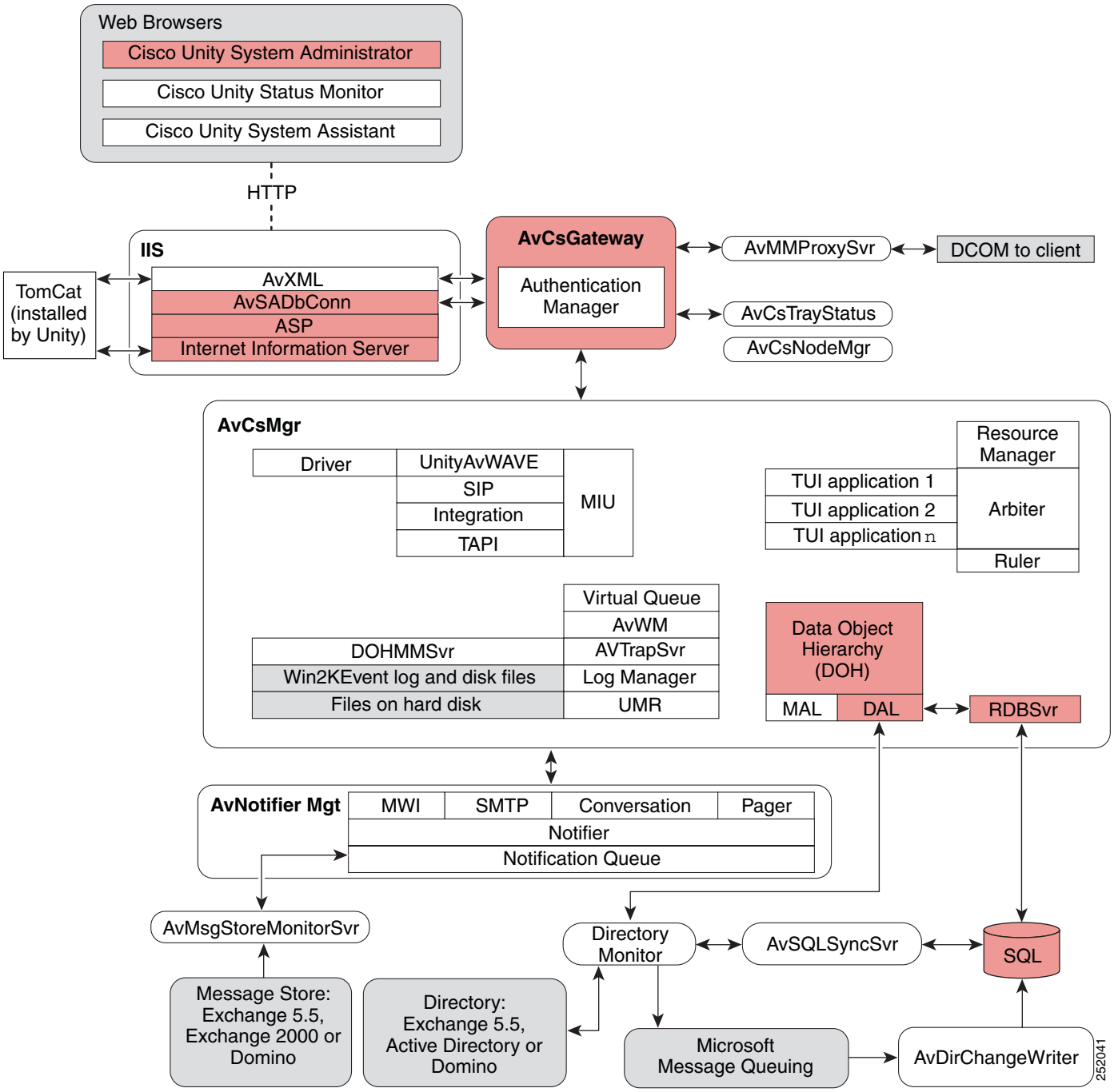
The "DEL: xxx" notation there indicates this is a deletion request. The AvDSAD then pushes the information onto the MSMQ and the AvDirChangeWriter deletes the appropriate record from the Subscriber table in the UnityDB SQL database. Triggers in SQL take care of cleaning the rest of the tables in the database of information related to that subscriber such as their call handlers, transfer rules, greeting rules etc. These structures are all covered in the "Unity Data Object Model" document available on www.CiscoUnityTools.com.

Before moving on from this topic, it's important to note that the monitors are also pulling in information about subscribers, location objects and distribution lists that are being added/changed/deleted on other Unity servers in the directory. In the case of Active Directory this is done by the AvDSGlobalCatalog service which watches the entire forest for objects of interest. When a mail user, for instance, is found that's "tagged" as a subscriber it pushes a very similar looking "information push" request onto MSMQ for the AvDirChangeWriter service to pick up and push into the GlobalSubscriber table in the UnityDB database in SQL. This information is used to provide the digital networking functionality which you can learn more about in the Digital Networking documentation noted above.

Administrator Updates Subscriber in SA

This flow walks through an Administrator logging in via the SA and making a change to a subscriber that then gets pushed to the directory.

Figure 1-7 Subscriber Updates in SA



The first step is the Administrator firing up the Unity SA URL from their Internet Explorer client at their desktop. The URL includes the server name of the Unity server they want to connect to. This is subtle but important. This requires DNS to be functioning properly since even if you resort to using the IP address of the server subsequent URLs generated by links from within the SA will insert the server name and, as such, the SA will not function properly.

The Unity setup registers pages for the SA, AA, Status Monitor and the new CUPCA (Cisco Unity Personal Communication Assistant) web pages with IIS on the local server. In this case the administrator is firing up the SA which is "http://servername/saweb". If Unity is configured to use NT security (the only option for systems prior to Unity 4.0) IIS is going to ensure that the user is authenticated first. If the user is logged into the local domain or is logged into a domain that's trusted by the local Domain, the process proceeds. If the users is not logged into a trusted domain then they will get a challenge and response login dialog before being allowed to proceed. It's important to note that Unity is not involved in the process at all up to this point. When the Unity web pages are added to IIS during the setup process they are configured such that only authenticated users can access them, no anonymous or clear text authentication is allowed. If, on the other hand, the system is configured for the new Unity authentication method, the user will get a web page where they have to provide their login name and password stored on the local Unity server before proceeding. It's possible to allow clients to store login name and/or password for defined periods of time on their client machines but by default this is disabled. This authentication model allows sites not using Windows NT or who are using multiple domains that don't trust each other to more easily gain Unity client access. More details on the authentication mechanisms available for Unity 4.0 are covered in the installation guides for Unity 4.0(x) which can be found here: http://www.cisco.com/univercd/cc/td/doc/product/voice/c_unity/unity40/inst/index.htm

Once the user is authenticated, the SA then uses the AvSADbConn.DLL from the IIS process to connect to the DOH in the AvCsMgr process to see if the user has administration rights on the local Unity box. If the user resolves to a subscriber that has a Class of Service setting that allows SA access or they are "mapped" to a local user that has such rights, they are allowed to proceed. The "mapping" process is a simple matter of adding the directory ID of a domain account into a table in SQL that indicates the user has the admin rights of a local subscriber (usually the built in "installer" account created by Unity during setup). This mechanism is in place so that administrators responsible for maintaining multiple Unity servers on a network can easily gain SA access on all the boxes without having to create separate accounts added as subscribers to each to do it. The tool for adding such mappings is the "GrantUnityAccess" application which you can find in the Tools Depot applet on the Unity desktop.

Assuming the user is a subscriber that has administration access, they will see the Unity SA screen pop up and they can now go about their business. In this case lets say they go in and change the display name on a subscriber and then hit save. The SA pages again go through the AVSADBConn.dll to talk to the DOH to request the subscriber's changes be written through. The DOH writes the changes first through to the directory via the directory monitor service and when that's complete it writes the changes through to SQL. If the write through to the directory fails for whatever reason, the changes are not written through to SQL and the SA is passed back an error.

It's important to note here that when the SA requests the subscriber's information be written through, it's all the data for that user, not just the property (display name in this case) that the administrator may have updated. The SA is not granular enough to keep track of which individual properties have been updated for a user. In much the same way that the directory monitor pulls in all data for a user in the directory that has been updated since it doesn't know which specific property has changed, changes in the SA result in a similar block-write going back out.

Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is an application layer protocol that facilitates the exchange of management information among network devices, such as nodes and routers. It is part of the TCP/IP protocol suite. System administrators can remotely manage network performance, find and solve network problems, and plan for network growth by using SNMP.

Instead of defining a large set of commands, SNMP places all operations in a *get-request*, *get-next-request*, *get-bulk-request*, and *set-request* format. For example, an SNMP manager can get a value from an SNMP agent or store a value in that SNMP agent. The SNMP manager can be part of a network management system (NMS), and the SNMP agent can reside on a networking device such as a router.

Cisco Management Information Base (MIB) variables are accessible by using SNMP which facilitates the exchange of management information between network devices. The SNMP system consists of three parts—SNMP manager, SNMP agent, and MIB. You can compile the Cisco MIB with your network management software.

The NMS uses the Cisco MIB variables to set device variables and to poll devices on the internetwork for specific information. The results of a poll can be graphed and analyzed to help you troubleshoot internetwork problems, increase network performance, verify the configuration of devices, and monitor traffic loads.

The SNMP agent gathers data from the MIB, which is the repository for information about device parameters and network data. The SNMP agent also can send traps (notifications) of certain events, to the SNMP manager. The Cisco trap file, mib.traps, which documents the format of Cisco traps, is available on the Cisco host //ftp.cisco.com.

The SNMP manager uses information in the MIB to perform the operations described in Table 1-1.

Table 1-1 SNMP Manager Operations

Operation	Description
get-request	Retrieve a value from a specific variable.
get-next-request	Retrieve the value following the named variable. Often used to retrieve variables from within a table. With this operation, an SNMP manager does not need to know the exact variable name. A sequential search is performed to find the needed variable from within the MIB.
get-response	The reply to a get-request, get-next-request, get-bulk-request, and set-request sent by an NMS.
get-bulk-request	Similar to get-next-request, but fills the get-response with up to max-repetition number of get-next interactions.
set-request	Store a value in a specific variable.
traps	An unsolicited message sent by an SNMP agent to an SNMP manager indicating that some event has occurred.

SNMP MIBs are ASCII text files and contain a list of data objects. Every device in the network must be defined by a MIB or it does not exist in the SNMP network.

When an SNMP device transmits a message, the device is identified with a number string called an object identifier (OID). The OID contains specific characteristics of the managed device and can have one or more object variables.

MIBs convert the OIDs into text which is readable by administrators. The OID gives the organization type and name and other valuable information about each device. MIBs are important because they define available alerts (notifications) and traps and assist in troubleshooting.

To view MIB dependencies including obsolete objects, across Cisco Unified Communications Manager releases, go to the following link—http://tools.cisco.com/Support/SNMP/do/BrowseMIB.do?local=en&step=2&mibName=CISCO-CCM-CAPABILITY

This section contains information on the following topics:

Note SNMP must also be enabled on network devices to allow network management applications such as Unified OM, Cisco Monitor Manager, and Cisco Monitor Director to get information on network devices at configured polling intervals and to receive alerts and faults via trap notification sent by the managed devices.

SNMP Basics

A network that uses SNMP requires three key components—managed devices, agents, and network management software (NMS).

- Managed device—Devices that contain SNMP agents and reside on a managed network. Managed devices collect and store management information and make it available by using SNMP.

  The first node in the Cisco UCM cluster acts as the managed device. In Cisco UCMBE, the server on which Cisco UCM is installed acts as the managed device.

- Agent—Software modules that reside on managed devices. An agent contains local knowledge of management information and translates it into a form that is compatible with SNMP.
  - Cisco Unified Communications Manager uses a master agent and subagent components to support SNMP. The master agent acts as the agent protocol engine and performs the authentication, authorization, access control, and privacy functions that relate to SNMP requests. It contains a few Management Information Base (MIB) variables that relate to MIB-II. The master agent also connects and disconnects subagents after the subagent completes necessary tasks.

    The SNMP master agent listens on port 161 and forwards SNMP packets for vendor MIBs.
  - The Cisco Unified Communications Manager subagent interacts with the local Cisco Unified Communications Manager only. The Cisco Unified Communications Manager subagents send trap and information messages to the SNMP Master Agent, and the SNMP Master Agent communicates with the SNMP trap receiver (notification destination).
- NMS—SNMP management application that runs on a PC and provides the bulk of the processing and memory resources that are required for network management. It executes applications that monitor and control managed devices. Cisco Unified Communications Manager works with the following NMS:
  - CiscoWorks2000
  - HP OpenView
  - Third-party applications that support SNMP and Cisco Unified Communications Manager SNMP interfaces

SNMP version 1 Support

SNMP version 1 (SNMPv1), the initial implementation of SNMP that functions within the specifications of the Structure of Management Information (SMI), operates over protocols, such as User Datagram Protocol (UDP) and IP.

The SNMPv1 SMI defines highly structured MIB tables that are used to group objects that contain multiple variables. Tables contain zero or more rows, which are indexed, so SNMP can retrieve or alter an entire row with a supported command.

With SNMPv1, the NMS issues a request, and managed devices return responses. Agents use the Trap operation to asynchronously inform the NMS of a significant event.

SNMP version 2c Support

As with SNMPv1, SNMPv2c functions within the specifications of SMI. MIB modules contain definitions of interrelated managed objects. The operations that are used in SNMPv1 are similar to those that are used in SNMPv2. The SNMPv2 trap operation, for example, serves the same function as that used in SNMPv1, but it uses a different message format and replaces the SNMPv1 trap.

The Inform operation in SNMPv2c enables one NMS to send trap information to another NMS and to receive a response from the NMS.

SNMP version 3 Support

SNMP version 3 (SNMPv3) provides security features such as the following:

- Authentication—Verifying that the request comes from a genuine source.
- Privacy—Encrypting data.
- Authorization—Verifying that the user allows the requested operation.
- Access control—Verifying that the user has access to the objects requested.

SNMPv3 prevents packets from being exposed on the network. Instead of using community strings like SNMP v1 and v2, SNMP v3 uses SNMP users, as described in the "SNMP Community Strings and Users" section on page 1-42.

SNMP Services

To support SNMP, you must use the following services:

- Cisco UCM SNMP service
- SNMP Master Agent
- MIB2 Agent
- Host Resources Agent
- System Application Agent
- Native Agent Adaptor
- Cisco CDP Agent
- Cisco Syslog Agent

For more information, refer to the *Cisco Unified Serviceability Administration Guide, Release 7.0(1)*.

SNMP Community Strings and Users

Although SNMP community strings provide no security, the strings authenticate access to MIB objects and function as embedded passwords. You configure SNMP community strings for SNMP v1 and v2c only.

SNMP v3 does not use community strings. It uses SNMP users that serve the same purpose as community strings, but provide security because encryption or authentication is configured.

No default community string or user exists.

SNMP Traps and Informs

An SNMP agent sends notifications in the form of traps or informs to identify important system events. Traps do not receive acknowledgments from the destination whereas informs do receive acknowledgments.

Note Cisco Unified Communications Manager supports SNMP traps in Cisco Unified Communications Manager and Cisco Unified Communications Manager Business Edition systems. Cisco Unity Connection SNMP does not support traps.

For all notifications, the system sends traps immediately if the corresponding trap flags are enabled. In the case of the syslog agent, the Cisco UCM alarms and system level log messages get sent to syslog daemon for logging. Also, some standard third-party applications send the log messages to syslog daemon for logging. These log messages get logged locally in the syslog files and also get converted into SNMP traps/notifications.

The following list contains Cisco UCM SNMP trap and inform messages that are sent to a configured trap destination:

- Cisco UCM failed
- Phone failed
- Phones status update
- Gateway failed
- Media resource list exhausted
- Route list exhausted
- Gateway layer 2 change
- Quality report
- Malicious call
- Syslog message generated

Tip Before you configure notification destination, verify that the required SNMP services are activated and running. Also, make sure that you configured the privileges for the community string/user correctly.

You configure the SNMP trap destination by choosing SNMP > V1/V2 > Notification Destination or SNMP > V3> Notification Destination in Cisco UCM, Serviceability Administration.

SNMP Trace Configuration

For Cisco CDP Agent and Cisco Syslog Agent, you use the CLI to change trace settings, as described in the *Command Line Interface Reference Guide for Cisco Unified Solutions* for Release 7.0(1).

Management Information Base

Management Information Bases (MIBs) convert object identifiers (OIDs) that are numerical strings into an ASCII text file. The OIDs identify data objects in a hierarchy. The OID represents specific characteristics of a device or application and can have one or more object instances (variables). Managed objects, alarms, notifications, and other valuable information are identified by the OID and listed in the MIB.

The OID is logically represented in a tree hierarchy. The root of the tree is unnamed and splits into three main branches—Consultative Committee for International Telegraph and Telephone (CCITT), International Organization for Standardization (ISO), and joint ISO/CCITT.

These branches and those that fall below each category have short text strings and integers to identify them. Text strings describe object names, while integers allow computer software to create compact, encoded representations of the names. For example, the Cisco MIB variable authAddr is an object name and is denoted by number 5, which is listed at the end of its OID of 1.3.6.1.4.1.9.2.1.5.

The OID in the Internet MIB hierarchy is the sequence of numeric labels on the nodes along a path from the root to the object. The Internet standard MIB is represented by the OID 1.3.6.1.2.1. It also can be expressed as iso.org.dod.internet.mgmt.mib.

The Cisco MIB set is a collection of variables that are private extensions to the Internet standard MIB II and many other Internet standard MIBs. MIB II is documented in RFC 1213, Management Information Base for Network Management of TCP/IP-based Internets—MIB-II.

Cisco Unified CCX and Cisco Unity support the following MIBs:

- CISCO-UNITY-MIB
- CISCO-CDP-MIB
- CISCO-SYSLOG-MIB
- HOST-RESOURCES-MIB
- MIB-II
- SYSAPPL-MIB
- Vendor-specific MIBs

For descriptions of the supported MIBs, see:

- Chapter 3, “Cisco Unity MIBs”

CHAPTER 2

Managing and Monitoring Cisco Unity

This chapter describes the tools available for managing and monitoring Cisco Unifed CCX and Cisco Unity.

For Cisco Unity, the following tools are available:

- Unity Advanced Settings Tool—This tool allows you to safely edit many of the "hidden registry settings" that commonly need to be modified. These are items that are not on the SA interface and require users to add/edit keys to the registry.
- Alternative Directory Handler Prompts—This is a zip file containing different wording on the prompts that outside callers hear when they go to find a subscriber by spelling their name in the directory handler.
- AudioStat utility—This tool allows audio driver statistics to be viewed in real time. The information provided by AudioStat can help isolate audio quality issues due to packet delay and codec-related problems.

Caution Cisco recommends using this tool only with Cisco TAC supervision.

- Bridge Analog Network And Node Analyzer (BANANA)—BANANA is a stand-alone application that runs on the Cisco Unity Bridge server. It is designed to assist with monitoring and troubleshooting of analog communication between the Bridge and other Octel nodes in the analog network. It also provides detail and summary information of call activity. BANANA contains an administration application called the BANANA admin that allows you to control how BANANA:
  - Extracts information from the call traces on the Bridge server and presents different views of the data.
  - Monitors the call logs for error conditions and logs warnings or errors to the Windows Event Viewer.
  - Generates test calls to any of the Octel systems that are networked with the Bridge server.

  With the BANANA admin, you can also install and configure the BANANA service to do any or all of the above tasks at configurable intervals.
- Cisco Unity Bridge Bulk Node Utility (CUBBNUT)—This utility allows a Cisco Bridge administrator to add, change or delete Unity and Octel nodes via a .csv file.
- BulkEdit Utility—The BulkEdit Utility is designed to allow you to select large numbers of call handlers or subscribers and make changes to them in bulk quickly and easily. Nearly every value you can see and edit via the SA is available to change en mass using BulkEdit as well as a few items not visible in the SA

- Bulk Subscriber Delete tool—This is a wizard tool that allows you to delete large numbers of subscribers from Unity. For example, installations at schools and universities where there is a large number of users that “turn over” at the end of the year. Removing subscribers from Unity one at a time through the SA is time consuming and tedious and it also leaves the user’s directory and messaging accounts in AD and Exchange which may or may not be desirable.

  This tool give you the option of only removing the subscriber information from Unity or removing the directory and messaging accounts for selected users entirely. You can choose subscribers based on their Class of Service membership, extension range, switch association, mailstore association, location assignment (for remote users), node ID (for Bridge users) or you can import them from a custom CSV file.
- Directory Walker—This utility walks the directory and makes the checks on all call handler, subscriber, subscriber template, interview handler, locations and directory handler objects in the database. If there's a problem the string "(error)" will appear in red in the output HTML. Warning strings in yellow that start with "(warning) are also logged for items that you should check on but are not necessarily problems. If an item is automatically fixed, a string that starts with "(fixed) will be logged in green directly under the error to indicate what was done.
- Subscriber Information Dump utility—This utility allows you to quickly dump out specific information about the subscribers in their system to a file they can be viewed or imported into another application such as a database utility or Excel. The file generated automatically creates a header row that lists the data type found in that column of the output for ease of import into other programs. This tool includes both full subscribers (Exchange and Domino) as well as “remote subscribers” (AMIS, VPIM, Bridge, SMTP).

