



Avaya Media Processing Server Series System Reference Manual

(Software Release 2.1)

**Avaya Business Communications Manager
Release 6.0**

Document Status: **Standard**

Document Number: **P0602477**

Document Version: **3.1.12**

Date: **June 2010**



© 2010 Avaya Inc.
All Rights Reserved.

Notices

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer

Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of this documentation unless such modifications, additions, or deletions were performed by Avaya. End User agree to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

Link disclaimer

Avaya is not responsible for the contents or reliability of any linked Web sites referenced within this site or documentation(s) provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty

Avaya provides a limited warranty on this product. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya's standard warranty language, as well as information regarding support for this product, while under warranty, is available to Avaya customers and other parties through the Avaya Support Web site: <http://www.avaya.com/support>

Please note that if you acquired the product from an authorized reseller, the warranty is provided to you by said reseller and not by Avaya.

Licenses

THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, [HTTP://SUPPORT.AVAYA.COM/LICENSEINFO/](http://SUPPORT.AVAYA.COM/LICENSEINFO/) ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AUTHORIZED AVAYA RESELLER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AUTHORIZED AVAYA RESELLER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA AUTHORIZED RESELLER, AND AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS "YOU" AND "END USER"), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE ("AVAYA").

Copyright

Except where expressly stated otherwise, no use should be made of the Documentation(s) and Product(s) provided by Avaya. All content in this documentation(s) and the product(s) provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software. Unauthorized reproduction, transmission, dissemination, storage, and/or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Third Party Components

Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the Product ("Third Party Terms"). Information regarding distributed Linux OS source code (for those Products that have distributed the Linux OS source code), and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply to them is available on the Avaya Support Web site: <http://support.avaya.com/Copyright>.

Trademarks

The trademarks, logos and service marks ("Marks") displayed in this site, the documentation(s) and product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, or other third parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the documentation(s) and product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party. Avaya is a registered trademark of Avaya Inc. All non-Avaya trademarks are the property of their respective owners.

Downloading documents

For the most current versions of documentation, see the Avaya Support. Web site: <http://www.avaya.com/support>

Contact Avaya Support

Avaya provides a telephone number for you to use to report problems or to ask questions about your product. The support telephone number is 1-800-242-2121 in the United States. For additional support telephone numbers, see the Avaya Web site: <http://www.avaya.com/support>

Table of Contents

Preface	9
Scope	10
Intended Audience	10
How to Use This Manual	11
Organization of This Manual	12
Conventions Used in This Manual	13
Solaris and Windows 2000 Conventions	15
Trademark Conventions	15
Avaya MPS Architectural Overview	17
Overview of the Avaya Media Processing Server (MPS) System	18
System Architecture	19
Hardware Overview	21
Front Control Panel (FCP)	22
Variable Resource Chassis (VRC)	22
Power Supplies	24
VRC Rear Panel	26
Drive Bays	26
Application Processor	26
Network Interface Controller (NIC) or Hub-NIC	27
Telephony Media Server (TMS).....	28
Phone Line Interface	28
Multiple DSP Module (MDM)	31
System LAN Interface	32
Field Programmable Gate Arrays (FPGA) and the Boot ROM	32
TelCo Connector Panel (TCCP)	33
Software Overview	34
Software Environment	35
ASE Processes	36
ASE/VOS Integration Layer	39
VOS Processes	39
System Utilities and Software	51
alarm	51
dlog	52
dlt	53
log	53
PeriProducer	54
PeriReporter	55
PeriStudio	56
PeriView	57
PeriWeb	59
vsh	60