Cisco Unity Reporting

The following tools provide reporting capabilites for Cisco Unity

- Bridge Traffic Analyzer - The Bridge Traffic Analyzer is a tool that allows administrators to obtain traffic data on their Cisco Bridge units to determine:
  - total size of messages sent
  - number of messages sent
  - which servers messages have come from or gone to
  - how long it takes for messages to arrive at their destinations
  - how many analog ports are in use for message transport
  - · How many failures were there sending and receiving messages between various nodes

  This tool allows you to generate the following reports:
  - Port Availability Report
  - Message Queue Activity Report
  - Message Latency Report
  - Node Message Traffic Report
  - Non-delivery Notifications Report

Authentication validates the user ID and password that are submitted during log in. An alarm gets raised when an invalid user ID and/or the password gets used.

CHAPTER 3

Cisco Unity MIBs

This chapter describes the Management Information Base (MIB) that are supported by Cisco Unified Contact Center Express and by Cisco Unity. These MIBs are used with Simple Network Management Protocol (SNMP)

Cisco Unity MIBs

Cisco Unity uses two MIBs:

CISCO-UNITY-MIB

This section contains the text of the CISCO-UNITY-MIB file.

```
CISCO-UNITY-MIB DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE, Unsigned32
      FROM SNMPv2-SMI
   TEXTUAL-CONVENTION, DateAndTime, TruthValue
      FROM SNMPv2-TC
   SnmpAdminString
      FROM SNMP-FRAMEWORK-MIB
   MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
      FROM SNMPv2-CONF
   ciscoMgmt
      FROM CISCO-SMI;

ciscoUnityMIB  MODULE-IDENTITY
```

```
    LAST-UPDATED    "200512120000Z"
    ORGANIZATION    "Cisco Systems, Inc."
    CONTACT-INFO
            "       Cisco Systems
                    Customer Service

                Postal: 170 W Tasman Drive
                    San Jose, CA  95134
                    USA

                  Tel: +1 800 553-NETS

                E-mail: cs-unity@cisco.com"
    DESCRIPTION
        "The MIB Module for the management of Cisco Unity server.
        Cisco Unity is a Unified Communications solution that provides
        advanced, convergence-based communication services. The MIB
        presents provision and statistics information.

            ACRONYMS

        AMIS
          Analog Messaging Interface Standard
        MWI
          Message Waiting Indicator
        TRAP
          Telephone Record and Playback
        TTS
          Text-To-Speech
        UM
          Unified Messaging
        VMI
          Visual Messaging Interface
        VPIM
          Voice Profile for Internet Mail"

    REVISION      "200512120000Z"
    DESCRIPTION
```

```
        "Added these objects:
            ciscoUnityLicLanguagesMax,
            ciscoUnityLicTTSSessionsMax,
            ciscoUnityLicSubscribersMax,
            ciscoUnityLicUMSubscribersMax,
            ciscoUnityLicVMISubscribersMax,
            ciscoUnityLicVoicePortsMax,
            ciscoUnityLicBridgeSessionsMax,
            ciscoUnityLicAMISIsLicensed,
            ciscoUnityLicMaxMsgRecLenIsLic,
            ciscoUnityLicPoolingIsEnabled,
            ciscoUnityLicVPIMIsLicensed,
            ciscoUnityLicPrimaryServerIsLic,
            ciscoUnityLicSecondServerIsLic,
            ciscoUnityLicUtilSecondServer,
            ciscoUnityLicUtilSubs,
            ciscoUnityLicUtilVMISubs,
            ciscoUnityPortActivity,
            ciscoUnityPortObjectId"

    REVISION      "200401060000Z"
    DESCRIPTION
            "The initial version of this MIB module."
    ::= { ciscoMgmt 385 }


--
--      Textual Conventions
--
CiscoUnityIndex ::= TEXTUAL-CONVENTION
    STATUS      current
    DESCRIPTION
        "This textual convention is used as the Index into a table.
        A positive value is used to identify a unique entry in
        the table."
    SYNTAX      Unsigned32(1..2147483647)

CiscoUnityServerStatus ::= TEXTUAL-CONVENTION
    STATUS              current
```

```
    DESCRIPTION
        "This textual convention is used to indicate the current
        status of the local Unity server.
        stopped(1)        The main Cisco Unity process is stopped
        starting(2)       The main Cisco Unity process is starting
        running(3)        The main Cisco Unity process is in normal
                          operational mode
        stopping(4)       The main Cisco Unity process is shutting
                          down"
    SYNTAX      INTEGER {
        stopped(1),
        starting(2),
        running(3),
        stopping(4)
    }

ciscoUnityMIBNotifs  OBJECT IDENTIFIER ::= { ciscoUnityMIB 0 }
ciscoUnityMIBObjects OBJECT IDENTIFIER ::= { ciscoUnityMIB 1 }
ciscoUnityMIBConform OBJECT IDENTIFIER ::= { ciscoUnityMIB 2 }


ciscoUnityGeneralInfo       OBJECT IDENTIFIER
    ::= { ciscoUnityMIBObjects 1 }
ciscoUnityGlobalInfo        OBJECT IDENTIFIER
    ::= { ciscoUnityMIBObjects 2 }
ciscoUnityNotificationsInfo OBJECT IDENTIFIER
    ::= { ciscoUnityMIBObjects 3 }


--
--      THE UNITY TABLE
--
ciscoUnityTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF CiscoUnityEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The table containing information about all the Unity servers
        on the network visible to the local Unity server."
```

```
    ::= { ciscoUnityGeneralInfo 1 }

ciscoUnityEntry OBJECT-TYPE
    SYNTAX      CiscoUnityEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry (conceptual row) in the Unity table, containing
        the information about a Unity server."
    INDEX { ciscoUnityIndex }
    ::= { ciscoUnityTable 1 }

CiscoUnityEntry ::= SEQUENCE {
    ciscoUnityIndex             CiscoUnityIndex,
    ciscoUnityName              SnmpAdminString,
    ciscoUnityVersion           SnmpAdminString
}

ciscoUnityIndex OBJECT-TYPE
    SYNTAX      CiscoUnityIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An arbitrary integer, selected by the local Unity, which
        uniquely identifies a Unity server on the network."
    ::= { ciscoUnityEntry 1 }

ciscoUnityName OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The host name of the Unity server."
    ::= { ciscoUnityEntry 2 }

ciscoUnityVersion OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..128))
    MAX-ACCESS  read-only
```

```
    STATUS      current
    DESCRIPTION
        "The version number of the Unity server software."
    ::= { ciscoUnityEntry 3 }



--
--      THE PORT TABLE
--
ciscoUnityPortTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF CiscoUnityPortEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The table containing information about the configuration of
        ports on the local Unity server."
    ::= { ciscoUnityGeneralInfo 2 }

ciscoUnityPortEntry OBJECT-TYPE
    SYNTAX      CiscoUnityPortEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry (conceptual row) in the Port table, containing
        the information about the port configuration on the local
        Unity server."
    INDEX { ciscoUnityPortIndex }
    ::= { ciscoUnityPortTable 1 }

CiscoUnityPortEntry ::= SEQUENCE {
    ciscoUnityPortIndex             CiscoUnityIndex,
    ciscoUnityPortNumber            Unsigned32,
    ciscoUnityPortIntegration       SnmpAdminString,
    ciscoUnityPortExtension         SnmpAdminString,
    ciscoUnityPortEnabled           TruthValue,
    ciscoUnityPortAnswerCalls       TruthValue,
    ciscoUnityPortMessageNotif      TruthValue,
    ciscoUnityPortDialoutMWI        TruthValue,
```

```
    ciscoUnityPortAMISDelivery      TruthValue,
    ciscoUnityPortTRAPConnection    TruthValue,
    ciscoUnityPortActivity          SnmpAdminString,
    ciscoUnityPortObjectId          SnmpAdminString
}

ciscoUnityPortIndex OBJECT-TYPE
    SYNTAX      CiscoUnityIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An arbitrary integer, selected by the local Unity, which
        uniquely identifies a port on the local Unity server."
    ::= { ciscoUnityPortEntry 1 }

ciscoUnityPortNumber OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The Unity voice messaging port number."
    ::= { ciscoUnityPortEntry 2 }

ciscoUnityPortIntegration OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..128))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The phone system integration to which this port belongs.
        This could be Cisco CallManager or a traditional PBX."
    ::= { ciscoUnityPortEntry 3 }

ciscoUnityPortExtension OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..128))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The extension for the port as assigned on the phone system."
```

```
    ::= { ciscoUnityPortEntry 4 }

ciscoUnityPortEnabled OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This indicates whether this port is enabled on the local
        Unity server."
    ::= { ciscoUnityPortEntry 5 }

ciscoUnityPortAnswerCalls OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This indicates whether this port is designated to answer
        incoming calls."
    ::= { ciscoUnityPortEntry 6 }

ciscoUnityPortMessageNotif OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This indicates whether this port is designated for notifying
        subscribers of messages."
    ::= { ciscoUnityPortEntry 7 }

ciscoUnityPortDialoutMWI OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "This indicates whether this port is designated for turning
        MWIs on and off."
    ::= { ciscoUnityPortEntry 8 }
```

```
ciscoUnityPortAMISDelivery OBJECT-TYPE
   SYNTAX      TruthValue
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
      "This indicates whether this port is designated for making
      outbound AMIS calls to deliver voice messages from Unity
      subscribers to users on another voice messaging system."
   ::= { ciscoUnityPortEntry 9 }

ciscoUnityPortTRAPConnection OBJECT-TYPE
   SYNTAX      TruthValue
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
      "This indicates whether this port is designated for use by
      subscribers as a Telephone Recording And Playback (TRAP)
      device in Unity web applications and e-mail clients."
   ::= { ciscoUnityPortEntry 10 }

ciscoUnityPortActivity OBJECT-TYPE
   SYNTAX      SnmpAdminString
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
      "The port activity of the voice mail port.  This value
      is available to monitor port activity over time.  The specific
      value of this object is not as useful as monitoring for the
      changing of this value.  During normal operation this value
      should change several times during a call.  You can use this
      value to watch for abnormal operation that might indicate a
      problem with this voice mail port.  For example, if you
      monitor this value and do not detect a change in value for
      a long time (like 60 minutes) it could be an indication of
      a problem with that voice mail port.  There are other
      explanations as well so more investigation on the server
      should be conducted before taking any action with the voice
      mail port."
```

```
    ::= { ciscoUnityPortEntry 11 }

ciscoUnityPortObjectId  OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(0..38))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A globally unique, system-generated identifier for a port
        object.  The ObjectId (UUID or GUID) uniquely identifies the
        port for the lifetime of this port.  A UUID is 128 bits long,
        and can guarantee uniqueness across space and time."
    REFERENCE
        "RFC-4122 A Universally Unique IDentifier (UUID) URN
        Namespace."
    ::= { ciscoUnityPortEntry 12 }

--
--      All Scalar Objects
--
ciscoUnityServerState OBJECT-TYPE
    SYNTAX      CiscoUnityServerStatus
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The current state of the local Unity server."
    ::= { ciscoUnityGlobalInfo 1 }

ciscoUnityPorts OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of ports on the local Unity server."
    ::= { ciscoUnityGlobalInfo 2 }

ciscoUnityPortsActive OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
```

```
    STATUS      current
    DESCRIPTION
        "The total number of ports that are currently active with
        calls."
    ::= { ciscoUnityGlobalInfo 3 }

ciscoUnityPortsInbound OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of ports that are designated to answer
        incoming calls."
    ::= { ciscoUnityGlobalInfo 4 }

ciscoUnityPortsInboundActive OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of inbound ports that are currently active with
        calls."
    ::= { ciscoUnityGlobalInfo 5 }

ciscoUnityPortsOutbound OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of ports that are designated for outbound
        calls (includes message notification, MWI dialout and AMIS
        delivery)."
    ::= { ciscoUnityGlobalInfo 6 }

ciscoUnityPortsOutboundActive OBJECT-TYPE
    SYNTAX      Unsigned32 (0..255)
    MAX-ACCESS  read-only
    STATUS      current
```

```
    DESCRIPTION
        "The number of outbound ports that are currently active with
        calls."
    ::= { ciscoUnityGlobalInfo 7 }

--
--      License Information
--
ciscoUnityLicLanguagesMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..255)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The number of languages that Unity may use concurrently."
    ::= { ciscoUnityGlobalInfo 8 }

ciscoUnityLicTTSSessionsMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..255)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The maximum number of ports licensed for Text-to-Speech
        operations."
    ::= { ciscoUnityGlobalInfo 9 }

ciscoUnityLicSubscribersMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The maximum number of subscribers licensed."
    ::= { ciscoUnityGlobalInfo 10 }

ciscoUnityLicUMSubscribersMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
```

```
        "The maximum number of subscribers licensed for unified
        messaging."
    ::= { ciscoUnityGlobalInfo 11 }

ciscoUnityLicVMISubscribersMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The maximum number of subscribers licensed for Visual
        Messaging Interface (VMI)."
    ::= { ciscoUnityGlobalInfo 12 }

ciscoUnityLicVoicePortsMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..255)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The maximum number of voice ports licensed."
    ::= { ciscoUnityGlobalInfo 13 }

ciscoUnityLicBridgeSessionsMax OBJECT-TYPE
    SYNTAX Unsigned32 (0..255)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The maximum number of sessions licensed for Unity Bridge
        on the server."
    ::= { ciscoUnityGlobalInfo 14 }

ciscoUnityLicAMISIsLicensed OBJECT-TYPE
    SYNTAX TruthValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This indicates whether or not AMIS is licensed
        for this server."
    ::= { ciscoUnityGlobalInfo 15 }
```

```
ciscoUnityLicMaxMsgRecLenIsLic OBJECT-TYPE
   SYNTAX TruthValue
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION
      "This indicates whether or not this server is
      licensed to record voice mail messages of any length."
   ::= { ciscoUnityGlobalInfo 16 }

ciscoUnityLicPoolingIsEnabled OBJECT-TYPE
   SYNTAX TruthValue
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION
      "This indicates whether or not license pooling
      is enabled for this server."
   ::= { ciscoUnityGlobalInfo 17 }

ciscoUnityLicVPIMIsLicensed OBJECT-TYPE
   SYNTAX TruthValue
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION
      "This indicates whether or not VPIM is licensed
      for this server."
   ::= { ciscoUnityGlobalInfo 18 }

ciscoUnityLicPrimaryServerIsLic OBJECT-TYPE
   SYNTAX TruthValue
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION
      "This indicates whether or not this server
      is licensed to run as a primary/stand-alone Unity server."
   ::= { ciscoUnityGlobalInfo 19 }

ciscoUnityLicSecondServerIsLic OBJECT-TYPE
```

```
    SYNTAX TruthValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This indicates whether or not a secondary
        (failover) server is licensed."
    ::= { ciscoUnityGlobalInfo 20 }

ciscoUnityLicUtilSecondServer OBJECT-TYPE
    SYNTAX Unsigned32 (0..1)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This indicates the current number of licensed secondary
        servers."
    ::= { ciscoUnityGlobalInfo 21 }

ciscoUnityLicUtilSubs OBJECT-TYPE
    SYNTAX Unsigned32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This indicates the current number of licensed subscribers."
    ::= { ciscoUnityGlobalInfo 22 }

ciscoUnityLicUtilVMISubs OBJECT-TYPE
    SYNTAX Unsigned32 (0..2147483647)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This indicates the current number of subscribers that are
        licensed for VMI."
    ::= { ciscoUnityGlobalInfo 23 }



--
--      Notification Related Objects
--
```

```
ciscoUnityEventType  OBJECT-TYPE
   SYNTAX      INTEGER {
      error(1),
      warning(2),
      informational(3)
   }
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "The classification on the event severity.
      error(1)          Error events indicate significant
                        problems that the user should know about
      warning(2)        Warning events indicate problems that are
                        not immediately significant, but that may
                        indicate conditions that could cause
                        future problems.
      informational(3)  Information events indicate infrequent
                        but significant successful operations."
   ::= { ciscoUnityNotificationsInfo 1 }

ciscoUnityEventSource OBJECT-TYPE
   SYNTAX      SnmpAdminString
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "The software that logged the event, which can be either a
      program name, a component of the system, or a component of
      a large program."
   ::= { ciscoUnityNotificationsInfo 2 }

ciscoUnityEventCategory OBJECT-TYPE
   SYNTAX      SnmpAdminString
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "A classification of the event by the event source."
   ::= { ciscoUnityNotificationsInfo 3 }
```

```
ciscoUnityEventId OBJECT-TYPE
   SYNTAX      Unsigned32 (0..2147483647)
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "A number identifying the particular event type.  The EventID
      and EventSource can be used to troubleshoot system problems."
   ::= { ciscoUnityNotificationsInfo 4 }

ciscoUnityEventDate OBJECT-TYPE
   SYNTAX      DateAndTime
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "The date and time the event occurred."
   ::= { ciscoUnityNotificationsInfo 5 }

ciscoUnityEventUser OBJECT-TYPE
   SYNTAX      SnmpAdminString
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "The user name of the user on whose behalf the event occurred.
      This name is the client ID if the event was actually caused by
      a server process, or the primary ID if impersonation is not
      taking place."
   ::= { ciscoUnityNotificationsInfo 6 }

ciscoUnityEventComputer OBJECT-TYPE
   SYNTAX      SnmpAdminString
   MAX-ACCESS  accessible-for-notify
   STATUS      current
   DESCRIPTION
      "The name of the computer where the event occurred."
   ::= { ciscoUnityNotificationsInfo 7 }

ciscoUnityEventDescription OBJECT-TYPE
   SYNTAX      SnmpAdminString
```

```
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "The event description indicates what happened or the
        significance of the event."
    ::= { ciscoUnityNotificationsInfo 8 }

ciscoUnityEventEMSNotes OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "The contents of the notes field as entered into the EMS for
        this monitored event."
    ::= { ciscoUnityNotificationsInfo 9 }

--
--      Notifications
--

ciscoUnityMonitoredEvent NOTIFICATION-TYPE
    OBJECTS     {
        ciscoUnityEventType,
        ciscoUnityEventSource,
        ciscoUnityEventCategory,
        ciscoUnityEventId,
        ciscoUnityEventDate,
        ciscoUnityEventUser,
        ciscoUnityEventComputer,
        ciscoUnityEventDescription,
        ciscoUnityEventEMSNotes
    }
    STATUS      current
    DESCRIPTION
        "This Notification contains information from the Windows Event
        Log concerning an event that the Event Monitoring Service is
        configured to monitor."
    ::= { ciscoUnityMIBNotifs 1 }
```

```
--
--      MIB Conformance Statements
--

ciscoUnityMIBCompliances OBJECT IDENTIFIER
   ::= { ciscoUnityMIBConform 1 }
ciscoUnityMIBGroups      OBJECT IDENTIFIER
   ::= { ciscoUnityMIBConform 2 }

-- Compliance

ciscoUnityMIBCompliance MODULE-COMPLIANCE
   STATUS  deprecated -- replaced by ciscoUnityMIBComplianceRev1
   DESCRIPTION
      "The compliance statement for entities which implement
      the Cisco Unity MIB"
   MODULE         -- this module
   MANDATORY-GROUPS {
      ciscoUnityInfoGroup, ciscoUnityNotificationsInfoGroup,
      ciscoUnityNotificationsGroup
   }
   GROUP ciscoUnityPortInfoGroup
   DESCRIPTION
      "This group is mandatory for Cisco Unity servers that include
      voice processing components."
   ::= { ciscoUnityMIBCompliances 1 }

ciscoUnityMIBComplianceRev1 MODULE-COMPLIANCE
   STATUS  current
   DESCRIPTION
      "The compliance statement for entities which implement
      the Cisco Unity MIB"
   MODULE         -- this module
   MANDATORY-GROUPS {
      ciscoUnityInfoGroup, ciscoUnityNotificationsInfoGroup,
      ciscoUnityNotificationsGroup, ciscoUnityLicInfoGroup
   }
```

```
    GROUP   ciscoUnityPortInfoGroup
    DESCRIPTION
        "This group is mandatory for Cisco Unity servers that include
        voice processing components."

    GROUP   ciscoUnityPortInfoGroup2
    DESCRIPTION
        "This group is mandatory for Cisco Unity servers that include
        voice processing components."

    GROUP   ciscoUnityPortInfoGroup3
    DESCRIPTION
        "This group is mandatory only for Cisco Unity Connection
        servers that include voice processing components."
    ::= { ciscoUnityMIBCompliances 2 }

-- Units of Conformance
--
ciscoUnityInfoGroup OBJECT-GROUP
    OBJECTS {
        ciscoUnityName,
        ciscoUnityVersion,
        ciscoUnityServerState,
        ciscoUnityPorts,
        ciscoUnityPortsActive,
        ciscoUnityPortsInbound,
        ciscoUnityPortsInboundActive,
        ciscoUnityPortsOutbound,
        ciscoUnityPortsOutboundActive
    }
    STATUS current
    DESCRIPTION
        "A collection of objects which provide info about the local
        Unity server."
    ::= { ciscoUnityMIBGroups 1 }

ciscoUnityPortInfoGroup OBJECT-GROUP
```

```
    OBJECTS {
        ciscoUnityPortNumber,
        ciscoUnityPortIntegration,
        ciscoUnityPortExtension,
        ciscoUnityPortEnabled,
        ciscoUnityPortAnswerCalls,
        ciscoUnityPortMessageNotif,
        ciscoUnityPortDialoutMWI,
        ciscoUnityPortAMISDelivery,
        ciscoUnityPortTRAPConnection
    }
    STATUS current
    DESCRIPTION
        "A collection of objects which provide info about the port
        configuration of a Unity server."
    ::= { ciscoUnityMIBGroups 2 }

ciscoUnityNotificationsInfoGroup OBJECT-GROUP
    OBJECTS {
        ciscoUnityEventType,
        ciscoUnityEventSource,
        ciscoUnityEventCategory,
        ciscoUnityEventId,
        ciscoUnityEventDate,
        ciscoUnityEventUser,
        ciscoUnityEventComputer,
        ciscoUnityEventDescription,
        ciscoUnityEventEMSNotes
    }
    STATUS current
    DESCRIPTION
        "A collection of objects which provide info about all the
        Notifications generated by the Cisco Unity Agent."
    ::= { ciscoUnityMIBGroups 3 }

ciscoUnityNotificationsGroup NOTIFICATION-GROUP
    NOTIFICATIONS {
        ciscoUnityMonitoredEvent
```

```
    }
    STATUS  current
    DESCRIPTION
        "A collection of notifications that are generated by the
        CISCO UNITY MIB Agent."
    ::= { ciscoUnityMIBGroups 4 }

ciscoUnityLicInfoGroup OBJECT-GROUP
    OBJECTS {
        ciscoUnityLicLanguagesMax,
        ciscoUnityLicTTSSessionsMax,
        ciscoUnityLicSubscribersMax,
        ciscoUnityLicUMSubscribersMax,
        ciscoUnityLicVMISubscribersMax,
        ciscoUnityLicVoicePortsMax,
        ciscoUnityLicBridgeSessionsMax,
        ciscoUnityLicAMISIsLicensed,
        ciscoUnityLicMaxMsgRecLenIsLic,
        ciscoUnityLicPoolingIsEnabled,
        ciscoUnityLicVPIMIsLicensed,
        ciscoUnityLicPrimaryServerIsLic,
        ciscoUnityLicSecondServerIsLic,
        ciscoUnityLicUtilSecondServer,
        ciscoUnityLicUtilSubs,
        ciscoUnityLicUtilVMISubs
    }
    STATUS current
    DESCRIPTION
        "A collection of objects which provide info about the local
        Unity server."
    ::= { ciscoUnityMIBGroups 5 }

ciscoUnityPortInfoGroup2 OBJECT-GROUP
    OBJECTS {
        ciscoUnityPortActivity
    }
    STATUS current
    DESCRIPTION
```

```
        "A collection of objects which provide info about the port
        configuration of a Unity Connection server."
    ::= { ciscoUnityMIBGroups 6 }

ciscoUnityPortInfoGroup3 OBJECT-GROUP
    OBJECTS {
        ciscoUnityPortObjectId
    }
    STATUS current
    DESCRIPTION
        "A collection of objects which provide info about the port
        configuration of a Unity Connection server. This object group
        is only implemented on Cisco Unity Connection."
    ::= { ciscoUnityMIBGroups 7 }

END
```