Base System Configuration	63
Base System Configuration	64
System Startup	65
Solaris Startup/Shutdown	67
Windows Startup/Shutdown	69
SRP (Startup and Recovery Process)	70
Manually Starting and Stopping SRP	70
VPS Topology Database Server (VTDB)	71
Restart of Abnormally Terminated Programs	72
Communication with VOS Processes	72
SRP Configuration Command Line Arguments	74
VSH Shell Commands	75
SRP Status	81
Call Control Manager (CCM/CCMA)	82
Startup Files	83
The hosts File	83
User Configuration Files	86
The .xhtrahostsrc File	86
The MPSHOME Directory	87
The MPSHOME/common Directory	88
The MPSHOME/common/etc Directory	88
The srp.cfg File	89
The vpshosts File	93
The compgroups File	95
The gen.cfg File	96
The global_users.cfg File	98
The alarmd.cfg and alarmf.cfg Files	99
The pmgr.cfg File	100
The periview.cfg File	102
The MPSHOME/common/etc/tms Directory	103
The sys.cfg File	103
The tms.cfg File	106
Protocol Configuration Files	123
The \$MPSHOME/packages Directory	125
%MPSHOME%\PERIase - /opt/vps/PERIase . .	127
The /etc/ase.conf file	127
The /etc/services File	129
%MPSHOME%\PERIbrdge - /opt/vps/PERIbrdge	132
%MPSHOME%\PERIdist - /opt/vps/PERIdist .	133
%MPSHOME%\PERIglobl - /opt/vps/PERIglobl	133
%MPSHOME%\PERIview - /opt/vps/PERIview .	134
%MPSHOME%\PERIplic - /opt/vps/PERIplic .	134
%MPSHOME%\PERItms - /opt/vps/PERItms . .	134
The /cfg/atm_triplets.cfg File	135
The /cfg/ps_triplets.cfg File	136

The /cfg/tms_triplets.cfg File	136
%MPSHOME%\PERImps - /opt/vps/PERImps .	137
The MPSHOME/tmscommN Directory.....	138
MPS 500	138
MPS 1000	138
The MPSHOME/mpsN Directory	139
The MPSHOME/mpsN/apps Directory	140
The MPSHOME/mpsN/etc Directory	142
VMM Configuration Files	144
The vmm.cfg File	144
The vmm-mmfcfg File	146
ASE Configuration Files.....	148
The ase.cfg File	148
The aseLines.cfg File	149
CCM Configuration Files.....	151
The ccm_phoneline.cfg File	151
The ccm_admin.cfg File	155
TCAD Configuration Files.....	157
The tcad-tms.cfg File	157
The tcad.cfg File	158
TRIP Configuration Files	159
The trip.cfg File	159
TMS Watchdog Functions	160

Common Configuration 163

Multi-Media Format Files (MMFs)	164
How to Create an MMF File.....	164
Vocabulary MMF Files vs. CMR MMF Files	165
Activating MMF Files	166
Delimited and Partial Loading	168
Audio Playback	169
Custom Loading	171
Using Hash Tables	172
System MMF Files	173
Application-Specific MMF Files	174
Default Vocabulary and Record MMF Files	175
Diagnostics and Reports	176
Synchronizing MMF Files Across Nodes.....	177
ZAP and MMF files on the MPS	177
MMF Abbreviated Content (MAC) File	178
Basic Implementation (Low Volume/Traffic)	178
Advanced Implementation (High Volume/Traffic) ...	181
Updating a Specific Element	185
Exception Processing	187
Log Files	188

- Synchronization (ZAP) Command Summary 191
- Troubleshooting 194
- Call Simulator Facility 195
 - VEMUL Script Format 195
 - Script Control 196
 - Configuration Parameters 196
 - Script Commands 196
 - Statements 197
 - Primitives 198
 - Phone Line Behavior During Simulation 199
 - Call Simulator Conditions and Usage 199
 - Command Line Interface 200
 - Example Call Simulation Script Files 202
- Alarm Filtering 203
 - Filtering Precepts 204
 - Command Line Interface 205
 - alarmf** Command Line Options 206
 - Notation Functionality 207
 - Logical Conditions 209
 - Action Functions 211
 - Filtering Examples 213
- Interapplication/Host Service Daemon Data Exchange 215
 - VMST (VMS) 215
 - Starting Under SRP 215
 - PeriPro Interaction 216
 - Arguments 217
 - Examples: 218
 - VTCPD 220
 - Single Connection to Host 221
 - Multiple Connections to Multiple Hosts 221
 - One Connection Per Line 222
 - Multiple VTCPD Daemons 222
 - Host Connections 222
 - Attaching to VMST 225
 - Message Format 227
 - Message Identification (ID) 231
 - Connection Capacity 233
 - Application-Host Interaction Configuration Options 234
 - Queuing Requests 236
 - Monitoring Host Connections 238
 - Backup LAN 239
 - VFTPD 240
 - Specifying a Port 240
 - Automatic Startup 241
 - Automatic FTP Logins 241
 - Identifying the Configured Host Computers 242