Cisco Discovery Protocol (CDP) MIB

The Cisco Discovery Protocol (CDP) is a Cisco-proprietary network protocol used to broadcast device discovery information to routers and/or switches on the network. Cisco Unified Operations Manager can use this device discovery data to build a network topology and to identify devices within that topology. This means that a network administrator could then click on the device icon for a product node and quickly identify it.

CISCO-CDP-MIB

This section contain the text of the CISCO-CDP-MIB file.

```
--CISCO-CDP-MIB DEFINITIONS ::= BEGIN

IMPORTS
MODULE-IDENTITY, OBJECT-TYPE,
        Integer32
FROM SNMPv2-SMI
MODULE-COMPLIANCE, OBJECT-GROUP
FROM SNMPv2-CONF
        TruthValue, DisplayString, TimeStamp
FROM SNMPv2-TC
ciscoMgmt
```

```
FROM CISCO-SMI
CiscoNetworkProtocol, CiscoNetworkAddress, Unsigned32
        FROM CISCO-TC
    VlanIndex
        FROM CISCO-VTP-MIB
    ifIndex
        FROM IF-MIB
    ;

ciscoCdpMIB   MODULE-IDENTITY
LAST-UPDATED"200111230000Z"
ORGANIZATION"Cisco System Inc."
CONTACT-INFO
"Cisco Systems
Customer Service

Postal: 170 West Tasman Drive,
San Jose CA 95134-1706.
USA

  Tel: +1 800 553-NETS

E-mail: cs-snmp@cisco.com"
DESCRIPTION
"The MIB module for management of the Cisco Discovery
Protocol in Cisco devices."
    REVISION      "200111230000Z"
    DESCRIPTION
        "Added cdpInterfaceExtTable which contains the following
         objects:
         cdpInterfaceExtendedTrust,
         cdpInterfaceCosForUntrustedPort."
    REVISION      "200104230000Z"
    DESCRIPTION
        "Added the following objects:
         cdpGlobalDeviceIdFormatCpb,
         cdpGlobalDeviceIdFormat."
    REVISION      "200011220000Z"
```

```
        DESCRIPTION
            "Added the following objects:
             cdpCacheApplianceID,
             cdpCacheVlanID,
             cdpCachePowerConsumption,
             cdpCacheMTU,
             cdpCachePrimaryMgmtAddrType,
             cdpCachePrimaryMgmtAddr,
             cdpCacheSecondaryMgmtAddrType,
             cdpCacheSecondaryMgmtAddr,
             cdpCacheLastChange,
             cdpCachePhysLocation,
             cdpCacheSysName,
             cdpCacheSysObjectID,
             cdpGlobalLastChange"
        REVISION        "9812100000Z"
        DESCRIPTION
            "Added cdpGlobalDeviceId object."
        REVISION        "9809160000Z"
        DESCRIPTION
            "added these objects to cdpCacheTable:
            cdpCacheVTPMgmtDomain,
            cdpCacheNativeVLAN,
            cdpCacheDuplex.
            "
    REVISION        "9607080000Z"
    DESCRIPTION
    "Obsolete cdpInterfaceMessageInterval and newly
    define cdpGlobal object."
    REVISION        "9508150000Z"
    DESCRIPTION
    "Specify a correct (non-negative) range for several
    index objects."
    REVISION        "9507270000Z"
    DESCRIPTION
    "Correct range of cdpInterfaceMessageInterval."
    REVISION        "9501250000Z"
    DESCRIPTION
```

```
"Move from ciscoExperiment to ciscoMgmt oid subtree."
::= { ciscoMgmt 23 }


ciscoCdpMIBObjects OBJECT IDENTIFIER ::= { ciscoCdpMIB 1 }

cdpInterface   OBJECT IDENTIFIER ::= { ciscoCdpMIBObjects 1 }
cdpCache       OBJECT IDENTIFIER ::= { ciscoCdpMIBObjects 2 }
cdpGlobal      OBJECT IDENTIFIER ::= { ciscoCdpMIBObjects 3 }

--
-- The CDP Interface Group
--
cdpInterfaceTable OBJECT-TYPE
   SYNTAX     SEQUENCE OF CdpInterfaceEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
        "The (conceptual) table containing the status of CDP on
        the device's interfaces."
   ::= { cdpInterface 1 }

cdpInterfaceEntry OBJECT-TYPE
   SYNTAX     CdpInterfaceEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
        "An entry (conceptual row) in the cdpInterfaceTable,
        containing the status of CDP on an interface."
   INDEX      { cdpInterfaceIfIndex }
   ::= { cdpInterfaceTable 1 }

CdpInterfaceEntry ::= SEQUENCE {
   cdpInterfaceIfIndex          Integer32,
   cdpInterfaceEnable           TruthValue,
   cdpInterfaceMessageInterval   INTEGER,
   cdpInterfaceGroup            Integer32,
   cdpInterfacePort             Integer32
```

```
}

cdpInterfaceIfIndex OBJECT-TYPE
   SYNTAX     Integer32 (0..2147483647)
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
         "The ifIndex value of the local interface.

         For 802.3 Repeaters on which the repeater ports do not
         have ifIndex values assigned, this value is a unique
         value for the port, and greater than any ifIndex value
         supported by the repeater; in this case, the specific
         port is indicated by corresponding values of
         cdpInterfaceGroup and cdpInterfacePort, where these
         values correspond to the group number and port number
         values of RFC 1516."
   ::= { cdpInterfaceEntry 1 }

cdpInterfaceEnable OBJECT-TYPE
   SYNTAX     TruthValue
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
         "An indication of whether the Cisco Discovery Protocol
         is currently running on this interface.  This variable
         has no effect when CDP is disabled (cdpGlobalRun = FALSE)."
   ::= { cdpInterfaceEntry 2 }

cdpInterfaceMessageInterval OBJECT-TYPE
   SYNTAX     INTEGER (5..254)
   UNITS      "seconds"
   MAX-ACCESS read-write
   STATUS     obsolete   -- replaced by cdpGlobalMessageInterval
                         --  this object should be applied to the
                         --  whole system instead of per interface
   DESCRIPTION
         "The interval at which CDP messages are to be generated
```

```
        on this interface.  The default value is 60 seconds."
    ::= { cdpInterfaceEntry 3 }

cdpInterfaceGroup OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "This object is only relevant to interfaces which are
        repeater ports on 802.3 repeaters.  In this situation,
        it indicates the RFC1516 group number of the repeater
        port which corresponds to this interface."
    ::= { cdpInterfaceEntry 4 }

cdpInterfacePort OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "This object is only relevant to interfaces which are
        repeater ports on 802.3 repeaters.  In this situation,
        it indicates the RFC1516 port number of the repeater
        port which corresponds to this interface."
    ::= { cdpInterfaceEntry 5 }

cdpInterfaceExtTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF CdpInterfaceExtEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
       "This table contains the additional CDP configuration on
        the device's interfaces."
    ::= { cdpInterface 2 }

cdpInterfaceExtEntry OBJECT-TYPE
    SYNTAX     CdpInterfaceExtEntry
    MAX-ACCESS not-accessible
    STATUS     current
```

```
    DESCRIPTION
        "An entry in the cdpInterfaceExtTable contains the values
         configured for Extented Trust TLV and COS (Class of Service)
         for Untrusted Ports TLV on an interface which supports the
         sending of these TLVs."
    INDEX      { ifIndex }
    ::= { cdpInterfaceExtTable 1 }

CdpInterfaceExtEntry ::= SEQUENCE {
    cdpInterfaceExtendedTrust        INTEGER,
    cdpInterfaceCosForUntrustedPort  Unsigned32
}

cdpInterfaceExtendedTrust OBJECT-TYPE
    SYNTAX     INTEGER {
                   trusted(1),
                   noTrust(2)
               }
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
        "Indicates the value to be sent by Extended Trust TLV.

         If trusted(1) is configured, the value of Extended Trust TLV
         is one byte in length with its least significant bit equal to
         1 to indicate extended trust. All other bits are 0.

         If noTrust(2) is configured, the value of Extended Trust TLV
         is one byte in length with its least significant bit equal to
         0 to indicate no extended trust. All other bits are 0."
    ::= { cdpInterfaceExtEntry 1 }

cdpInterfaceCosForUntrustedPort OBJECT-TYPE
    SYNTAX     Unsigned32 (0..7)
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
        "Indicates the value to be sent by COS for Untrusted Ports TLV."
```

```
    ::= { cdpInterfaceExtEntry 2 }

--
-- The CDP Address Cache Group
--

cdpCacheTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF CdpCacheEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "The (conceptual) table containing the cached
            information obtained via receiving CDP messages."
    ::= { cdpCache 1 }

cdpCacheEntry OBJECT-TYPE
    SYNTAX     CdpCacheEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "An entry (conceptual row) in the cdpCacheTable,
            containing the information received via CDP on one
            interface from one device.  Entries appear when
            a CDP advertisement is received from a neighbor
            device.  Entries disappear when CDP is disabled
            on the interface, or globally."
    INDEX      { cdpCacheIfIndex, cdpCacheDeviceIndex }
    ::= { cdpCacheTable 1 }

CdpCacheEntry ::= SEQUENCE {
    cdpCacheIfIndex             Integer32,
    cdpCacheDeviceIndex         Integer32,
    cdpCacheAddressType         CiscoNetworkProtocol,
    cdpCacheAddress             CiscoNetworkAddress,
    cdpCacheVersion             DisplayString,
    cdpCacheDeviceId            DisplayString,
    cdpCacheDevicePort          DisplayString,
    cdpCachePlatform            DisplayString,
```

```
    cdpCacheCapabilities            OCTET STRING,
    cdpCacheVTPMgmtDomain           DisplayString,
    cdpCacheNativeVLAN              VlanIndex,
    cdpCacheDuplex                  INTEGER,
    cdpCacheApplianceID             Unsigned32,
    cdpCacheVlanID                  Unsigned32,
    cdpCachePowerConsumption        Unsigned32,
    cdpCacheMTU                     Unsigned32,
    cdpCacheSysName                 DisplayString,
    cdpCacheSysObjectID             OBJECT IDENTIFIER,
    cdpCachePrimaryMgmtAddrType     CiscoNetworkProtocol,
    cdpCachePrimaryMgmtAddr         CiscoNetworkAddress,
    cdpCacheSecondaryMgmtAddrType   CiscoNetworkProtocol,
    cdpCacheSecondaryMgmtAddr       CiscoNetworkAddress,
    cdpCachePhysLocation            DisplayString,
    cdpCacheLastChange              TimeStamp
}

cdpCacheIfIndex OBJECT-TYPE
    SYNTAX     Integer32 (0..2147483647)
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "Normally, the ifIndex value of the local interface.
            For 802.3 Repeaters for which the repeater ports do not
            have ifIndex values assigned, this value is a unique
            value for the port, and greater than any ifIndex value
            supported by the repeater; the specific port number in
            this case, is given by the corresponding value of
            cdpInterfacePort."
    ::= { cdpCacheEntry 1 }

cdpCacheDeviceIndex OBJECT-TYPE
    SYNTAX     Integer32 (0..2147483647)
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
            "A unique value for each device from which CDP messages
```

```
        are being received."
    ::= { cdpCacheEntry 2 }

cdpCacheAddressType OBJECT-TYPE
    SYNTAX     CiscoNetworkProtocol
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "An indication of the type of address contained in the
        corresponding instance of cdpCacheAddress."
    ::= { cdpCacheEntry 3 }

cdpCacheAddress OBJECT-TYPE
    SYNTAX     CiscoNetworkAddress
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
      "The (first) network-layer address of the device's
      SNMP-agent as reported in the Address TLV of the most recently
      received CDP message.  For example, if the corresponding
      instance of cacheAddressType had the value 'ip(1)', then
      this object would be an IP-address."
    ::= { cdpCacheEntry 4 }

cdpCacheVersion OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The Version string as reported in the most recent CDP
        message.  The zero-length string indicates no Version
        field (TLV) was reported in the most recent CDP
        message."
    ::= { cdpCacheEntry 5 }

cdpCacheDeviceId OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
```

```
    STATUS     current
    DESCRIPTION
        "The Device-ID string as reported in the most recent CDP
        message.  The zero-length string indicates no Device-ID
        field (TLV) was reported in the most recent CDP
        message."
    ::= { cdpCacheEntry 6 }

cdpCacheDevicePort OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The Port-ID string as reported in the most recent CDP
        message.  This will typically be the value of the ifName
        object (e.g., 'Ethernet0').  The zero-length string
        indicates no Port-ID field (TLV) was reported in the
        most recent CDP message."
    ::= { cdpCacheEntry 7 }

cdpCachePlatform OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The Device's Hardware Platform as reported in the most
        recent CDP message.  The zero-length string indicates
        that no Platform field (TLV) was reported in the most
        recent CDP message."
    ::= { cdpCacheEntry 8 }

cdpCacheCapabilities OBJECT-TYPE
    SYNTAX     OCTET STRING (SIZE (0..4))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The Device's Functional Capabilities as reported in the
        most recent CDP message.  For latest set of specific
```

```
            values, see the latest version of the CDP specification.
            The zero-length string indicates no Capabilities field
            (TLV) was reported in the most recent CDP message."
        REFERENCE "Cisco Discovery Protocol Specification, 10/19/94."

        ::= { cdpCacheEntry 9 }

    cdpCacheVTPMgmtDomain   OBJECT-TYPE
        SYNTAX    DisplayString (SIZE (0..32))
        MAX-ACCESS read-only
        STATUS    current
        DESCRIPTION
            "The VTP Management Domain for the remote device's interface,
            as reported in the most recently received CDP message.
            This object is not instantiated if no VTP Management Domain field
            (TLV) was reported in the most recently received CDP message."
        REFERENCE "managementDomainName in CISCO-VTP-MIB"
        ::= { cdpCacheEntry 10 }

    cdpCacheNativeVLAN OBJECT-TYPE
        SYNTAX    VlanIndex
        MAX-ACCESS read-only
        STATUS    current
        DESCRIPTION
            "The remote device's interface's native VLAN, as reported in the
            most recent CDP message.  The value 0 indicates
            no native VLAN field (TLV) was reported in the most
            recent CDP message."
        ::= { cdpCacheEntry 11 }

    cdpCacheDuplex OBJECT-TYPE
        SYNTAX    INTEGER {
                      unknown(1),
                      halfduplex(2),
                      fullduplex(3)
                  }
        MAX-ACCESS read-only
        STATUS    current
```

```
    DESCRIPTION
        "The remote device's interface's duplex mode, as reported in the
        most recent CDP message.  The value unknown(1) indicates
        no duplex mode field (TLV) was reported in the most
        recent CDP message."
    ::= { cdpCacheEntry 12 }

cdpCacheApplianceID OBJECT-TYPE
    SYNTAX     Unsigned32 (0..255)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The remote device's Appliance ID, as reported in the
        most recent CDP message. This object is not instantiated if
        no Appliance VLAN-ID field (TLV) was reported in the most
        recently received CDP message."
    ::= { cdpCacheEntry 13 }

cdpCacheVlanID OBJECT-TYPE
    SYNTAX     Unsigned32 (0..4095)
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The remote device's VoIP VLAN ID, as reported in the
        most recent CDP message. This object is not instantiated if
        no Appliance VLAN-ID field (TLV) was reported in the most
        recently received CDP message."
    ::= { cdpCacheEntry 14 }

cdpCachePowerConsumption OBJECT-TYPE
    SYNTAX     Unsigned32
    UNITS      "milliwatts"
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The amount of power consumed by remote device, as reported
        in the most recent CDP message. This object is not instantiated
        if no Power Consumption field (TLV) was reported in the most
```

```
        recently received CDP message."
    ::= { cdpCacheEntry 15 }

cdpCacheMTU OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Indicates the size of the largest datagram that can be
        sent/received by remote device, as reported in the most recent
        CDP message. This object is not instantiated if no MTU field
        (TLV) was reported in the most recently received CDP message."
    ::= { cdpCacheEntry 16 }

cdpCacheSysName OBJECT-TYPE
    SYNTAX     DisplayString (SIZE (0..255))
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Indicates the value of the remote device's sysName MIB object.
        By convention, it is the device's fully qualified domain name.
        This object is not instantiated if no sysName field (TLV) was
        reported in the most recently received CDP message."
    ::= { cdpCacheEntry 17 }

cdpCacheSysObjectID OBJECT-TYPE
    SYNTAX     OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Indicates the value of the remote device's sysObjectID MIB
        object. This object is not instantiated if no sysObjectID field
        (TLV) was reported in the most recently received CDP message."
    ::= { cdpCacheEntry 18 }

cdpCachePrimaryMgmtAddrType OBJECT-TYPE
    SYNTAX     CiscoNetworkProtocol
    MAX-ACCESS read-only
```

```
    STATUS      current
    DESCRIPTION
            "An indication of the type of address contained in the
            corresponding instance of cdpCachePrimaryMgmtAddress."
    ::= { cdpCacheEntry 19 }

cdpCachePrimaryMgmtAddr OBJECT-TYPE
    SYNTAX      CiscoNetworkAddress
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "This object indicates the (first) network layer address at
        which the device will accept SNMP messages as reported in the
        most recently received CDP message. If the corresponding
        instance of cdpCachePrimaryMgmtAddrType has the value 'ip(1)',
        then this object would be an IP-address. If the remote device
        is not currently manageable via any network protocol, this
        object has the special value of the IPv4 address 0.0.0.0.
        If the most recently received CDP message did not contain any
        primary address at which the device prefers to receive
        SNMP messages, then this object is not instanstiated."
    ::= { cdpCacheEntry 20 }

cdpCacheSecondaryMgmtAddrType OBJECT-TYPE
    SYNTAX      CiscoNetworkProtocol
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "An indication of the type of address contained in the
            corresponding instance of cdpCacheSecondaryMgmtAddress."
    ::= { cdpCacheEntry 21 }

cdpCacheSecondaryMgmtAddr OBJECT-TYPE
    SYNTAX      CiscoNetworkAddress
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "This object indicates the alternate network layer address
```

```
        (other than the one indicated by cdpCachePrimaryMgmtAddr) at
        which the device will accept SNMP messages as reported in the
        most recently received CDP message. If the corresponding
        instance of cdpCacheSecondaryMgmtAddrType has the value 'ip(1)',
        then this object would be an IP-address. If the most recently
        received CDP message did not contain such an alternate network
        layer address, then this object is not instanstiated."
    ::= { cdpCacheEntry 22 }

cdpCachePhysLocation OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Indicates the physical location, as reported by the most recent
        CDP message, of a connector which is on, or physically connected
        to, the remote device's interface over which the CDP packet is
        sent. This object is not instantiated if no Physical Location
        field (TLV) was reported by the most recently received CDP
        message."
    ::= { cdpCacheEntry 23 }

cdpCacheLastChange OBJECT-TYPE
    SYNTAX     TimeStamp
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Indicates the time when this cache entry was last changed.
        This object is initialised to the current time when the entry
        gets created and updated to the current time whenever the value
        of any (other) object instance in the corresponding row is
        modified."
    ::= { cdpCacheEntry 24 }


--
-- The CDP Global Group
--
```

```
cdpGlobalRun OBJECT-TYPE
   SYNTAX     TruthValue
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
        "An indication of whether the Cisco Discovery Protocol
        is currently running.  Entries in cdpCacheTable are
        deleted when CDP is disabled."
   DEFVAL     { true }
   ::= { cdpGlobal 1 }

cdpGlobalMessageInterval OBJECT-TYPE
   SYNTAX     INTEGER (5..254)
   UNITS      "seconds"
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
        "The interval at which CDP messages are to be generated.
        The default value is 60 seconds."
   DEFVAL     { 60 }
   ::= { cdpGlobal 2 }

cdpGlobalHoldTime OBJECT-TYPE
   SYNTAX     INTEGER (10..255)
   UNITS      "seconds"
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
        "The time for the receiving device holds CDP message.
        The default value is 180 seconds."
   DEFVAL     { 180 }
   ::= { cdpGlobal 3 }

cdpGlobalDeviceId OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
```

```
        "The device ID advertised by this device. The format of this
         device id is characterized by the value of
         cdpGlobalDeviceIdFormat object."
      ::= { cdpGlobal 4 }

cdpGlobalLastChange  OBJECT-TYPE
      SYNTAX     TimeStamp
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
         "Indicates the time when the cache table was last changed. It
         is the most recent time at which any row was last created,
         modified or deleted."
      ::= { cdpGlobal 5 }

cdpGlobalDeviceIdFormatCpb  OBJECT-TYPE
      SYNTAX     BITS {
                   serialNumber(0),
                   macAddress(1),
                   other (2)
                 }
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
         "Indicate the Device-Id format capability of the device.

         serialNumber(0) indicates that the device supports using
         serial number as the format for its DeviceId.

         macAddress(1) indicates that the device supports using
         layer 2 MAC address as the format for its DeviceId.

         other(2) indicates that the device supports using its
         platform specific format as the format for its DeviceId."
      ::= { cdpGlobal 6 }

cdpGlobalDeviceIdFormat  OBJECT-TYPE
      SYNTAX     INTEGER {
```

```
                serialNumber(1),
                macAddress(2),
                other(3)
              }
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
       "An indication of the format of Device-Id contained in the
       corresponding instance of cdpGlobalDeviceId. User can only
       specify the formats that the device is capable of as
       denoted in cdpGlobalDeviceIdFormatCpb object.

       serialNumber(1) indicates that the value of cdpGlobalDeviceId
       object is in the form of an ASCII string contain the device
       serial number.

       macAddress(2) indicates that the value of cdpGlobalDeviceId
       object is in the form of Layer 2 MAC address.

       other(3) indicates that the value of cdpGlobalDeviceId object
       is in the form of a platform specific ASCII string contain
       info that identifies the device. For example: ASCII string
       contains serialNumber appended/prepened with system name."
    ::= { cdpGlobal 7 }

-- conformance information

ciscoCdpMIBConformance
            OBJECT IDENTIFIER ::= { ciscoCdpMIB 2 }
ciscoCdpMIBCompliances
            OBJECT IDENTIFIER ::= { ciscoCdpMIBConformance 1 }
ciscoCdpMIBGroups
            OBJECT IDENTIFIER ::= { ciscoCdpMIBConformance 2 }


-- compliance statements

ciscoCdpMIBCompliance MODULE-COMPLIANCE
```

```
    STATUS  obsolete  -- superseded by ciscoCdpMIBComplianceV11R01
    DESCRIPTION
        "The compliance statement for the CDP MIB."
    MODULE  -- this module
      MANDATORY-GROUPS { ciscoCdpMIBGroup }

    ::= { ciscoCdpMIBCompliances 1 }

ciscoCdpMIBComplianceV11R01 MODULE-COMPLIANCE
    STATUS  obsolete -- superseded by ciscoCdpMIBComplianceV11R02
    DESCRIPTION
        "The compliance statement for the CDP MIB."
    MODULE  -- this module
      MANDATORY-GROUPS { ciscoCdpMIBGroupV11R01 }

    ::= { ciscoCdpMIBCompliances 2 }

ciscoCdpMIBComplianceV11R02 MODULE-COMPLIANCE
    STATUS  obsolete -- superseded by ciscoCdpMIBComplianceV12R02
    DESCRIPTION
        "The compliance statement for the CDP MIB."
    MODULE  -- this module
      MANDATORY-GROUPS { ciscoCdpMIBGroupV11R02 }

    ::= { ciscoCdpMIBCompliances 3 }

ciscoCdpMIBComplianceV12R02 MODULE-COMPLIANCE
    STATUS  current
    DESCRIPTION
        "The compliance statement for the CDP MIB."
    MODULE  -- this module
      MANDATORY-GROUPS { ciscoCdpMIBGroupV12R02 }

    ::= { ciscoCdpMIBCompliances 4 }

-- units of conformance

ciscoCdpMIBGroup  OBJECT-GROUP
```

```
    OBJECTS { cdpInterfaceEnable, cdpInterfaceMessageInterval,
              cdpCacheAddressType, cdpCacheAddress, cdpCacheVersion,
              cdpCacheDeviceId, cdpCacheDevicePort,
              cdpCacheCapabilities, cdpCachePlatform
            }
    STATUS  obsolete -- superseded by ciscoCdpMIBGroupV11R01
    DESCRIPTION
            "A collection of objects for use with the Cisco
            Discovery Protocol."
    ::= { ciscoCdpMIBGroups 1 }

ciscoCdpMIBGroupV11R01  OBJECT-GROUP
    OBJECTS { cdpInterfaceEnable, cdpInterfaceMessageInterval,
              cdpInterfaceGroup, cdpInterfacePort,
              cdpCacheAddressType, cdpCacheAddress, cdpCacheVersion,
              cdpCacheDeviceId, cdpCacheDevicePort,
              cdpCacheCapabilities, cdpCachePlatform
            }
    STATUS  obsolete -- superseded by ciscoCdpMIBGroupV11R02
    DESCRIPTION
            "A collection of objects for use with the Cisco
            Discovery Protocol."
    ::= { ciscoCdpMIBGroups 2 }

ciscoCdpMIBGroupV11R02  OBJECT-GROUP
    OBJECTS { cdpInterfaceEnable,
              cdpInterfaceGroup, cdpInterfacePort,
              cdpCacheAddressType, cdpCacheAddress, cdpCacheVersion,
              cdpCacheDeviceId, cdpCacheDevicePort,
              cdpCacheCapabilities, cdpCachePlatform,
              cdpGlobalRun, cdpGlobalMessageInterval, cdpGlobalHoldTime
            }
    STATUS  obsolete -- superseded by ciscoCdpMIBGroupV12R02
    DESCRIPTION
            "A collection of objects for use with the Cisco
            Discovery Protocol."
    ::= { ciscoCdpMIBGroups 3 }
```

```
ciscoCdpMIBGroupV12R02 OBJECT-GROUP
   OBJECTS { cdpInterfaceEnable,
             cdpInterfaceGroup, cdpInterfacePort,
             cdpCacheAddressType, cdpCacheAddress, cdpCacheVersion,
             cdpCacheDeviceId, cdpCacheDevicePort,
             cdpCacheCapabilities, cdpCachePlatform,
             cdpCacheVTPMgmtDomain, cdpCacheNativeVLAN, cdpCacheDuplex,
             cdpGlobalRun, cdpGlobalMessageInterval, cdpGlobalHoldTime,
             cdpGlobalDeviceId
           }
   STATUS  current
   DESCRIPTION
           "A collection of objects for use with the Cisco
           Discovery Protocol."
   ::= { ciscoCdpMIBGroups 5 }

ciscoCdpV2MIBGroup OBJECT-GROUP
   OBJECTS {
             cdpCacheApplianceID, cdpCacheVlanID,
             cdpCachePowerConsumption, cdpCacheMTU,
             cdpCacheSysName, cdpCacheSysObjectID,
             cdpCacheLastChange, cdpCachePhysLocation,
             cdpCachePrimaryMgmtAddrType,
             cdpCachePrimaryMgmtAddr,
             cdpCacheSecondaryMgmtAddrType,
             cdpCacheSecondaryMgmtAddr,
             cdpGlobalLastChange, cdpGlobalDeviceIdFormatCpb,
             cdpGlobalDeviceIdFormat
           }
   STATUS  current
   DESCRIPTION
           "A collection of objects for use with the Cisco
           Discovery Protocol version 2."
   ::= { ciscoCdpMIBGroups 6 }

ciscoCdpV2IfExtGroup OBJECT-GROUP
   OBJECTS {
             cdpInterfaceExtendedTrust,
```

```
            cdpInterfaceCosForUntrustedPort
        }
    STATUS  current
    DESCRIPTION
        "A collection of objects for use with the Cisco
        Discovery Protocol version 2 to configure the value
        for Extended Trust TLV and COS for Untrusted Port TLV."
    ::= { ciscoCdpMIBGroups 7 }
END
```

CHAPTER 4

Cisco Unity PerfMon and Alerts

This chapter contain perfmon and alert information for Cisco Unity and the Unity Object Model. It contains the following sections:

Unity PerfMon Counters

This list contains all the Perfmon counters that our CPR folks recommend monitoring when watching for a problems for versions of Unity 4.1(x) and later. This list includes not only Unity counters but also other counters such as total memory and SQL. These are the same counters used by the CUPID monitoring tool available on the 4.x tools page.

Active Server Pages\Sessions Current

Active Server Pages\Session Duration

Active Server Pages\Sessions Total

Cache\Data Map Hits %

Memory\Available Bytes

Memory\Commit Limit

Memory\Committed Bytes

Memory\Page Faults/sec

Memory\Pages Input/sec

Memory\Pages Output/sec

Memory\Pages/sec

Memory\Pool Nonpaged Bytes

Network Interface(*)\Bytes Received/sec

Network Interface(*)\Bytes Sent/sec

Paging File(_Total)\% Usage Peak

PhysicalDisk(*)\% Disk Time

PhysicalDisk(*)\Avg. Disk Bytes/Transfer
PhysicalDisk(*)\Avg. Disk Queue Length
PhysicalDisk(*)\Current Disk Queue Length
PhysicalDisk(*)\Disk Reads/sec
PhysicalDisk(*)\Disk Writes/sec
Process(AvCsGateway)\% Privileged Time
Process(AvCsGateway)\% Processor Time
Process(AvCsGateway)\Handle Count
Process(AvCsGateway)\Page Faults/sec
Process(AvCsGateway)\Private Bytes
Process(AvCsGateway)\Thread Count
Process(AvCsGateway)\Virtual Bytes
Process(AvCsGateway)\Working Set
Process(AvCsMgr)\% Privileged Time
Process(AvCsMgr)\% Processor Time
Process(AvCsMgr)\Handle Count
Process(AvCsMgr)\Page Faults/sec
Process(AvCsMgr)\Private Bytes
Process(AvCsMgr)\Thread Count
Process(AvCsMgr)\Virtual Bytes
Process(AvCsMgr)\Working Set
Process(AvCsTrayStatus)\% Privileged Time
Process(AvCsTrayStatus)\% Processor Time
Process(AvCsTrayStatus)\Handle Count
Process(AvCsTrayStatus)\Page Faults/sec
Process(AvCsTrayStatus)\Private Bytes
Process(AvCsTrayStatus)\Thread Count
Process(AvCsTrayStatus)\Virtual Bytes
Process(AvCsTrayStatus)\Working Set
Process(AvDcwService.ex)\% Privileged Time
Process(AvDcwService.ex)\% Processor Time
Process(AvDcwService.ex)\Handle Count
Process(AvDcwService.ex)\Page Faults/sec
Process(AvDcwService.ex)\Private Bytes
Process(AvDcwService.ex)\Thread Count
Process(AvDcwService.ex)\Virtual Bytes
Process(AvDcwService.ex)\Working Set
Process(AvDSAD)\% Privileged Time

Process(AvDSAD)\% Processor Time
Process(AvDSAD)\Handle Count
Process(AvDSAD)\Page Faults/sec
Process(AvDSAD)\Private Bytes
Process(AvDSAD)\Thread Count
Process(AvDSAD)\Virtual Bytes
Process(AvDSAD)\Working Set
Process(AvDSGlobalCatalog)\% Privileged Time
Process(AvDSGlobalCatalog)\% Processor Time
Process(AvDSGlobalCatalog)\Handle Count
Process(AvDSGlobalCatalog)\Page Faults/sec
Process(AvDSGlobalCatalog)\Private Bytes
Process(AvDSGlobalCatalog)\Thread Count
Process(AvDSGlobalCatalog)\Virtual Bytes
Process(AvDSGlobalCatalog)\Working Set
Process(AvGaenSvr)\% Privileged Time
Process(AvGaenSvr)\% Processor Time
Process(AvGaenSvr)\Handle Count
Process(AvGaenSvr)\Page Faults/sec
Process(AvGaenSvr)\Private Bytes
Process(AvGaenSvr)\Thread Count
Process(AvGaenSvr)\Virtual Bytes
Process(AvGaenSvr)\Working Set
Process(AvLic)\% Privileged Time
Process(AvLic)\% Processor Time
Process(AvLic)\Handle Count
Process(AvLic)\Page Faults/sec
Process(AvLic)\Private Bytes
Process(AvLic)\Thread Count
Process(AvLic)\Virtual Bytes
Process(AvLic)\Working Set
Process(AvMMProxySvr.ex)\% Privileged Time
Process(AvMMProxySvr.ex)\% Processor Time
Process(AvMMProxySvr.ex)\Handle Count
Process(AvMMProxySvr.ex)\Page Faults/sec
Process(AvMMProxySvr.ex)\Private Bytes
Process(AvMMProxySvr.ex)\Thread Count
Process(AvMMProxySvr.ex)\Virtual Bytes

Process(AvMMProxySvr.ex)\Working Set
Process(AvRepDirSvrSvc.)\% Privileged Time
Process(AvRepDirSvrSvc.)\% Processor Time
Process(AvRepDirSvrSvc.)\Handle Count
Process(AvRepDirSvrSvc.)\Page Faults/sec
Process(AvRepDirSvrSvc.)\Private Bytes
Process(AvRepDirSvrSvc.)\Thread Count
Process(AvRepDirSvrSvc.)\Virtual Bytes
Process(AvRepDirSvrSvc.)\Working Set
Process(AVSCAV~1)\% Privileged Time
Process(AVSCAV~1)\% Processor Time
Process(AVSCAV~1)\Handle Count
Process(AVSCAV~1)\Page Faults/sec
Process(AVSCAV~1)\Private Bytes
Process(AVSCAV~1)\Thread Count
Process(AVSCAV~1)\Virtual Bytes
Process(AVSCAV~1)\Working Set
Process(AvTtsSvr)\% Privileged Time
Process(AvTtsSvr)\% Processor Time
Process(AvTtsSvr)\Handle Count
Process(AvTtsSvr)\Page Faults/sec
Process(AvTtsSvr)\Private Bytes
Process(AvTtsSvr)\Thread Count
Process(AvTtsSvr)\Virtual Bytes
Process(AvTtsSvr)\Working Set
Process(CsBridgeConnect)\% Privileged Time
Process(CsBridgeConnect)\% Processor Time
Process(CsBridgeConnect)\Handle Count
Process(CsBridgeConnect)\Page Faults/sec
Process(CsBridgeConnect)\Private Bytes
Process(CsBridgeConnect)\Thread Count
Process(CsBridgeConnect)\Virtual Bytes
Process(CsBridgeConnect)\Working Set
Process(AvUMRSyncSvr.ex)\% Privileged Time
Process(AvUMRSyncSvr.ex)\% Processor Time
Process(AvUMRSyncSvr.ex)\Handle Count
Process(AvUMRSyncSvr.ex)\Page Faults/sec
Process(AvUMRSyncSvr.ex)\Private Bytes

Process(AvUMRSyncSvr.ex)\Thread Count
Process(AvUMRSyncSvr.ex)\Virtual Bytes
Process(AvUMRSyncSvr.ex)\Working Set
Process(inetinfo)\% Privileged Time
Process(inetinfo)\% Processor Time
Process(inetinfo)\Handle Count
Process(inetinfo)\Page Faults/sec
Process(inetinfo)\Private Bytes
Process(inetinfo)\Thread Count
Process(inetinfo)\Virtual Bytes
Process(inetinfo)\Working Set
Process(sqlservr)\% Privileged Time
Process(sqlservr)\% Processor Time
Process(sqlservr)\Handle Count
Process(sqlservr)\Page Faults/sec
Process(sqlservr)\Private Bytes
Process(sqlservr)\Thread Count
Process(sqlservr)\Virtual Bytes
Process(sqlservr)\Working Set
Process(STORE)\% Privileged Time
Process(STORE)\% Processor Time
Process(STORE)\Handle Count
Process(STORE)\Page Faults/sec
Process(STORE)\Private Bytes
Process(STORE)\Thread Count
Process(STORE)\Virtual Bytes
Process(STORE)\Working Set
SQLServer:Buffer Manager\Buffer cache hit ratio
SQLServer:Buffer Manager\Total pages
SQLServer:Databases(UnityDb)\Active Transactions
SQLServer:Databases(UnityDb)\Data File(s) Size (KB)
SQLServer:Databases(UnityDb)\Log Truncations
SQLServer:Databases(UnityDb)\Log Growths
SQLServer:Databases(UnityDb)\Log Shrinks
SQLServer:Databases(UnityDb)\Percent Log Used
SQLServer:Databases(UnityDb)\Transactions/sec
SQLServer:General Statistics\Logins/sec
SQLServer:General Statistics\Logouts/sec

SQLServer:General Statistics\User Connections
SQLServer:Latches\Average Latch Wait Time (ms)
SQLServer:Latches\Latch Waits/sec
SQLServer:Locks(_Total)\Average Wait Time (ms)
SQLServer:Locks(_Total)\Lock Requests/sec
SQLServer:Locks(_Total)\Lock Timeouts/sec
SQLServer:Locks(_Total)\Lock Waits/sec
SQLServer:Memory Manager\Connection Memory (KB)
SQLServer:Memory Manager\Total Server Memory (KB)
SQLServer:SQL Statistics\SQL Compilations/sec
System\Processes
System\Processor Queue Length
System\System Calls/sec
System\Threads
Telephony\Current Incoming Calls
Telephony\Current Outgoing Calls
Telephony\Incoming Calls/sec
Telephony\Outgoing Calls/sec
Unity(AvCsMgr)\Logon Silence
Unity(AvCsMgr)\Get Header Silence
Unity(AvCsMgr)\Play Message Silence
Unity(AvCsMgr)\Message Delete Silence
Unity(AvCsMgr)\Logons
Unity(AvCsMgr)\Opening Silence
Unity(AvCsMgr)\Record Silence
Unity(AvCsMgr)\Any Silence
Unity(AvCsMgr)\Message Store Lock
Web Service(_Total)\Connection Attempts/sec
Web Service(_Total)\Current Connections
Web Service(_Total)\Get Requests/sec
Web Service(_Total)\ISAPI Extension Requests/sec
Web Service(_Total)\Logon Attempts/sec
Web Service(_Total)\Maximum Connections
Web Service(_Total)\Maximum ISAPI Extension Requests
Web Service(_Total)\Maximum NonAnonymous Users
Web Service(_Total)\Post Requests/sec
Web Service(_Total)\Total ISAPI Extension Requests
Processor(_Total)\% Interrupt Time

Processor(_Total)\% Processor Time

Processor(_Total)\Interrupts/sec

Processor(_Total)\% Privileged Time

Processor(_Total)\% User Time

Process(AvDirChangeWrit)\% Privileged Time

Process(AvDirChangeWrit)\% Processor Time

Process(AvDirChangeWrit)\Handle Count

Process(AvDirChangeWrit)\Page Faults/sec

Process(AvDirChangeWrit)\Private Bytes

Process(AvDirChangeWrit)\Thread Count

Process(AvDirChangeWrit)\Virtual Bytes

Process(AvDirChangeWrit)\Working Set

TCP\Connection Failures

TCP\Connections Active

TCP\Connections Established

TCP\Connections Passive

TCP\Connections Reset

TCP\Segments Received/Sec

TCP\Segments Retransmitted/Sec

TCP\Segments Sent/sec

TCP\Segments/sec

Process(AvMsgStoreMonit)\% Privileged Time

Process(AvMsgStoreMonit)\% Processor Time

Process(AvMsgStoreMonit)\Handle Count

Process(AvMsgStoreMonit)\Page Faults/sec

Process(AvMsgStoreMonit)\Private Bytes

Process(AvMsgStoreMonit)\Thread Count

Process(AvMsgStoreMonit)\Virtual Bytes

Process(AvMsgStoreMonit)\Working Set

Unity(AvCsMgr)\TTS sessions

Unity(AvTtsSvr)\TTS sessions

PhysicalDisk(*)\% Idle Time

Process(AvDsDomino)\% Privileged Time

Process(AvDsDomino)\% Processor Time

Process(AvDsDomino)\Handle Count

Process(AvDsDomino)\Page Faults/sec

Process(AvDsDomino)\Private Bytes

Process(AvDsDomino)\Thread Count

Process(AvDsDomino)\Virtual Bytes

Process(AvDsDomino)\Working Set

Unity(AvCsGateway)\Authentication Count

Unity(AvCsGateway)\Authentication Time

Unity(AvCsGateway)\Average Authentication Time

Unity(AvCsGateway)\Maximum Authentication Time

Unity(AvCsGateway)\Concurrent Authentications

Process(java)\% Privileged Time

Process(java)\% Processor Time

Process(java)\Handle Count

Process(java)\Page Faults/sec

Process(java)\Private Bytes

Process(java)\Thread Count

Process(java)\Virtual Bytes

Process(java)\Working Set

Process(jk_nt_service.e)\% Privileged Time

Process(jk_nt_service.e)\% Processor Time

Process(jk_nt_service.e)\Handle Count

Process(jk_nt_service.e)\Page Faults/sec

Process(jk_nt_service.e)\Private Bytes

Process(jk_nt_service.e)\Thread Count

Process(jk_nt_service.e)\Virtual Bytes

Process(jk_nt_service.e)\Working Set

Unity Data Object Model

This section assumes you have a good understanding of the Unity web based system administration console (SA) and have a basic understanding of the objects you can create and use via the SA and how they work.