Configuration Procedures and Considerations	243
Making Changes to an Existing System	244
Adding Spans	244
Modifying the Span Resource Set	244
Changing Pool/Class Names	245
Renumbering a Component	245
Renaming a Solaris MPS Node	246
Renaming a Windows MPS Node	247
Introducing a New Node.	248
Enabling Statistics Collection.	249
Debug Terminal Connection	250
Connection Using a Dumb Terminal or PC	250
Connection from the System Console	250
Verifying/Modifying Boot ROM Settings	252
DCC Boot ROM	252
TMS Boot ROM	256
NIC Boot ROM	260
Resetting the NIC	264
TMS Computer Telephony (CT) Bus Clocking	265
N+1 Redundancy	267
Sample MPS 1000 N+1 Redundancy System Configuration	267
TRIP Failback.	268
Directory Layout on a Secondary (Backup) Node	269
Least Cost Routing Daemon.	271
Redundancy Configuration Daemon (RCD).	271
The Failover/Failback Process	273
Installation and Configuration	274
Create the Secondary Node	274
TMSCOMM Component Configuration	274
Edit the vpshosts File	275
Edit the tms.cfg File	276
Edit TRIP and RCD Configuration Files	276
Edit the gen.cfg file	276
PMGR configuration	277
Media Directories	278
First Startup After Configuration	280
Verifying N+1 Functionality	283
Failover	283
Failback	284
Speech Server Resources in N+1 Redundancy.	285
Pool Manager (PMGR)	288
Terminology	288
Resource Object	288
Allocation/Deallocation	288
Pool	289

- Resource Identifier/String 289
- Scheme 289
- Configuration 289
 - Port Service States 291
 - Network Failure Detection (Pinging) 291
- Database Conversion 292
 - Platform Conversion..... 292
 - Starting a Reader 292
 - Starting a Writer 292
 - Database Format Conversion 293
 - Reader/Writer Synchronization 293
 - File Size Limitations..... 293
- Call Monitoring 294
 - Listening to Calls 294
- Security 297**
 - Antivirus Software 298
 - Secure Shell 299
- Index 301**

Preface

Scope

The *Avaya Media Processing Server Series System Reference Manual* details the procedures and parameters for configuring the Avaya Media Processing Server (MPS) Series system for online operation in a variety of telephony environments. In addition, this manual provides configuration parameters and basic file information for elements common to all MPS within the network. Note, however, that though there are two basic products available in the MPS system - a single rack-mounted version known as the Avaya MPS Series and a cabinet enclosed network configuration which relies on the MPS 500 - this manual deals almost exclusively with the latter.

In addition to this document, the *Avaya Media Processing Server Series System Operator's Guide* may be particularly helpful. They provide a road map through the major functions in the daily operation and monitoring of the MPS system. For a list of other user manuals, see the *Reference Material* link in PeriDoc.

Intended Audience

This manual is intended for the persons who will be configuring the MPS for a specific site and/or maintaining it from a particular perspective. The reader should be familiar with telecommunications and computer equipment, their functions, and associated terminology. In addition, the reader must be familiar with the characteristics of the specific installation site, including site-specific power systems, computer systems, peripheral components, and telephone networks.

Some of the material covered here involves the configuration of basic and critical MPS parameters. Small inaccuracies in the configuration of these parameters can impede system performance. Individuals without highly specialized knowledge in this area should not attempt to change the defaults.