SQL tables are covered in broad terms, however to get down to the details on what each column in particular tables means and what it's valid values are, you'll want to get the latest version of the Cisco Unity Data Link Explorer (CUDLE) from the CiscoUnityTools.com web site. This tool has a full data dictionary built into it that's designed specifically to help folks wanting to drill down on these specifics quickly. For now let's walk through the objects collections and their relationships.

The following sections briefly describe each of the primary objects in the system and discuss some of their more important properties. A more complete list of each property, what it's value means and which columns in which SQL tables contain which values and what their legal ranges are can be seen in the CUDLE data explorer tool on CiscoUnityTools.com.

Applications

The applications object is not currently exposed in the Unity administration interface and isn't used for anything. Every object in the database, however, is associated with the one default application object in this collection by convention. The original idea behind this was to use it as a way to group objects together into separate application groups for providing tenant services type applications, however work for this was never done. Future versions of Unity may press this collection into service for tenanting applications.

Call Handlers

Call handlers are the essential building blocks for constructing audio text applications within Unity and are the primary mechanism for routing calls around within Unity as well as sending callers to internal and external phones. It is the call handler that plays custom recorded greetings to callers, responds to input from users and rings phones. It is, in short, the most fundamentally important object in the system.

Each subscriber is associated with a call handler which is referred to as the "primary call handler" for that user. If you look on the Unity SA and compare the call handler's administration pages with the subscribers pages you will notice that a subscriber is an almost perfect superset of a call handler. The only important limitation a primary call handler has that a stand alone call handler (referred to as an "application call handler") does not is that the subscriber administration interface only allows one transfer rule to be enabled (the alternate). An application call handler has 3 transfer rules (alternate, standard and off hours). This limitation was imposed early on to simplify the subscriber administration phone conversation. Subscribers can simply enable or disable transfers to their phones since dealing with lists of devices that can each be enabled/disabled via a phone interface can be somewhat daunting.

In figure 1 the call handlers that have a prefix of "ch_" are primary call handlers that are associated with a subscriber that has the alias that matches the remainder of the call handler alias. "ch_jsmith" is the primary call handler for the mail user with the alias "jsmith" for instance. The "cht_test template" call handler seen in figure 1 is a primary call handler for a subscriber template. Oddly, the two built in subscriber templates are associated with primary call handlers "ch_DefaltTemplate" and "ch_DefaultAdminTemplate". New templates you create will be prefaced with "cht_" however. The call handlers that have no prefix are application call handlers and they appear in the Unity administration interface under the call handlers page. The other call handlers listed in the collection do not appear on their own in the SA, their properties are exposed on the subscriber and the subscriber templates pages instead.

All call handlers also have 3 sub collections of objects for messaging rules (greetings), contact rules (transfers) and menu entries (user input rules). If you click on any of these properties in DPT a pop up window will come up on top of the main DPT interface to show the objects in the collection. Figure 2 shows the AVP_CONTACT_RULES collection.

AVP_CONTACT_RULES

Contact rules are the internal term for what the Unity SA calls transfer rules. There are three transfer rules on each call handler although, as noted above, primary call handlers associated with subscribers only use one: the alternate. The values for most of these properties are exposed on the "Call Transfer" page for call handlers, subscribers and subscriber template pages in the SA.

In short a contact rule is designed to optionally allow you to ring a phone when a call is passed to that call handler. By default, the first thing a call handler does when a call is passed into it is to process its contact rules and act on them. We'll cover this in much more detail in the Audio Text applications chapter later.

The three rules in the collection are "alternate", "standard" and "off hours". Which rule is evaluated depends on which are enabled in the SA and what time of day it is. If the alternate rule is active it is always the one evaluated, it over rides the other two in all cases. If the off hours rule is enabled in the SA and the schedule the call handler is associated with indicates it's after hours, it will be evaluated. If neither the alternate or the off hours rules trigger, the standard rule is evaluated. The standard rule can never be disabled in the SA and if it's disabled programmatically or by fiddling directly with SQL or DTP callers can end up being sent to the "failsafe" conversation ("I'm sorry, I can't talk to you now…") and hung up on. The standard rule is enabled to act as a "backstop" and prevent calls from falling through all three rules and into oblivion like that.

AVP_MENU_ENTRIES

The menu entries collection corresponds to the data visible on the "Caller Input" pages on the call handler and subscriber pages in the Unity SA. A total of 12 objects are in this collection that represent the programmable actions for the 0-9, * and # keys.

Each key can be setup to perform an action when pressed during the greeting for a call handler or subscriber. You can opt to hang up the call, take a message, send the caller to a different handler, route the call to the subscriber sign in conversation and the like. The values for the properties on the menu entry objects themselves can be looked up in the CUDLE tool on CiscoUnityTools.com as mentioned above.

The same set of menu entries is active and will take action for any of the 5 greetings that can play for a handler. This means you can't have different key actions active during the day vs. after hours which can present difficulties in some audio text applications. We'll cover ways to work around this in the Audio Text Applications chapter later.

AVP_MESSAGING_RULES

The messaging rules correspond to the greetings on the call handler and subscriber pages in the SA. You will see a total of 6 messaging rules in this collection, however the astute observer will note only 5 greetings are visible in the SA by default. The "Error" greeting is hidden by default in the SA but can be exposed by making a registry edit available in the Advanced Settings tool. You can also edit the Error greeting using the Bulk Edit Utility or the Audio Text Manager tool, both available in the Tools Depot or off the CiscoUnityTools.com web site. The Error greeting is a special greeting that dictates what happens when a user attempts to dial an extension that does not exist in the system during a greeting. This is hidden on the SA by default since we got so many calls and questions about how it worked in early versions and the need to customize it is reasonably rare in the field.

When a call is handed off to a call handler (either an application or a primary call handler associated with a subscriber) the transfer rules are processed first by default and then the call proceeds to the messaging rules if appropriate. You can change this to skip the transfer rules entirely by sending the caller to the greetings entry point in the call handler directly, which we'll discuss in the Audio Text applications chapter later. The schedule the handler is associated with and the source of the call and how it was routed to Unity determine how which greeting gets played. The greetings are processed in the following order:

- Alternate. If the alternate greeting is enabled, it will always play no matter what. It over rides all other greetings when active.
- Internal. If the internal greeting is enabled and the calling extension corresponds to a subscriber in the database, the internal greeting will play. Notice the distinction here: this greeting plays if the calling number corresponds to a subscriber's ID in the database, it has nothing to do with the actual origin of the call. This is one folks stumble on fairly frequently in the field.
- Busy. If the busy greeting is enabled and the forwarding reason is busy, this greeting will play.
- After Hours. If the after hours greeting is active and the schedule associated with the call handler indicates it's after hours, this greeting will play.
- Standard. If none of the other greetings kick in, the standard greeting will always play. It cannot be disabled in the SA and is always active.
- Error. The Error greeting is always active and enabled by default and is the one that gets played only when a caller enters an extension that cannot be found in the database while another greeting for the call handler or subscriber is playing. By default it tells the user "I'm sorry I did not hear that entry…" and routes the caller back to the opening greeting call handler created by setup. This can cause headaches for folks trying to do simple tenant services type applications since there may be multiple opening greeting handlers for multiple incoming numbers. Ways for dealing with basic tenant services type scenarios are covered in the Audio Text Applications chapter.

Aside from the sub collections noted above, a call handler has several important properties that link it to other objects in the system:

AVP_ADMINISTRATOR_OBJECT_ID

The Administrator Object ID corresponds to the "owner" property on the profile page of a call handler or an interview handler. Anything that has the "Object ID" tag on it indicates a unique identifier for an object in a collection.

By clicking on the "Find" button DPT will automatically take you to the appropriate object collection and show you that object. This is very handy for jumping around following various links off an object since the Object ID values themselves are not human readable in Unity 3.x. In versions of Unity 2.4.x and earlier the Object ID values were LDAP strings that referenced the container and alias of the object which made it reasonably easy to figure out which one it was pointing at. In 3.x and later they are GUID strings which would require tedious manual filtering to run down on your own. The fine folks in the DOH group added this functionality to make our lives a bit easier.

The Administrator Object ID can point to either a mail user or a distribution list and is found on call handlers, public distribution lists, interview handlers and name lookup handlers. While the Administrator Object ID value has been in the schema since day one it hasn't been used for anything of significance until the release of Unity 4.0. This value was originally intended to allow owners of objects to administer them over the phone and/or via the SA interface. For instance the ability to record the greeting on a call handler over the phone would be limited to the owern(s) of that handler. This was another one of those things that was originally slated to go in early on and the resources and time just never allowed us to get to it. However, starting in Unity 4.0 if you are the mail user or are a member of the public distribution list noted as the administrator (or "owner" in the SA) for a call handler you are allowed to record the greetings for that handler over the phone. This will make some folks happy in the field since the only other way to change the greeting over the phone is to use a "dummy" subscriber which, of course, uses a subscriber license. Currently that's the only use for this property, it is not possible to change properties on any other objects in the directory over the phone yet. It should be noted that this value has no impact whatsoever for user's access to objects via the web based SA interface.

AVP_RECIPIENT_OBJECT_ID

The Recipient Object ID points to a mail user or a public distribution list object that will get messages left for a call handler or interview handler. This is exposed on the SA on the "Messages" page for call handlers and subscribers.

In the case of a primary call handler associated with a subscriber, both the recipient and the administrator object Ids should both point to the mail user. This is one of the checks the dbWalker application makes while crawling the Unity database. If a primary call handler doesn't have both these properties pointing to a mail user and the mail user doesn't have the primary call handler object Id pointing back at the same handler, there's a problem that needs to be cleaned up. When a user creation action fails for whatever reason such a "cross linked" primary call handler can result. This means a primary call handler is left in the database that's pointing to the wrong subscriber for the owner and/or the recipient values.

The first thing Unity does when a call is sent to a call handler is to go fetch handles to both the recipient and the administrator objects. If the recipient value cannot be found, the caller will go to the "fail safe" conversation even if no greetings on that call handler are configured to take a message. This is where Unity sends calls it doesn't know what to do with. The caller hears "I'm sorry, I can't talk to you now, please try your call again later" and Unity logs one or more errors in the application event log to help in diagnosing the situation. It was decided to do this rather than risk the possibility of taking a message we couldn't deliver. If the owner value is not valid, in Unity 3.1(5) and earlier we used to also send the caller to the failsafe conversation, however in 4.0(1) and later this is no longer the case. An error is logged however the call proceeds as normal.

If, for instance, you have a customer feedback call handler setup to leave a message for Bill and you delete Bill as a subscriber (or delete Bill's mail account entirely) then that call handler has a "broken link" and will not work properly. The dbWalker utility is designed to help quickly run down and help administrators fix such broken links but it's important to understand that currently Unity does not dynamically "fix up" these types of links when objects are removed since it's very difficult to do this only the fly. For instance deciding what to do with broken owner and recipient links requires some sort of user feedback from the administrator.

AVP_LOCATION_OBJECT_ID

The Location Object ID points to the location this call handler is associated with. The location object is one of the items pushed into the directory and is replicated around to other Unity servers on the network so they can all "know" about one another automatically. All objects in the Unity database are associated with a location object, but only special types of subscribers can be associated with anything other than the default location object created by the Unity setup. Eventually this design will allow for full tenant applications in Unity by allowing entire groups of objects (call handlers, interview handlers, subscribers, directory handlers, COS objects etc…) to be associated with different locations on the same Unity server. Currently, however, there's only one "primary" location object per Unity box.

Additional location objects can be created for various networking schemes using AMIS, SMTP, the Cisco Unity Bridge and VPIM transport mechanisms to other Unity servers and/or other foreign voice mail systems. This is covered in detail in the Networking chapter later.

The Location object also serves an important roll for identifying other Unity servers on the network and finding out which Unity server a subscriber in the directory is associated with. When you install a Unity server onto your network the setup creates a uniquely named primary location object which is also replicated into the directory. All Unity servers in the directory, then, can identify one another via

information in these location objects and their subscribers also in the directory. The details of how this works are covered in the Unity Networking chapter later. An example showing how to find and attach to a remote Unity server's database to update the properties on a subscriber found in the directory will also be covered in the Administering Unity Programmatically chapter.

AVP_AFTER_MESSAGE_ACTION, AVP_AFTERMESSAGE_CONVERSATION, AVP_AFTER_MESSAGE_OBJECT_ID

These three properties work together to define where a caller is sent after leaving a message for a call handler or subscriber. This values for these are exposed on the "Messages" page for call handlers and subscribers under the "after message action" section in the SA.

You'll see a very similar trilogy of properties for action, conversation, destination object ID in the messaging rules and menu entry collections and other places as well. They all work in much the same way. The Action determines if a call is to be hung up or sent to another object. The conversation indicates which type of object and how it's to be used if the action is set to send the caller to another object. The destination object ID indicates which object will handle the call.

For instance if the after message action in the SA were setup to go to the greeting for the Operator Call handler, the AVP_AFTER_MESSAGE_ACTION would be 2, the AVP_AFTER_MESSAGE_CONVERSATION would be "phGreeting" and the AFTER_MESSAGE_OBJECT_ID would be the CallHandlerObjectID column for the operator call handler object in the call handlers collection. If you use the CUDLE tool to view these columns in their respective tables it will indicate which actions and conversation names are legal for that particular instance.

Tip The conversation "phGreeting" actually stands for "Phone Handler Greeting". Early in the development of Unity we were calling things phone handlers until it dawned on us that we weren't handling phones, we were handling calls. In the user interface things were renamed to call handlers but the underlying conversation code was already in place with the "ph" prefix and remains there today.

AVP_SCHEDULE_OBJECT_NAME

Schedules do not appear in the DOH Property Tester tool but call handlers have an AVP_SCHEDULE_OBJECT_NAME reference. This schedule defines what times of the day are considered "standard" and what times are considered "off hours" in 30 minute increments for each of the 7 days of the week. This is used to determine which greeting and transfer rules are triggered for the call handler when processing calls. In Unity 3.1(3) and later schedule definitions appear in the Schedule table in the UnityDB database in SQL. For versions of Unity prior to that they appear as separate keys in the registry under:

HKLM\Software\Active Voice\Schedule

The unique identifier for schedules in all versions is simply their name (i.e. "Day Shift") which means the schedule names themselves must be unique. You define different schedules in the SA on the "Schedules" page. There's no practical limit to the number of schedules you can create.

COS Objects

The Class of Service object is used to dictate which features a subscriber has access to over the phone, which administration functions they are allowed to perform via the web administration interface, if any, and which numbers they are allowed to dial. This is how licensing and administration restrictions are enforced and how dialing restrictions are imposed throughout the system. Every subscriber is associated with one and only one COS object which is set on the profile page for subscribers in the Unity SA. No other objects are associated with COS objects other than subscribers.

When you delete a class of service object via the SA interface, it does provide you with a mechanism for replacing references to the deleted COS object with a link to a valid one instead of leaving the link broken as is the case with most other objects you can remove via the SA. If you break this link by editing SQL or DPT directly you will cause many problems in the phone conversation and administration end of things. This is another check made by the dbWalker application covered in chapter XXX.

The COS object contains no sub collections however it does include links to 3 external objects that are of interest. The AVP_FAX_RESTRICTION_OBJECT_ID, AVP_OUTCALL_RESTRICTION_OBJECT_ID and AVP_XFER_RESTRICTION_OBJECT_ID all point to objects found under the restriction tables collection. The restriction tables limit which numbers can be entered into the fax delivery number, transfer number and out dial delivery numbers respectively. It's important to note that these restrictions are enforced at the time they are changed, not at the time they are dialed. This was done such that administrators could change delivery/transfer/fax numbers for subscribers using their own COS privileges. Once a number is accepted into a field, Unity will always dial it no questions asked.

There are a number of properties on the COS object that are not used by Unity. Most of the administration access properties (i.e. AVP_ACCESS_CALL_HANDLER_CUD) end in _CUD and _RO which was originally going to be to provide Read Only and Change/Update/Delete rights on a per access flag basis. This was never implemented and only the _CUD flags actually have any affect. You'll notice the COS objects created by setup may have the _RO flags set to "1" but any COS you create yourself will never change any of those values from "0" no matter what you do in the SA.

In Unity 3.1 a more granular SA access scheme was added to the COS object. Access to subscribers, public distribution lists and the COS administration pages in the SA were expanded to include read, edit, add and delete flags for each. This allows customers to have more specific limitations for these areas for helpdesk type applications where full administration access is not desired. You'll see these as sets of 4 flags for each of those objects (i.e. AVP_PDL_ADD_ACCESS, AVP_PDL_DELETE_ACCESS, AVP_PDL_MODIFY_ACCESS and AVP_PDL_READ_ACCESS). Earlier versions of Unity will not contain these properties.

Distribution Lists

Public Distribution Lists are unique in the Unity object model in that they are shared objects across all Unity servers in a directory and are also used by Exchange/Domino clients as well. No other objects in the scheme are shared across multiple systems like this. While a subscriber, for instance, is a mail user in Exchange/Domino it can only be a Unity subscriber in one Unity system. Attempting to import that same subscriber into another Unity server will fail. Public Distribution Lists in Unity are actually just regular old distribution lists in Exchange or Domino. When you "import" a distribution lists via the SA it simply writes a few properties through to the list object that indicate to Unity that it's a distribution list that we know about. The properties we write through to are covered in the Architecture Overview chapter. If you have configured Unity to be able to create new distribution lists via the SA, when you add a new list it will show up in Exchange 5.5/AD/Domino as a normal distribution list. There's nothing special or proprietary about these lists. In DPT you will only see distribution lists in this collection that have been "imported" into Unity using the SA.

A public distribution list is only available for addressing over the phone directly if it has an extension number and/or a voice name recorded on it. However, you can "import" a distribution list and make it the recipient of a call handler's messages, for instance, as a way to get messages to a group of users without having them address the distribution list by name or by ID. This is also a way to let unidentified callers leave messages for groups of users. It is not necessary to associate an extension with a distribution list or record a voice name for it to use it in this way.

Up through Unity 4.0 any properties changed on a distribution list via the SA simply writes through to the distribution list object itself in the directory. For instance if you have 2 Unity servers using the same public distribution list in AD and an administrator changes the extension number on it via the SA, this change will also happen on the other Unity server. This can cause problems in large organizations where some distribution lists will be shared across numerous Unity servers. Unity will eventually move to a model where each Unity server can dictate it's own values for extension and voice names on each distribution list instead of having to share those values.

The public distribution list object has one sub collection, AVP_MEMBERS, that contains a list of all the top level mail users and distribution lists contained in the public DL, it does not provide a full, flattened list of all members of all sub lists (and sub-sub lists and so on) contained in the distribution list. DOHPropTest does not show the new "scope" distribution lists that do have tables that include all members of all sub lists contained within them. This is discussed in more detail in the Audio Text Applications chapter later.

Note When Unity sends a message to a public distribution list, it simply addresses the message to the DL itself and lets Exchange or Domino "flatten" the list of addresses for us. We do not populate the "to" line with the members of the DL on our own. It's much faster and easier to let the messaging back end handle it for us.

Interview Handlers

Interview handlers are a special type of call handler that has very specific and limited functionality. You can configure an interview handler to ask a series of questions and get answers for each. It appends all the answers, separated by beeps, and sends the resulting message to the selected recipient which can be either a single subscriber or a public distribution list.

An interview handler has no greetings collection, no transfer capabilities no user input abilities or anything else. It's sole purpose in life is to ask callers a series of questions and collect the answers. These are typically used for support type applications where it's important that the caller leaves their name, number, version number, customer ID etc… when requesting a callback.

While the interviewers have fewer properties than call handlers, you'll notice most of the properties on interview handlers are also seen on call handlers. These properties serve the same purpose for interviewers as they do for call handlers. For instance the administrator and recipient object ID values and the after message action/conversation/destination object Id work the same as they do on call handlers. The one item in the property collection for interviewers that you wont find on call handlers, however, is the AVP_QUESTIONS sub collection. This is the list of questions you can add to the interviewer, up to 20 total. The questions are played in the order of the alias names (1-20) even though DOHPropTest doesn't display them in order – it sorts them alphabetically instead.

Locations

All objects in the Unity database are associated with a location object, but only special types of subscribers can be associated with anything other than the default location object created by the Unity setup. Eventually this design will allow for full tenant applications in Unity by allowing entire groups of objects (call handlers, interview handlers, subscribers, directory handlers, COS objects etc…) to be associated with different location on the same Unity server. Currently, however, there's only one "primary" location object per Unity box which is created by setup.

Additional location objects can be created for various networking schemes using AMIS, SMTP, the Cisco Unity Bridge and VPIM transport mechanisms to other Unity servers and/or other foreign voice mail systems. This is covered in detail in the Voice Mail Interoperability chapter later.

The Location object also serves an important roll for identifying other Unity servers on the network and finding out which Unity server a subscriber in the directory is associated with. When you install a Unity server onto your network the setup creates a uniquely named primary location object which is also replicated into the directory. All Unity servers in the directory, then, can identify one another via information in these location objects and their subscribers also in the directory. The details of how this works are covered in the Unity Networking chapter later. An example showing how to find and attach to a remote Unity server's database to update the properties on a subscriber found in the directory will also be covered in the Administering Unity Programmatically chapter.

Mail Users

The mail users collection includes all the subscribers on the local Unity server. As noted in the About Unified Messaging chapter earlier subscribers in Unity are simply regular mail users in Exchange or Domino that have some additional properties written through to them in the directory. The Directory and Messaging section covers the details of which properties are added to the directory schema for mail users.

There are several "types" of subscribers are contained in the mail users collection, the value for the AVP_SUBSCRIBER_TYPE will dictate which one you're looking at. The basic distinction is between a "full" or "regular" subscriber (type 1 for Exchange or type 3 for Domino) and an Internet Subscriber (type 2, 4, 6, 8). Refer to the CUDLE application's data dictionary for a full accounting of the subscriber type meanings. A regular subscriber is built on top of a full mail user in Exchange/Domino and has a mail store, can call in to check messages and all the things you'd expect a voice mail subscriber to be able to do. An Internet Subscriber is basically a routing mechanism for getting messages to a remote store via STMP, AMIS, OctelNet or VPIM transport mechanisms. These subscribers appear to callers as any other subscriber but they do not have a message store associated with them and, as such, you cannot call into Unity and sign in as an internet subscriber. The various subscriber types are covered in more detail in the Unity Networking chapter later.

The Mail User object contains four sub collections for notification rules and devices, MWI devices and private distribution lists.

AVP_NOTIFICATION_DEVICE

The notification device collection contains all the definitions for the devices that Unity can use to notify subscribers they have messages of a specific type and urgency. In Unity 4.0 these include 13 devices for phones, text pagers (email addresses) and DTMF pagers that a subscriber can configure to contact them based on a simple set of rules. Information about the devices shows up for subscribers in the SA on the "Message Notification" page.

The notification device object indicates the numbers to be dialed, what to dial (if anything) after connecting, how many times to retry if the number is RNA/Busy and the like. It works in conjuction with the AVP_NOTIFICATON_RULES collection below.

Note Earlier versions of Unity will have fewer notification objects. Additional notification devices were added in versions 2.3.6, 2.4.5 and 3.1. And, since I get asked this a lot, it's not possible to simply add new objects into this collection to increase the number of delivery devices in Unity. The notifier component that actually triggers the delivery action references these objects by alias and it wont simply use any object it finds in the collection.

AVP_NOTIFICATION_RULE

The notification rule collection contains the same 13 object aliases that the notification device collection does. The rules collection defines the types of messages the device should trigger on (i.e. urgent voice messages), what schedule the notification device should use (i.e. so you don't get paged at 3am), what to do if the device fails to connect, how many times to trigger the device and the like. The notification rules and the notification devices collection together contain all the settings you see on the "message notification" page for subscribers in the SA.

The reason the rule and the device information is stored separately is because originally the idea was that folks may want to have the same device, say a cell phone, that would have multiple sets of rules and schedules associated with it. This concept, however, proved to be more confusing that helpful and the idea was dropped but the work to merge the tables into one was never done.

AVP_NOTIFICAITON_MWI

The notification MWI collection contains information about activating message waiting indicators when a voice mail message arrives for a subscriber. Normally this collection contains one object named "MWI-1" which is the default MWI extension found on the "Messages" page for subscribers in the SA. However you can add up to 10 alternate MWI strings for a single subscriber which show up as additional items in this collection. When a voice message arrives for a subscriber all active MWI objects for that user will have their lamp activated. This can be useful for phone systems that support multiple message waiting lamps per phone or for doing simple shared mailbox scenarios.

It is not possible to turn MWIs on for other types of messages other than voice mail messages. The Notifier component that triggers the dialouts is hard coded to look specifically for messages in the inbox that are considered voice mails for triggering the MWI devices.

Unity can simultaneously support lighting lamps on multiple switches using multiple methods such as dialing DTMF codes on analog lines and sending serial packets across RS-232 connections. See the Integration chapter for more details on how that's configured.

AVP_PERSONAL_DLS

The personal distribution list collection contains up to 20 lists of objects for each subscriber that can be used to address messages to groups of subscribers or public distribution lists. Private lists are visible only to the subscriber they belong to, they cannot be shared across users and do not appear in the directory for Exchange or Domino. The personal DLs can contain references to subscribers or public distribution lists, but not other private distribution lists.

In Unity 3.x and later each private list can contain any number of mail users and public distribution list references. In versions prior to that each private had a limited number of entries allowed since they had to be crammed into existing fields in the Exchange 5.5 schema.

By default the personal DLs collection is empty for new subscribers. Users can create/edit personal distribution lists via the SA or the personal assistant web interface

The Mail User object has several important references to other external objects:

AVP_CALL_HANDLER_OBJECT_ID

This references an object in the call handler collection which is the "primary call handler" for this subscriber. As noted above, the call handler controls greetings rules, transfer rules and user input mappings that are exposed on the Subscriber pages in the SA.

AVP_COS_OBJECT_ID

This references which class of service object the subscriber is associated with. As noted above this dictates which features the user has access to, what numbers they are allowed to dial and what access, if any, they have to the web based administration consoles.

AVP_LOCATION_OBJECT_ID

This points to the location object the mail user is associated with. As noted above for normal subscribers this is not editable and points to the primary call handler created by the Unity setup. For internet subscribers this value can point to other location objects used to connect to remote voice mail systems using AMIS, OctelNet, VPIM or SMTP transport mechanisms. This is covered in the Unity Networking chapter later.

ALTERNATE EXTENSIONS

If you add additional extensions for a subscriber in the SA (Unity allows for up to 10 of these) and then go looking in DPT for where those extensions show up, you wont find them. All alternate extensions are written to the DTMFAccessID table in the UnityDB database in SQL. This table, in fact, will hold all information about all extensions for call handlers, subscribers, interviewers, public distribution lists, location objects and name lookup handlers in Unity database. DOHPropTest was not updated to pull out information from this table as a sub collection on the mail user and call handlers objects. We'll cover how to deal with alternate extensions and check for ID uniqueness using this table in the Administering Unity programmatically chapter later.

Mail User Templates

Mail users templates are very similar to mail users with just a few exceptions. The mail user templates are used to define a set of default data that is copied onto a new mail user when they are created or imported. By default there is a "Default Subscribers" and a "Default Administrators" mail user template created by the Unity setup. You can add additional templates from the Subscriber Template page in the

SA. As noted above, mail user templates are associated with a primary call handler just like mail users are. New templates created after setup get a call handler that has a prefix of "CHT_" for an alias as opposed to a prefix of "CH_" for mail users.

The primary differences between mail users templates and mail users is that templates do not have properties that must be defined on a "per user" basis which include extensions, alternate extensions and private distribution lists. The subscriber template includes a few properties that are not found on the normal mail user.

- The AVP_ADD_TO_DLS is a collection of public distribution lists that a new subscriber is added to when created or imported using this template. While it's a collection DPT does not pop up the usual collection dialog it does for most other collections in the system. Public Distribution Lists are simply referenced in a table at the bottom by their object id.
- AVP_DEFAULT_PW_XXX. There's a set of 4 properties that dictate default TUI (telephone user interface) and NT passwords. The phone password defaults are applied to all new or imported users. The NT passwords are used only when creating a new user in the directory, imported users do not use this value.

It's important to note that changes made to the subscriber templates only affect new users created after that change. This question comes up in the field from time to time. Changes made to a COS object affect those users associated with that COS immediately since mail users reference that COS object directly via the AVP_COS_OBJECT_ID property noted above. Values in the mail user template, however, are simply copied over when the user is created. There is no reference back to the template used to create a new user.

Name Lookup Handlers

The Name Lookup Handler corresponds to the "Directory Handler" page in the SA. This is the object that handles outside callers searching for subscribers by spelling their name in the case where they don't know the user's extension number. In versions prior to Unity 4.0 there was only one system wide name lookup handler per Unity install. In 4.0 and later there are multiple name lookup handlers possible and each can be configured to allow callers to search a different groups of users across the directory.

The name lookup handler in 4.0 has a couple of properties added to handle the new search scope options. The AVP_SEARCH_SCOPE and AVP_SEARCH_SCOPE_OBJECT_ID values work together to determine which subscribers the caller's spelled name search includes for this particular name lookup handler. You can refer to the CUDLE application to get details on what values in these two fields mean.

The Name Lookup handler has the usual links to external objects that include the location object ID, administrator object ID (not used for anything yet) and the application object ID (also not used) that show up on other objects in the system. However there are three important links to external objects in the name lookup handler you should notice:

AVP_EXIT_OBJECT_ID, AVP_EXIT_ACTION, AVP_EXIT_CONVERSATION

These three properties work together to determine what Unity does with a call when they exit the name lookup handler by pressing * (the exit key). Obviously if a user selects a subscriber by name, Unity sends the caller to that mail user. However if they "back out" by hitting * that's considered an exit action and the values for these three properties determine what Unity does with the call. You have the option of hanging up on the call, sending the caller to another object, sending them to a specific conversation such as the subscriber sign in conversation and the like. By default it sends the caller back to the "opening greeting" call handler created by the Unity setup. This value can be adjusted on the Caller Input page of the directory handler section of the SA.

There are two other "exit" action property sets for no selection and zero key events in this table. For the 4.0(1) release, however, the other two sets of actions are not used, all exit actions from the directory handler follow the instructions contained in the AVP_EXIT_XXX values. In future versions of Unity these may all be individually adjustable but at the time of this writing there are no concrete plans along those lines.

Note The details for what these property values are and what they mean is covered in the CUDLE application's data dictionary capabilities.

AVP_NO_SELECTION_OBJECT_ID, AVP_NO_SELECTION_ACTION, AVP_NO_SELECTION_CONVERSATION

These three properties work together to determine what Unity does with a call when they fail to spell the name of a user they are looking for and simply wait on the line. By default it sends the caller to the "say goodbye" call handler created by the Unity setup.

This value is not exposed on the directory handler page in the SA and it is not used. The no selection action follow the AVP_EXIT_xxx values above.

AVP_ZERO_OBJECT_ID, AVP_ZERO_ACTION, AVP_ZERO_CONVERSATION

These three properties work together to determine what Unity does with a call when they press 0 instead of spelling the name of a subscriber in the system. By default it sends the caller to the "operator" call handler created by the Unity setup.

This value is not exposed on the directory handler page in the SA and it is not used. The zero action follow the AVP_EXIT_xxx values above.

Primary Domain Accounts

This collection isn't really part of the Unity object model. It shows all the accounts Unity sees in the directory regardless of if they're tagged as a subscriber or not. This is useful for some troubleshooting scenarios.

Primary Domain Groups

Similar to the domain accounts collection, this is not part of the Unity object model. It shows all the groups defined in the directory. These objects are not referenced directly by objects in the Unity database, this collection is here for troubleshooting purposes.

PW Policies

The password policy object determines how phone passwords are treated by the Unity conversation. Administrators can dictate the passwords expire after a specified period of days, must be of a certain minimum length, how long users are locked out if they exceed the max number of PW retries etc… The values for this show up on the Account Policy page in the SA.

Currently there's only one password policy defined for the entire Unity server so this collection will never contain more than the one object created by the Unity setup. Down the road, however, there will likely be multiple policies to handle multiple tenant type configurations.

Restriction Tables

The restriction tables are referenced by the COS objects noted above. These objects are used to determine which numbers a subscriber is allowed to delivery messages to, transfer callers to or deliver faxes to. The Unity setup program creates three restriction tables by default: "DefaultFax", "DefaultOutdial" and "DefaultTransfer". All three of these restriction tables are all configured to limit long distance and international numbers in the North American dialing plan.

As noted above, restriction table rules are applied at the time the number is set, NOT when the number is dialed. The user making the change to the number in question uses the restriction tables associated with their Class of Service to determine if they can set a number to a particular string.

AVP_NUMBER_PATTERNS

This collection contains a series of "masks" which can be configured to allow or disallow a number to be entered into a particular field in the SA or over the phone. The strings consist of numbers, "?", and "*" symbols used to match any single digit or string of digits respectively. The masks are evaluated from the top of the collection down until a match for the number is found using the AVP_INDEX value to sort by. Only the first match encountered is evaluated. The "AVP_BLOCKED" property on the number pattern determines if a number match indicates it's allowed or if it's disallowed.

The restriction table can contain as many number patterns as you like, however as noted above the rules are evaluated from the top down until a match is made and then it stops.

Trusted Domains

The trusted domains collection is not part of the Unity object model, it's there to show all the domains visible to the Unity server for troubleshooting purposes.

Data Storage

With a basic understanding of how the objects in the Unity system relate to one another, the next obvious question to ask is where is all the configuration information actually stored on the system? There's four different areas configuration data is located for Unity, we'll cover the important items in each area.

SQL

The majority of the Unity related information is stored in the SQL database which runs locally on each Unity server. When conversations in the telephone interface or the SA/PCA pull up information about users, call handlers, interviewers etc., they are, for the most part, pulling this from the SQL database.

Other interfaces into the Unity database such as XLM and SOAP options or possibly web services mechanisms will come on line, however at the moment the only viable alternative to the DOH is going directly to the SQL database itself. Tools and utilities that ship with Unity which can be found on the http://www.CiscoUnityTools.com web site or the CCO utilities page.

A quick word about tables versus views. A "view" in SQL is a way to provide an interface into the database which allows you to change the actual structure of the tables in subsequent versions without breaking clients that have written queries against the view. In Unity 4.0 each of the tables will have a corresponding view, however initially many of them will simply be a straight mapping of columns from their corresponding table. As Unity moves forward the database will be reorganized to make it more efficient and fast. As data moves around between tables or tables are collapsed into each other and the like, any client writing directly to the tables themselves will have to update their applications each time a new release of Unity hits the streets. Not ideal. The views, however, can help prevent this by providing logic underneath to get the data from it's new location such that clients are none the wiser to the back end shenanigans going on under the covers. As a rule folks interested in writing applications to interact with the Unity database should use views when querying data and stored procedures when writing information into the database. When opening a connection to a database in SQL you access views in the same way you would a table. All the views for Unity tables are named with a 'vw_' prefix and for the 4.0 release correspond more or less one for one to the tables we'll cover here. More on this in the Administering Unity Programmatically chapter later.

Note In each database there are a number of tables that are all in lower case and start with "sys" such as "syscolumns" and "syscomments". These are present in every SQL database and are used by the system. They are not relevant to the inner workings of Unity and wont be covered here.

UnityDB

This section contains tables found int the Unity database.

AccountLockout

AccountLockout, AccountLockoutPolicy and AccountLogoinPolicy tables are used for the authentication model being used for the Unity 4.0 release and later. These dictate the desktop login authentication behavior for the clients. The values in these tables are exposed on the SA administration console on the Authentication page.

ADMonitorDirObjsList

This is a "scratch pad" database used by the active directory monitor. It's basically a list of GUIDs for objects in the directory it's interested in at any given time. See the Unity Architecture sectionr for more on what the directory monitor component does.

ADMonitorDistributionListMember

This table, along with the ADMonitorScopeDistributionList, ADMonitorScopeDistributionListMember, ADMonitorScopeDistributionListPendingChanges are all used as a "scratch pad" tables for the Active Directory monitor to use when pulling in distribution list members from the directory. There are three sets of such scratch pad tables for use when the system is connected to Active Directory (Exchange 2000), Exchange 5.5 or Domino.

None of these 12 "scratch pad" tables contains static information that's of interest, data in here eventually ends up in one of the four general distribution list tables listed below. There are two types of distribution lists stored in the database: System and Scope. A system distribution list is a public distribution list that has been imported into Unity and optionally assigned a voice name and/or extension number. This distribution list is updated in the directory with Unity specific information (see the schema extensions made to distribution lists in the directory section of this chapter). These distribution lists are used for addressing messages or as a recipient and/ore administrators for handlers in Unity. Scope lists, on the other hand, can be any public distribution list in the system, even those that have not been imported into Unity. These are used only for limiting user searches for the directory handlers starting in Unity 4.0. A public distribution list can be both a system distribution list and a scope distribution list and will appear in both tables.

The following four tables contain all the relevant information Unity knows about distribution lists in the system, regardless of which back end it's hooked up to.

- DistributionList. This table contains all the public distribution lists that have been imported into Unity. These are the distribution lists that can be addressed by subscribers over the phone by name or ID (if assigned) and can be set as the administrator or recipient of a handler. Yes, this table really should be called "SystemDList" but since this existed before the concept of a ScopeDList came into existence it's not.
- SystemDListMember. This table contains all the members of all the distribution lists in the DistributionList table. You find all the members of a specific distribution list by filtering this table's ParentObjectID column against the SystemDListObjectID column in the DistributionList table. You can also filter against the ParentAlias field but since strictly speaking the alias is not guaranteed to be unique and the ObjectID is, this is not a good practice. A member of this list can be a mail user OR a distribution list.
- ScopeDList. This table contains all the public distribution lists that are being used to as subscriber lists for directory handlers in Unity. These public distribution lists don't have voice names or extensions added to them and Unity doesn't update any extended attributes on it. The big difference here, though, is that the list of members for the scope distribution lists is "flattened". In other words if a distribution list contains other distribution lists Unity will recursively traverse the entire tree of lists and get all mail users in the list out. The SystemDList members, on the other hand, include only the top level members. In other words distribution lists included in the system distribution list are simply referenced in the SystemDListMember table and users in that list (and sub lists) are not explicitly referenced.
- ScopeDListMember. This table contains all the members of all the ScopeDList objects. Members of this table can only include mail users since, as noted above, Unity recursively traverses the list and all sub lists to pull out all users. This is done so the directory handler conversation can quickly do name lookups across all users in the list. To find all members of a specific scope distribution list you need to filter this table by it's ParentObjectID column with the ScopeDListObjectID column in the ScopeDList table.

Be aware that Unity is monitoring the directory for changes to the distribution lists setup as "scope" lists. If changes are made to a distribution list in the directory, it may take a while to replicate around the directory to where Unity "sees" the change and pulls it into the ScopeDListMember table.

Application

Currently the application database will only contain one row which doesn't contain any data of interest just yet. The "applications" collection in DOHPropTest simply shows this row. As noted in the object hierarchy section above, however, this table may get pressed into service when tenant services features are added to Unity down the road.

AuthenticationProvider

This table contains a list of all the authentication providers Unity supports for authenticating desktop users connecting to the SA and PCA web administration consoles. This concept is new to 4.0, earlier versions of Unity supported only NTLM authentication for clients. See section XXX for more on the new authentication schemes supported by 4.x.

CallHandler

All call handlers in the system are stored in this table. This includes primary call handlers for subscribers and templates. The CallHandlers collection in DOHPropTest is simply a list of the rows found in this table. Sub collections such as contact rules, messaging rules and user input rules are stored in separate tables (below). DOHPropTest does queries in those tables to populate those sub collections when you click on them. To find all the information for a specific call handler in this table you need to:

- Find the administrator (or "owner") of the call handler by following the AdministratorObjectID value which could reference a row in the Subscriber table or the DistributionList table. The AdministratorObjectIDType column will tell you which to look for.
- Find the message recipient for the call handler by following the RecipientObjectID value which could reference a row in the Subscriber table or the DistributionList table. The RecipientObjectIDType column will tell you which.
- Get it's transfer rules. To do this, filter the ContactRule table's ParentObjectID column against the CallHandler's CallHandlerObjectID column.
- Get its user input key mappings. To do this, tilter the MenuEntry table's ParentObjectID column against the CallHandlerObjectID column.
- Get the greetings for the handler. To do this, filter the MessagingRule table's ParentObjectID column against the CallHandlerObjectID.
- Get the schedule associated with the handler. To do this, find the Schedule row that has a DisplayName value that matches the call handlers ScheduleObjectName column.

ContactRule

All the contact rules (transfer rules on the SA) in the system are stored in this table. To get the contact rules for a particular call handler you filter the ParentObjectID column in this table against a match for the CallHandlerObjectID in the CallHandler table above. This should return 3 rows from the ContactRule corresponding to the standard, off hours and alternate transfer rules for the selected call handler. Remember, primary call handlers associated with subscribers only use the alternate contact rule.

COS

This stores all the Class of Service objects in the system. The COS objects determine what subscriber can and can't do in the web based administration console, what numbers they're allowed to dial, how long their greetings and messages can be and which features they are licensed for among other things. The restriction tables referenced in this table are stored in RestrictionTable table below.