This guide assumes that the user has completed an on-site system familiarization training program conducted as part of the initial system installation. Basic knowledge of the Solaris and/or Windows 2000 operating system(s) is also assumed.

How to Use This Manual

This manual uses many standard terms relating to computer system and software application functions. However, it contains some terminology that can only be explained in the context of the MPS system. Refer to the *Glossary of Avaya Media Processing Server Series Terminology* for definitions of product specific terms.

It is not essential that this document be read cover-to-cover, as the entire contents is not universally applicable to all MPS environments. It is essential, however, that there is a clear understanding of exactly what information pertains to your environment and that you can identify, locate, and apply the information documented in this manual. Later, you can use the Table of Contents to locate topics of interest for reference and review.

If you are reading this document online, use the hypertext links to quickly locate related topics. Click once with your mouse while positioned with your cursor over the hypertext link. Click on any point in a Table of Contents entry to move to that topic. Click on the page number of any Index entry to access that topic page. Use the hyperlinks at the top and bottom of each HTML “page” to help you navigate the documentation. Pass your cursor over the Avaya Globemark to display the title, software release, publication number, document release, and release date for the HTML manual you are using.

For additional related information, use the *Reference Material* link in PeriDoc. To familiarize yourself with various specialized textual references within the manual, see [Conventions Used in This Manual on page 13](#).



Periphonics is now part of Avaya. The name Periphonics, and variations thereof, appear in this manual only where it is referred to in a product. (For example, a PeriProducer application, the PERImpS package, the **perirev** command, etc.)

Organization of This Manual

This document is designed to identify the procedures and configuration parameters required for successful MPS operations. It provides an overview of the MPS system and proceeds to document both basic and common system parameters. The following passages provide an overview of the information contained in each area of this manual.

Chapter 1 - Avaya Media Processing Server Series Architectural Overview

Provides a description of the MPS system and an overview of its hardware and software. Diagrams and describes the MPS structure, its software processes, and identifies other system utilities.

Chapter 2 - Base System Configuration

Describes and diagrams the system directory structure and startup and shutdown, delineates the Startup and Recovery Process (SRP), and details MP\$HOME and all required configuration files.

Chapter 3 - Common Configuration

Documents the facilities available on all (common) MPS platforms. Details MultiMedia Format (MMF) file creation and utilization. Also covers call simulation, alarm filtering, and exchange of data between applications, hosts, and MPS.

Chapter 4 - Configuration Procedures and Considerations

Contains common procedures and comprehensive considerations for modifying existing systems and adding features.

Appendix A - Process and Utility Command Summary

Lists commands for some of the processes and utilities most commonly interacted with in the MPS system. Provides brief definitions for each and links to more detailed information.



Appendix B - Avaya MPS Specifications

Contains physical, electrical, environmental, and interface specifications for the MPS.



Conventions Used in This Manual

This manual uses different fonts and symbols to differentiate between document elements and types of information. These conventions are summarized in the following table.

Conventions Used in This Manual Sheet 1 of 2

Notation	Description
Normal text	Normal text font is used for most of the document.
<i>important term</i>	The Italics font is used to introduce new terms, to highlight meaningful words or phrases, or to distinguish specific terms from nearby text.
system command	This font indicates a system command and/or its arguments. Such keywords are to be entered exactly as shown (i.e., users are not to fill in their own values).
command, condition and alarm	Command, Condition and Alarm references appear on the screen in magenta text and reference the <i>Command Reference Manual</i> , the <i>PeriProducer User's Guide</i> , or the <i>Alarm Reference Manual</i> , respectively. Refer to these documents for detailed information about Commands , Conditions , and Alarms .
file name / directory	This font is used for highlighting the names of disk directories, files, and extensions for file names. It is also used to show displays on text-based screens (e.g., to show the contents of a file.)
on-screen field	This font is used for field labels, on-screen menu buttons, and action buttons.
<KEY NAME>	A term that appears within angled brackets denotes a terminal keyboard key, a telephone keypad button, or a system mouse button.
<i>Book Reference</i>	This font indicates the names of other publications referenced within the document.
cross reference	A cross reference appears on the screen in blue text. Click on the cross reference to access the referenced location. A cross reference that refers to a section name accesses the first page of that section.
	The Note icon identifies notes, important facts, and other keys to understanding.
	The Caution icon identifies procedures or events that require special attention. The icon indicates a warning that serious problems may arise if the stated instructions are improperly followed.