Credential

This table contains a mapping of SIDs of users in the directory to ObjectID of subscribers on the local Unity server. When a user attempts to gain access to the SA or PCA web administration consoles, their SID is collected by IIS and passed through to Unity. This ID is then searched for in this table. If a match is found and the ObjectID corresponds to a subscriber that has a COS reference which allows access to the SA or PCA then they are allowed to proceed. If no entry is found or the subscriber the ID is associated with does not have rights to access the SA/PCA then they are denied entry.

Every subscriber on the system will have an entry in this table regardless of if they have SA/PCA rights or not. This table is also how administrators can gain SA access to multiple Unity servers without being a subscriber on more than one. The GrantUnityAccess tool found in the ToolsDepot can be used to add additional DirectoryID mappings to this table. This allows a single account to be associated with a subscriber that has SA access on multiple Unity servers. For large sites that use a central administration staff this is critical.

DefaultDListMembership

The next 4 tables that start with "default" are ones that should go away by the time Unity 4.0 is released, I list them here only because there's a chance this work wont get completed before final release. These are used by the DOH when creating new subscribers. These tables are not needed any longer and are residual stuff left over from older versions of Unity that needed them. All the default information for creating new subscribers should be pulled from the subscriber template selected by the user when creating/importing a subscriber in the system.

DistributionList

All the public distribution lists "imported" into Unity are listed in this table. Currently any public distribution list that's tagged with a voice name and/or an extension number by any Unity server on the network will end up showing up in this list. In later versions each Unity servers will only "see" the public distribution list specifically imported into it's database and every Unity server on the network can use it's own, unique extension number and voice name to identify the same, shared distribution list. For the time being, however, all public distribution lists share a single extension number and voice name among all Unity servers on the network.

The top level members of these distribution lists are stored in the SystemDList table. See the ADMonitorDistributionListMember comments above for more on the distribution list scheme in the database.

DTMFAccessID

The DTMFAccessId table contains all the extension numbers for all subscribers, call handlers, interview handlers, directory handlers, location objects and public distribution lists in the entire system. Remember that call handlers can have up to 10 alternate extension numbers, however if you look through the call handler table you will only find one DTMFAccessID column in the table. The alternate extensions for all call handlers (which, of course, also includes primary call handlers associated with subscribers) are stored in this table. To find all the extension numbers for a particular call handler you need to filter this table's ParentObjectID column against the CallHandlerObjectID from the CallHandler table. The resulting rows will be all the extension numbers for the selected call handler.

This table also contains subscriber and location extensions from other Unity servers on the network. The use of "subscriber" here instead of "call handler" is deliberate. As discussed in the Architecture Overview chapter, only subscriber and location extensions get pushed into the directory and replicated around such that other Unity servers can pull that information in. Call handlers and interview handlers are local objects only and are not replicated around the directory. The GlobalSubscriber and GlobalLocation tables (below) contain subscriber and location object information from other Unity servers in the directory. These 3 tables work together to provide the networking functionality discussed in the Digital Networking chapter later.

The DTFMAccessID tables serves a couple of very important rolls. When you go to add a new subscriber, handler etc… you need to make sure it does not conflict with an ID at the Dialing Domain scope (this term will be discussed in depth in the Digital Networking chapter later). This table provides an easy way of doing that with a single query instead of having to check several tables individually. When conversations are collecting an ID from the caller for addressing a message by ID or for doing an auto attendant ID lookup or the like, the lookups for a match have to be done against this single table instead of using a complex multi table search. In fact, the DTMFAccessID columns found in many of the tables will simply go away, all extension information will be stored only in this table. Since some client code is written to expect the DTMFAccessID columns to be there, however, the views for these tables will still have these columns represented and will pull the data out for you to provide backwards compatibility when this change is made. Yet another example of why you want to be using views instead of direct table queries.

GlobalLocation

The global location table contains all the location objects for all Unity servers on the network. This is important to the networking functionality for Unity and is discussed later in the Digital Networking chapter. The monitor "watches" the directory for any location objects for other Unity servers that might be on the network and then pulls its information into this table. The Architecture Overview chapter talks more about this process.

GlobalSubscriber

The global subscriber table contains a small amount of information about all subscribers on all Unity servers on the network. This is important to the networking functionality for Unity and is discussed later in the Digital Networking chapter. The monitor "watches" the directory for any mail users that have been "tagged" as a subscriber and pulls their information into this table. The global subscriber table contains much less information than you'll find in the subscriber table for local subscribers, of course, since pushing all the hundreds of pieces of data for each subscriber into the directory would be impractical

and unnecessary. This information is only used to locate and address messages to users on other Unity servers. The Architecture Overview chapter discusses which properties are written through to the directory and replicated around for subscribers.

This table is extremely useful for creating global management tools such as the Global Subscriber Manager found in the tools depot section. Unity does a lot of heavy lifting for you here to collect information about remote users, locations and servers that you can leverage to find which server a subscriber in the directory is associated with and, for instance, launch the SA page for that user directly. An example of how to do just this is covered in the Administering Unity Programmatically chapter later.

Holiday

The holiday table contains dates for days that schedules can indicate they are in holiday mode. When in holiday mode a schedule simply acts as "after hours" for the day regardless of what time it is or what the schedule normally considers "standard" vs. "off hours" times. Schedule can be set to respect or ignore holidays on a per schedule basis.

InterviewHandler

This table contains all the interview handlers on the local system. Like a call handler, an interview handler has an owner and recipient which are found by following the AdministratorObjectID and RecipientObjectID column values. These can be either a subscriber or a distribution list reference, you use the AdministratorObjectIDType and RecipientObjectIDType columns to determine which it is. The primary piece of data for an interview handler are the questions the caller hears when they hit this object which are stored in the InterviewQuestion table.

InterviewQuestion

All questions for all interview handlers on the system are stored in this table. To get all the questions for a specific interview handler you need to filter this table's "ParentObjectID" column against the "InterviewHandlerObjectID" column in the InterviewHandler table. The resulting list of questions will those for the interview handler you're interested in. The conversation is going to play these questions out in the order of the "Alias" field which is a number from 1 to 20 representing up to 20 questions allowed per interview handler. Note that it is possible to have non contiguous entries in here where you have, say, questions with alias 1, 2, 3, 6 and 8. The conversation is simply going to sort these in ascending order and play them out, it wont freak out if there's a "missing" number in there. The SA, on the other hand, will look a little odd since it'll include "blank" questions for the missing entries so in general this is to be avoided.

Location

The location table contains all the location objects created on the local Unity server. By default there is one "primary" location created by setup that must always be in this table since all objects created on the local server are associated with this location either directory or indirectly. Additional locations can be created for addressing messages to remote systems via AMIS, SMTP, OctelNet and VPIM. This is discussed in the Digital Networking chapter later.

All location objects created on the local Unity server are also "pushed" into the directory so they will replicate around. All other Unity servers using the same directory will "see" these location objects and pull their information into their local GlobalLocation table mentioned above. The directory section later in this chapter talks in more detail about how this process works.

MailboxStore

The MailboxSotre table contains an entry for each mail store the local Unity server cares about. Every Exchange or Domino mailstore that contains one or more Unity subscribers on the local box should have an entry in here. In the case of Exchange 2000 a single Exchange server can have several separate mail stores which can show up in this table. For Exchange 5.5 there is one store per Exchange server. This table is automatically updated to when you import or create a new subscriber, there should never be any need to edit any values in here or remove any rows for any reason.

MenuEntry

This table contains all the user input key mappings for each call handler in the system. Of course this includes primary call handlers associated with subscribers as well. There are 12 key mappings per call handler for all 12 of the keys on the phone pad (0-9, * and #). To get the menu entries for a particular call handler you need to filter the "ParentObjectID" column on this table against the "CallHandlerObjectID" column in the CallHandler table. The resulting list should contain 12 rows for the 12 key mapping actions for the call handler you're interested in. The "Alias" field contains the key map the row is for.

MessageRule

This table contains all the message rules (greetings) for all the call handlers in the system. Again, this includes primary call handlers for subscribers. Each call handler has a total of 6 messaging rules: Standard, Off Hours, Busy, Internal, Alternate and Error. To get the message rules for a particular call handler you need to filter this table on it's "ParentObjectID" column using the "CallHandlerObjectID" column in the CallHandler table.

MovedMailbox

When a mailbox gets moved in Exchange (either between Exchange servers or between mail stores in the case of Exchange 2000) Unity needs to take some special action. From the architecture section in the last chapter you'll remember that the notifier component stays logged into every subscriber's mailbox to monitor for message events that might require a MWI update or a notification dialout or the like. When a mailbox moves, the notifier is now no longer monitoring that guy. When the Exchange monitor notices that the mailbox of a subscriber has now changed, it throws a row in this table. The notifier will then know to log into this new mailbox and start monitoring it and should remove the row. Technically this table should not have any rows hanging out in it for very long.

NameLookupHandler

This contains all the directory handlers (also call “name lookup handlers”, also called “alpha directories”) in the local Unity server. Prior to Unity 4.0 there was only one hard coded directory containing all subscribers in the local system and optionally including subscribers on other Unity servers that were a part of the same dialing domain (this term is discussed in the Digital Networking chapter later). In Unity 4.0 and later there can be any number of directory handlers per Unity box with custom user lists.

NotificationDevice

Contains all the notification devices for all subscribers on the local Unity server. In Unity 4.0 there are a total of 13 notification devices available per subscriber. Earlier versions of Unity have fewer going back to the base set of 4 devices supported in the Unity 2.1 release a few years back. The notification device contains information about the delivery mechanism (i.e. pager, email, phone call) such as the phone number to dial, email address to send to and the like. To find all the notification devices for a particular subscriber you need to filter this table on it’s “ParentObjectID” matching the “SubscriberObjectID” value from the Subscriber table. The resulting list of rows should be all the notification devices for the subscriber in question. The alias column in this table indicates which notification device you’re working with.

Note that you cannot simply add new devices here and expect Unity to start triggering on them (this has been tried, believe me). The notifier is hard coded to deal with devices it knows about only, not just any deviced added to the collection.

NotificationRule

Contains all the notification rules for all the subscribers on the local Unity server. Each device (above) is associated with a rule which determines the schedule the device is active for, what types of messages will cause the device to trigger and the like. To find all the notification rules for a particular subscriber you need to filter this table on it’s “ParentObjectID” matching the “SubscriberObjectID” value from the Subscriber table. The resulting list of rows should be all the notification rules for the subscriber in question. The alias column in this table indicates which notification rule you’re working with.

Note that there is no direct link from the notification device to the notification rule or back. You have to get the list of devices and rules for a subscriber and match them up using the alias column in the rule and device tables. These tables will eventually merge into one (with separate views for each being maintained for backwards compatibility of course). The need to have the device information and the rule information in separate tables is questionable at best since there’s a strict one to one relationship here. The original idea was to have a single device, such as a cell phone, support multiple rules and schedules for it. However this concept proved to be more confusing than helpful and was scrapped.

NotificationMWI

Contains all the MWI devices for all subscribers on the local Unity server. Remember that each subscriber can have up to 10 MWI numbers associated with their mailbox for handling multiple line appearances or multiple phones. To find all the MWI devices for a particular subscriber you need to filter this table on it’s “ParentObjectID” value matching the SubscriberObjectID from the subscriber table. The alias column in this table indicates which MWI device you’re working with. By default all subscribers get one MWI device with an alias of “MWI-1”. Subsequent devices added via the SA get an

alias which is made up of the subscriber alias followed by the MWI extension itself. For instance if you added a 2nd MWI extension for the Example Administrator that dialed "4321" you'd see an MWI device in this table with an alias of "Eadmin4321". Since multiple MWI extensions must be unique for a single user, this ensures the aliases for all MWI devices associated with a single user are always unique.

OctelNetObjectQueue

This table is used as a "scratch pad" table for the service that talks to the Unity Bridge server. The Bridge is used to communicate with OctelNet nodes when interoperating with legacy voice mail systems. One of the features it supports is getting user information about subscribers on remote systems including their voice mail name. This allows for name confirmation and addressing by name or ID options for subscribers wanting to get messages to these remote users. This table temporarily holds that type of information coming from and going to the Unity Bridge. Nothing hangs out in this table for long, however. The Bridge functionality is discussed later in the Digital Networking chapter.

PersonalDList

This table contains all the personal distribution lists for all subscribers on the local Unity server. Each subscriber can have up to 20 private distribution list that contain subscribers and/or public distribution lists. By default this list is empty, lists are added only when subscribers configure them via the SA or PCA web interfaces or over the subscriber phone conversation. To find all the personal distribution lists associated with a particular subscriber you need to filter this table by it's ParentObjectID column matching the SubscirberObjectID column in the subscriber table.

PersonalDListMember

This table contains all the members of all the personal distribution lists for all subscribers on the local Unity server. A member can be either a mail user or a public distribution list. You cannot include a private list in another private list. Only subscribers (i.e. mail users that have been imported into Unity) and public distribution lists that have been imported into Unity can be included in this list. The reference to the subscriber or public distribution lists is done via an ObjectID (a Unity identifier) as opposed to a DirectoryID (an AD/Ex55 identifier) which means we need to be able to find these guys in the subscriber or Public Distribution lists tables in SQL to get messages to them.

To find all the members of a specific private distribution list you need to filter this table's ParentObjectID column on the PersonalDListObjectID column from the PersonalDList table. The resulting set of rows will be all the members of the personal distribution list in question.

In earlier versions of Unity the number of members in a private list used to be limited to 20 or 25 users per list. This restriction was lifted as Unity moved to SQL and you can now add as many members to a list as you like via the GUI administration applications. However the phone conversation wasn't updated to allow this so adding new users to a private list still limits the total members to be no more than 25 at the time of this writing. This is a know issue and will be fixed in later versions of Unity.

Note When a subscriber is deleted via the SA or the mail user is removed entirely from the directory (the monitor picks this up and deletes the subscriber on the fly) there are a series of SQL triggers that fire to make sure all the references to that subscriber are removed from the system. All their contact rules, messaging rules, primary call handler etc… are removed. All private distribution lists membership references to that guy should be removed as well, however in Unity 3.1(4) and earlier this is not the case.

This doesn't cause any big problems just be aware that you may see "stranded" users referenced in this table which causes the SA to show a blank entry in the list of recipients displayed for that private list.

PwPolicy

Currently this contains one row which stores the settings for the phone password rules that appear on the Account Policy page in the SA. This is in it's own table since at some point down the road there will be multiple phone password policies supported for multiple tenants on a single Unity server. Currently, however, the phone password policy applies system wide.

RestrictionTable

This table contains all the restriction table definitions referenced by all the Class of Service objects on the local Unity server. The COS object references three restriction tables to limit what numbers can be entered into transfer strings, delivery dialout strings and fax delivery numbers. There is no ParentObjectID type filter you do on this table since a single restriction table can be used by multiple COS objects. You query the RestrictionTableObjectID column in this table using the FaxRestrictionObjectID, OutcallRestrictionObjectID and XferRestrictionObjectID columns found in the COS table to find the restriction table you're looking for.

RestrictionPattern

This table contains all the dial restriction strings contained in all the restriction tables on the local Unity server. A restriction table can have any number of string patterns (i.e. "91??????????" or "011*") which can be marked for allow or disallow. To find all the number patterns for a specific restriction table you need to query on the ParentObjectID column in this table against the RestrictionTableObjectID column in the RestrictionTable table. The strings are evaluated in order of the Index column value in this table which goes from 0 to the number of patterns associated with the table. Order is important here since the first string pattern that matches the number will dictate if the number is considered blocked or allowed.

Rule

This table contains rows for all the call routing rules used to route incoming calls to the local Unity server. This is the info that shows up on both the direct and forwarded pages in the SA's Call Routing section.

Schedule

All the schedules defined in Unity that you associate with call handler and subscribers are stored in this table. For each day there is a binary string with a series of 0s and 1s that represent 48 half hour increments for the day indicating if the schedule is active or inactive for that 30 minute period. This is the same format for schedule information you'll find in the notification rules table above.

Note that although this table has the usual ScheduleObjectID in it, references to objects in this table are done by name, not by ObjectID which is normally the case. In the CallHandler table there is a column named "ScheduleObjectName" which contains the string found in the "Display Name" column in the Schedule table. This means, of course, that the display name must be unique for all schedules (which is

enforced by the SA). This is a little unusual and is a hold over of the fact that up until Unity 3.1(2) the schedule data was stored locally in the registry and referenced by it's key name. It was moved into SQL to help support the fail over feature and in an effort to minimize the impact of this move, the existing reference scheme was kept in tact.

Servers

In early versions of Unity this table contained all the Unity servers found in the directory but it's no longer used for that. The Servers table now actually only contains one row in it which stores information about the UnityDB table version. Whenever Unity updates any of the tables as part of an upgrade, this row gets updated to indicate what revolution the database is on. Based on this value the upgrade knows which SQL scripts to apply to a database to get it to the version being installed.

StreamsToDelete

When an object that contains a voice name or a set of greetings gets deleted, this table gets a list of WAV files (called "streams" internally) that need to be removed from the \Commserver\StreamFiles\ directory. For instance if a call handler is removed, it's voice name and up to 6 greeting files will be put into this table. A background process will eventually get around to checking this table and will remove those WAV files from the StreamFiles directory and then delete the rows from this table. There shouldn't ever be any static information in here that's of any value.

Subscriber

This table contains all the subscribers on the local Unity server. This table and the CallHandler table are the two most central tables in the database. This table references the objects in several other tables and several tables contain items that can be associated with the Subscriber row through their ParentObjectID column. To get all information about a particular subscriber on a row in this table you need to:

- Find their "Primary call handler" by filtering the CallHandler table against the CallHandlerObjectID column. The call handler, of course, has several collections with data in other tables (contact rules, message rules and menu entries for instance). See the CallHandler table above.
- Get the class of service for the user. To do this, filter the COS table against the COSObjectID column
- Get the notification devices the user has setup. To do this, filter the NotificationDevice and NotificationRule tables against their ParentObjectID columns matching the SubscriberObjectID column.
- Get the MWI device information for the user. To do this, filter the NotificationMWI table's ParentObjectID column against the SubscriberObjectID column.
- Get the private distribution lists the user may have set up. To do this, filter the PersonalDList table's ParentObjectID column against the SubscriberObjectID column. PersonalDLists, of course, have members stored in the PersonalDListMembers table.

In the Administering Unity Programmatically chapter we'll actually walk through an example that uses all these tables and the call handler related tables to glean most of the basic information about a subscriber and dumps it to a CSV file.

SubscriberPwDTMFHistory

This table contains encrypted versions of all the subscriber passwords for all subscribers on the local Unity server. These are kept around as long as is dictated on the password policy page in the SA. This is used to make sure subscribers don't reuse passwords more often than administrators want them to.

SubscriberTemplate

This table contains all the subscriber templates on the local Unity sever. A subscriber template contains much of the same data and references all the same tables as the a subscriber object does. These objects are used to copy in default settings for new subscribers being created on the system. See the Subscriber table above for more details on what to find in this table, they are very similar. One thing to be aware of is the primary call handler associated with a subscriber template is stored in the CallHandler table along with all other handlers. There is no special "CallHandlerTemplate" table and there's no property you can check for that indicates the handler is associated with a template as opposed to a subscriber. As such, determining which handlers are being used for templates in the system can be a bit tedious since you need to walk this table and create the list backwards.

SystemState

This table contains an encrypted binary blob that contains a bunch of licensing information for all Unity servers that can be "seen" by the local Unity box. This is unpacked and used by the licensing module running on Unity to determine which features/limits are licensed and how many are used up. This is particularly important for the new "pooled" licensing features added in Unity 4.0(1) that allow multiple Unity servers to "share" sets of licenses with one another.

UnitySetupParameters

This table contains a number of replacement variables that are used by the setup and upgrade routines and other scripts run on the Unity box.

VPIMObjectQueue

This is another "scratch pad" table that keeps a list of subscribers that have changed their display name, extension or voice name for the purposes of knowing when to include that information with messages to remote systems. No static information sticks around in this table for long. This table is only used if the VPIM feature is enabled.

Unity Reports

The UnityReports database contains tables for each of the reports you can run in the SA. These are used as scratch pad tables the report engine uses when generating the HTML or CSV outputs requested from the SA. After a report is complete, however, the information in the tables is kept around. This can be useful for generating your own reports from the residual data.

Two tables in here, however, are a little different than the others in that they actually contain data regardless of if you've run a report on the Unity server or not:

- The EVENTS table is populated by information in the event logs every 30 minutes by a background process in Unity. When you run the "Event Log" report it actually pulls data from this table, not from the event log itself.
- The SYSTEMAUDIT table gets information about adds/moves/changes made in the SA. This data is gleaned from the IIS logs every 30 minutes as well. When you run the "Administrative Access" report it pulls it's information from this table.

The reports engine in Unity is going to get a pretty significant overhaul down the road in the not-too-distant future. The same backwards compatibility requirements imposed on the UnityDB table do not apply here so expect these tables to change or be removed entirely without notice.

Registry

There's quite a lot of data stored in the Unity branch in the registry. A large portion of it is "scratch pad" information for various components that are not intended to be altered. Another large portion consists of keys that are not even in the registry by default that can be added for debugging purposes. Quite a bit, however, are items that folks troubleshooting problems in Unity or administrators wanting to affect system wide behavior of various components of Unity would be interested in. Unfortunately there is little documentation on many of these properties, either internally or externally. Some of the more common properties administrators and field techs might want to edit are exposed in the Advanced Settings Tool found in the tools depot on the desktop. The CAT group tries to keep the Advanced Settings Tool up to date with the most necessary registry edits and it's strongly recommended you stick to this list unless someone in TAC directs you otherwise. Changing properties in the registry without knowing the appropriate range of values and what behavior change to expect is not advised.

As Unity moves forward with post 4.0 releases, much of the configuration information currently stored in the registry will start to move into new tables in SQL. This will be necessary for being able to support proper clustering designs where this information needs to be shared across multiple servers acting in concert to provide more scalable solutions. It's also going to be necessary for being able to easily administer a Unity server programmatically since access to a remote registry is difficult at best. We'll cover an example on how to get at some of the more critical registry data from off box in the Programmatically Administering Unity chapter later. An effort is being made as part of this process to get decent documentation in place for each component's configuration data and to track changes to it. The careful observer of the current registry information will also note quite a bit of duplicated data in slightly different formats. This is another issue that should be improved as this configuration information moves into SQL. As a general rule at the moment, however, developers treat the registry as their own private spot to stick data necessary for debugging or for not hard coding some behavior that may be changing down the road or the like. Data in the registry should be treated with caution by administrators, field technicians and anyone wanting to write applications that uses this information. All that being said, we'll cover the high level keys here just to give the more eager students of Unity an idea of what type of information can be found down each path.

Note Much of the data in the registry is only read during the initial Unity startup sequence. As such, changing anything in the registry may require a restart of Unity before it's change takes effect. For keys exposed in the Advanced Settings Tool a note is made of which ones require a reboot, however it's always a good idea to assume a restart is necessary.

The following branches can be found in the registry under HKEY_LOCAL_MACHINE\Software\Active Voice\

Arbiter

The Arbiter component is responsible for handling inbound calls and allocating voice port resources to incoming and outbound calls. Most of the settings under this branch have to do with what each port on the local Unity server is capable of doing which is exposed in the SA on the Ports page. There's also settings under here which determine if Unity will allow an incoming call which originated from a Unity port in the first place (by default this is off) and how many ports Unity will allow to be busy before stopping dialouts to handle the incoming call load.

AvCsGateway

The CsGateway (Active Voice CommServer Gateway) is the component that talks to all external parties wanting to connect to the Unity services. Access to the DOH requires a component to first authenticate through the gateway service before being allowed to connect. By default there aren't any visible keys under this branch, "hidden" keys can be added under here for debug purposes.

AvCsMgr

The AvCsMgr (Active Voice CommServer Manager) is the primary service for Unity. Most processes currently run under this umbrella although individual items will be breaking out into their own services moving forward to provide better scalability and multiple box cluster configurations in the future. The primary piece of data under here is the list of which Unity components to load and in what order. This is the same information that technicians can edit using the MaestroTools.exe application in the \commserver directory. Removing services from the startup list or changing the order is not something folks should be attempting without instruction from TAC, of course.

AvCsNodeMgr

The Node Manager is used for Unity fail over configuration data. You'll see this key on all Unity 3.1(1) and later systems however there will only be data under here if the local Unity server setup as a primary failover server.

AvLogMgr

The Log Manager is used by all Unity components to write to data files (used for reports), diagnostic files and to the event log. Under this branch you'll find settings to control how much hard drive space diagnostics can take up, how low the hard drive space can get before diagnostics are turned off, which directory the log manager will use for data and diagnostic files and, most importantly, the diagnostic levels for each component. When you turn a diagnostic trace on in the SA or using the Unity Diagnostic Tool, that value is set in this branch.

AvRdbSvr

The Remote Database Server component provides a wrapper around the database lookup functions. Currently there's nothing under this branch, items can be added for debugging purposes.

AvRepDir

The Report Director is responsible for aging data and diagnostic files so they don't end up eating up the entire hard drive space. The settings in the SA on the Configuration page under the "Files Cleanup" section are stored in this branch.

AvRepMgr

The Report Manager, on the other hand, controls which reports are available to run and what directory their output ends up in. Each report offered in the system and subscriber pages in the SA is represented under this branch as a collection of modules (DLLs) that are used to extract the raw data and crunch that information into a report. You'll also find settings under this branch to control the "file scavenging" behavior of Unity. Every so often (every 30 minutes by default) the report manager will go scrape (or "scavenge") information out of the event logs and the IIS activity logs and pull them into tables in the UnityReports database.

AvSkinny

Unity uses the "Skinny" protocol to talk with the Call Manager system. Settings for that connection appear under here, most of which are not visible by default but can be added for debugging purposes.

AvWM

The Windows Messaging component (or as it's affectionately known in house, the "Wedgie Manager") is in charge of determining the up or down status of all external Exchange servers the local Unity server is interested in (i.e. all Exchange boxes that home one or more Unity subscribers). This is referred to internally as the "MAPI Traffic Cop" service. Since MAPI has such long timeouts (in excess of 15 minutes at times) when you attempt to log into a mailbox that is not available, Unity has to take preventative measures to ensure we don't try to do that. Every so often (15 seconds by default) Unity "pings" remote Exchange servers to ensure that the necessary message store and directory services are up and running on that box. If those services are not running properly or the server does not respond in the specified time (25 seconds by default) then Unity assumes that Exchange server is "off line". Subscribers homed on that server will not be allowed to log into their mailbox over the phone interface while their server is considered off line. You will see error messages in the application event log each time Unity marks an Exchange server as being off or back on line.

CallControl

The Call Control feature was never completed in Unity, however this registry branch still hangs out containing keys for the partially implemented functionality. It's unclear at this point if the call control functionality will ever be fully added into Unity or not, for now there's nothing of interest in this branch.

CDE

The Conversation Development Environment is the engine on which all telephone conversations are developed in Unity. The idea was to allow the development of several different types of conversations using the same engine such that partners could design and implement their own phone conversations. The dream of having separate subscriber mailbox conversations, for instance, has not yet been fully realized although a few partners reselling Unity for Active Voice did develop their own custom conversations on their own. The most interesting piece of information under here from an administration standpoint is the soft key configuration file. This file determines if the subscriber message retrieval conversation uses the default Unity menu options or the optional conversation menus (which sound more like what an Octel Aria user might be used to). This key is exposed in the Advanced Settings tool.

CommServer

This branch holds some general information about the local Unity configuration setup such as the server name, the Unity install path and the path to the localized resource files and event log files. You'll notice some of this data (in particular the server name and install path) is also available in several other branches below.

Commserver Setup

This branch holds a number of general configuration settings for the local Unity install. Information about the Dialogic and NMS drivers installed (if any), the Unity installation path (in a couple of places in this branch), the version of Unity installed, which type of mailstore Unity is connected to, the default languages to use for the SA, TTS and phone conversation and a number of other items can be found under here.

Conversations

The conversation branch has several settings that can change the phone conversation behavior system wide, some of which are visible in the branch by default and some which need to be added for the change to take effect. For instance the option to characterize return receipts as voice mail messages or not is under this branch. The need for that option will go away in 4.0 since receipts of all types will have their own stack in the phone conversation. The options in this branch are exposed in the Advanced Settings tool.

DalDB

The Directory Access Layer branch indicates the program ID of the monitor the local Unity server is using to keep the directory and the local SQL database in synch. There is a separate monitor for Exchange 5.5, Exchange 2000 (Active Directory) and Domino. Depending on what back end your Unity server is connected to, the monitor for that back end will be noted in this branch.

Directory Connectors

The directory connectors branch holds quite a bit of very important information for the local Unity server. The sub branches under here will look somewhat different depending on if you're connected to AD (Exchange 2000), Exchange 5.5 or the Domino directories but the basic idea is the same. All the domains the local Unity server is monitoring, the root containers used for searches in each of those domains and which container objects are created in by default is all stored under this branch. Information about the Unity Bridge Server the local server is connected to (if you're using that feature) is also found under this branch since the Bridge is considered a connection of sorts to the remote OctelNet directory.

One important piece of information found under this branch if you are using AD (i.e. the Unity server is connected to an Exchange 2000 server) is the global catalog server reference. This is found under \Directory Connectors\DirSynchGlobalCatalog\1.00\Directory\DefaultGlobalCatalogServer. It's stored as it's fully qualified domain name (i.e. "testbox.mydomain.com"). If Unity is unable to contact the global catalog server using that name for whatever reason, it will not run properly. If a global catalog server is taken off line and replaced with another, for instance, it may be necessary to change this reference and restart Unity. The Unity 4.0 development team is working on a mechanism to dynamically discover and replace this connection on the fly but at the time of this writing it's not known when this functionality will make it into a production line of Unity.

DOH

The Data Object Hierarchy (DOH) as noted in the architecture chapter is actually a collection of three layers: The Directory Access Layer (DAL), the Message Access Layer (MAL) and the Security Access Layer (SAL). This branch has some general defaults the DOH component uses when initializing it's components. The only items under this branch that field technicians might end up editing is the option to not create domain accounts when subscribers are created in the SA (i.e. in Exchange 5.5 only create a mail account, don't create an NT account with it). This option is exposed in the Advanced Settings tool.

DPT

The DOHPropTest branch will only be visible if you've run the DOHPropTest.exe application at least once. By default it has no data under it but keys can be added for debugging purposes.

ExchangeMonitor

If the Unity server is connected to Exchange 5.5 or 2000 the keys under here will determine how often Unity checks the "companion" Exchange server (i.e. the Exchange server selected during the configuration setup) to see if it's message store is up and running properly. If the companion server is determined to be off line or unresponsive then Unity will go into "UMR mode" and start storing messages locally until the mail store comes back up.

FailureConv

Whenever the Unity conversation runs into an error it can't resolve (i.e. it loads a call handler but the message recipient for that call handler is no longer in the database) it sends to caller to the "fail safe" conversation. The caller is told "I'm sorry, I can't talk to you now. Please try your call again later" and an error is logged to the application event log indicating why the call was routed to the failsafe conversation. The FailureConv branch indicates the path and the WAV file to play for the failsafe conversation. The WAV file to play for the shutdown conversation is also listed under here. The Shutdown conversation is played to callers currently in the Unity conversation when an administrator selects to shut Unity down through the status monitor web page and opts not to wait for the conversations to finish up on their own first.

GAEN

The General Audio Error Notification (GAEN) utility monitors the event log for errors and then notifies administrators by voice mail, email or pager access that there is a problem with the local system. This utility is known more commonly by the name "Event Notification Utility" or ENU. This registry branch contains some default configuration data for the ENU utility which includes the path to the directory containing the Access database with the list of errors it's monitoring and notification method(s) to use when each error is seen. This information is configurable using the Event Notification Utility administration interface which is available in the Tools Depot or the Unity program group in the start menu.

Initalization

This branch used to contain some of the default system wide settings now found in the Commserver Setup and Commserver branches above. While it's not used for anything the branch continues to live on in the registry.

Keypad Mapping

When callers spell the name of a subscriber or a distribution list they are looking for over the phone, the mapping of letters to numbers on the phone keypad is called a "key pad map". There are three basic keypad mappings used in the world today, although there are a few proprietary phone systems that use their own and some countries (i.e. Japan) that don't use any. This branch contains information about the 3 keypad mappings supported by Unity and which one the local Unity server is using. Since Unity stores everyone's first and last "DTMF names" using all three keypad mappings it's possible for users on other Unity servers to spell the name of a user on another Unity server using a different keypad map, however Unity will make the translation for them so they wont have to account for it. In versions of Unity prior to 3.0 this was not possible and callers had to adjust the keys they used to spell the user's name by paying attention to the message addressing prompts played to them over the phone with predictably high failure rates.

MALEx/MALDom

As noted above the Message Access Layer (MAL) is a DOH component and it's used to talk to the message store the Unity server is configured to talk to. In the case of the MALEx branch that means an Exchange server is configured as the Unity "companion" server. This branch contains information about

that server and the local message profile used to log into the Unity messaging system account on that server. In addition to the message store connectivity information in here there is also configuration information for how Unity identifies fax messages in the inbox of subscribers. Since Unity supports numerous 3rd party fax servers, the message class(es) and file extensions supported by the fax server software being used by a customer can vary quite a bit. This information can be adjusted either in the Third Party Fax Configuration tool found in the Tools Depot interface.

MIU

The Media Interface Unit (MIU) component is responsible for all communication to the switch(es) Unity is configured to work with. There are numerous keys under the initalization branch in here that are exposed on the SA pages and in the UTIM (Unity Telephone Interface Manager) application. This branch also contains information about the Automatic Gain Control (AGC) settings for the local system as well as the recording codec being used for new recordings in Unity.

MsgStoreMonitor

This branch just contains the program ID of the mailstore monitor used for the message back end Unity is setup to use. There's no editable configuration information that lives under this branch.

Notifier

The notifier is responsible for all outdials and MWI triggers for the local Unity server. This process logs into the mailbox of every subscriber homed on the local Unity box and "watches" for message activity that may require a notification or MWI event. This includes the AMIS message delivery functionality as well. This registry branch contains default configuration data for the notifier, most of which should not be fiddled with unless TAC specifically requests it. One value under here that is of interest is the alias of the public distribution list that gets all outside caller messages which could not be delivered. Many times sites in the field do not want to use the Unaddressed Messages distribution list created by the Unity setup and delete it without realizing the impact this can have on the system (i.e. "lost messages" from outside callers). It's possible to enter the mail alias of any distribution list in this key, restart Unity and you're good to go.

Note The need for this unaddressed messages distribution list is reduced, but not eliminated with the Unity 4.0 release. By default Unity 4.0 can be configured to check if a subscriber's mailbox is full before taking a message for that subscriber from an outside caller. There's a small resource impact of making such a check across the network each time a message is taken from an unidentified user so this option is configurable system wide.

ResourceLoader

The resource loader is the mechanism Unity uses for providing localized versions running in several languages at once. Components such as the SA send an Id representing a string and a language identifier to the resource loader which returns that string in the appropriate language to display. This branch contains a few global variables used by the resource loader, there's nothing under here which needs to be configured.

Ruler

The ruler component is used by the Arbiter when processing incoming calls to determine where the call should be sent in the Unity system. These rules are visible in the SA under the "call routing rules" pages. In Unity 3.1(3) and later the rules are stored in an SQL table in the UnityDB database but for earlier versions of Unity they were stored in the ROUTING.RUL file on the hard drive. This branch just stores the path to that file. For whatever reason this branch continues to hang around for versions after 3.1(3) even though it contains no usable data.

SA

This branch stores the data for the "Contacts" section under the configuration pages in the SA. It's just a series of text strings for the contact name and phone numbers for the customer and administrator for the Unity box.

SystemParameters

This branch contains some more general properties for the system as a whole. This information could easily be contained in the Commserver Setup branch noted above but for whatever reason another branch of general settings was added by one development group or another along the way. The primary data under here has to do with the Graphical User Interface (GUI) and Telephone User Interface (TUI) languages loaded on the system and what the default languages are for the GUI, TUI and TTS functions. This is also where the number of simultaneous SA sessions can be configured which is exposed in the Advanced Settings tool.

ToolsDepot.

Starting in Unity 3.1(3) the Tools Depot showed up on the desktop. This branch contains information about all the applications and categories visible in that tool. There is also occasionally application specific data such as recent file lists, window size/position and the like stored under here.

TTS

This branch contains information about which TTS engines are installed on the box, the languages supported by each engine and characteristics of that engine. Most notably the pitch, speed and default speaker for the installed TTS engine is set under here.

UMR

This branch is no longer used. The UnityUMR branch contains the information that was once in this branch. Why this was done is not clear at this point and why this branch is still around is even less clear.

UnityUMR

The Unity Message Repository (UMR) is used to store messages that need to be sent to the mail store while that mail store is off line or unavailable to Unity for whatever reason. When Unity is in this state, it's referred to as being in "UMR mode". Messages are stored as pairs of files (the message itself and a routing file indicating where it needs to go) on the hard drive in a directory specified in this registry branch (by default \commserver\UnityMTA). You can also indicate what the minimum amount of disk space can be left on the drive the UMR is storing messages on before Unity stops allowing messages to be stored here (by default 5 MB). If that limit is reached and Unity is still in "UMR mode" then callers will simply not be allowed to leave messages, they will instead get the "fail safe" conversation noted above.

VirtualQueue

There's nothing under this branch, however some keys can be added for debugging Unity's internal call queuing capabilities.

Directory

Just a brief note about information that's in the directory itself. The details on which objects are added to the directory and which objects have additional information added to them is covered in the Architecture Overview chapter.

In versions of Unit prior to 3.0 all Unity directory information was stored right in the Exchange 5.5 directory. All lookups from the SA, conversations and other clients were done to the DOH interface which did lookups via LDAP right in the directory. This was rather slow and presented serious scalability issues among other things.

In Unity 3.0 and later the primary storage location for Unity directory data is the local SQL database. In this model a small subset of the data stored in the local SQL database is also pushed into the directory the Unity server is connected to (either Exchange 5.5, AD or Domino). This is done so other Unity servers on the network can address messages to one another easily and allow transfers across multiple Unity boxes in a "dialing domain" (more on this in the digital networking chapter later). Monitors on the local Unity store "watch" the directory for information about subscribers and location objects from other Unity servers attached to the same directory and pull that information into the SQL database. More details on this process can be found in the Architecture Overview Chapter.

For the most part clients do not go straight to the directory to get at this information, they go to SQL and use the data the monitors have gleaned from the directory for them. This is much faster than accessing the directory itself and is also much less complicated since the client code remains identical regardless of which back end Unity is connected to. All the complexity of dealing with the quirks of the various back ends is concentrated in the monitors and change writers.

There are two exceptions to this rule:

- The Import tools (both in the SA and in the stand alone import tools) have to go to the directory to get mail users that can be imported as subscribers.
- The Internet Voice Connector (IVC) does lookups right to the directory when routing inbound messages. This is necessary since a single IVC can service many Unity servers and it normally does not run on a Unity server so access to SQL on the box could present problems. The IVC handles routing messages that come into Unity from external sources via SMTP, AMIS, VPIM and via the Unity Bridge server. More on this in the Digital Networking chapter later.

Local Files

In Unity 4.0 there is little in the way of configuration information stored in local files on the hard drive itself. As mentioned earlier, moving forward there will be even less of this as Unity moves towards an architecture that will allow for clustering multiple boxes together which will require mechanisms to share all such configuration data across boxes. Most configuration data will eventually end up in a table in SQL which can be configured for remote access and/or replication to other servers on a network. Currently, however, there is some data on the hard drive itself which you should be aware of.

- Switch configuration information. The characteristics of the phone system the Unity box is connected to and the specifics about it's integration data are stored in local files under the commserver\Intlib directory. A switch INI file is referenced in the MIU section of the registry based on what's selected in the switch configuration pages in the SA and, in Unity 4.0, the Unity Telephone Interface Manager (UTIM) application. Various PBXs are supported with serial, analog and digital integration mechanisms. This will be covered in the Switch Integration chapter later.
- Prior to 3.1(3) routing rules exposed in the "call routing" page in the SA were stored in a Unicode file called "routing.rul" stored in the \commserver\support directory. You'll still see this file in later versions of 3.x however it is not used. These rules got moved into SQL so failover configurations between two Unity servers could share schedule information easily.
- The Event Notification Utility (ENU) has an Access database stored by default under \commserver\localize\GAEN\Gaen.mdb. The name of the database reflects it's original name: General Audio Error Notification utility. This database stores information about which event log errors the ENU utility should notify administrators about as well as who to notify and how. This data is exposed in the ENU Administration utility found in the Tools Depot interface or the Unity program group.
- The logs that generate reports are, of course, on the local hard drive as well. By default these are stored under commserver\logs. These are "aged" and cleaned up automatically based on settings in the configuration section of the SA.
- Greetings and Voice Names are both stored in the stream files directory which is by default in \commserver\streamfiles. When the conversation goes to play a voice name or a greeting, it's being pulled off the local hard drive from this directory. Greetings are not pushed around in the directory but voice names are since they are needed for name confirmation when addressing to a subscriber which may be homed on another Unity server in the network. The monitors push and pull voice name information for subscribers and public distribution list in and out of the shared directory. As such if you just copy a voice name WAV file over an existing voice name WAV file in the stream files directory, it will eventually get over written with the old version that's floating around in the directory. Special consideration is required if you want to change/add/remove voice names from subscribers in bulk. We'll cover this in the Administering Unity Programmatically chapter later.
- Of course there has to be an exception to every rule. While MOST voice names and greetings can be found in the stream files directory noted above, there is a big exception to this. All objects created by the Unity setup have their voice names and greetings stored in the default configuration directory which is found under \commserver\localize\defaultconfiguraiton\enu. The reason for doing this has long since been lost in the fog of history but it is something you need to be aware of. The list of default objects created by the Unity setup that have greetings and voice names stored in this directory include:
  - The operator, opening greeting and say goodbye call handlers
  - The default interview handler
  - The example administrator and example subscriber mail users
  - The unaddressed messages, system event messages and all subscribers public distribution lists.

It's important to note that even if you change or delete and rerecord the greetings and voice names on these objects they will still end up being stored in the default configuration directory instead of the stream files directory with everything else.

As noted above, the Unity Message Repository (UMR) stores voice mail messages as pairs of files on the local hard drive until they can be delivered to the companion Exchange server which then routes them to the destination mailbox. If Exchange is off line or the network connectivity to that box is interrupted or the like, those messages will stay on the Unity server's hard drive until connectivity is restored. By default the directory where these files reside is under \commserver\UnityMTA.