Conventions Used in This Manual Sheet 2 of 2

Notation	Description
	The flying Window icon identifies procedures or events that apply to the Windows 2000 operating system only. ¹
	The Solaris icon identifies procedures or events that apply to the Solaris operating system only. ²

1. Windows 2000 and the flying Window logo are either trademarks or registered trademarks of the Microsoft Corporation.
2. Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

Solaris and Windows 2000 Conventions

This manual depicts examples (command line syntax, configuration files, and screen shots) in Solaris format. In certain instances Windows 2000 specific commands, procedures, or screen shots are shown where required. The following table lists examples of general operating system conventions to keep in mind when using this manual with either the Solaris or NT operating system.

	Solaris	Windows 2000
Environment	\$MP\$HOME	%MP\$HOME%
Paths	\$MP\$HOME/common/etc	%MP\$HOME%\common\etc
Command	<command> &	start /b <command>

Trademark Conventions

The following trademark information is presented here and applies throughout for third party products discussed within this manual. Trademarking information is not repeated hereafter.

Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows, Windows 2000, Internet Explorer, and the Flying Windows logo are either trademarks or registered trademarks of Microsoft Corporation.

Netscape® and Netscape Navigator® are registered trademarks of Netscape Communications Corporation in the United States and other countries. Netscape's logos and Netscape product and service names are also trademarks of Netscape Communications Corporation, which may be registered in other countries.

This page has been intentionally left blank.



1

Avaya MPS Architec- tural Overview

This chapter covers:

- 1. Overview of the Avaya
Media Processing Server
Series System**
- 2. System Architecture**
- 3. System Utilities and
Software**

Overview of the Avaya Media Processing Server System

The Avaya Media Processing Server (MPS) Series products comprise hardware and software to create a call and web-based processing environment. These systems integrate the call processing environment with speech, telephony, data communications, and transaction processing functions. The platform is based on the Avaya Telephony Media Server (TMS) which provides high phone port densities and increased user flexibility and extensibility. The basic TMS assembly provides resources for telephony media management including switching/bridging, digital signal processing, voice and data memory, and network interfaces. A variety of interactive voice processing applications are accommodated, from simple information delivery services to complex multimedia (voice/fax/data/web) call processing implementations with local databases, multiple services, and transaction processing functions.

The MPS system supports a wide selection of telephony and host computer connectivity interfaces for easy integration into an existing data-processing/communications environment. It also includes a set of easy to use object-oriented Graphical User Interface (GUI) tools. These tools are used for:

- application and vocabulary development
- system configuration, control, and monitoring
- collection and reporting of statistical data
- access to on-line documentation and its concurrent implementations

The application development environment provides a totally graphical environment for the entire application life cycle, and also allows typically phone-line applications to be ported over to Internet-based Web usage. The PeriProducer GUI is the suggested tool of choice for application development. The PeriWeb package allows these phone line applications to be run as interactive World Wide Web apps.

The MPS systems employ industry standards and distributed processing in an open architecture, allowing plug-in integration of future technological developments. In addition, networking elements of the MPS support multiple LAN/WAN interfaces, providing an environment ready for distributed computing.

This chapter of the *Avaya Media Processing Server Series System Reference Manual* presents an overall view of the MPS hardware and software, describes the software processes responsible for operations, and provides a series of diagrams that illustrate both hardware and software relationships.

Base System Configuration on page 64, documents the process of getting the MPS system up and running, identifies the individual configuration files, details some of the newer processes, and describes the directory structure of the operating environment and predefined environment variables.

System Architecture

The MPS family is designed with a flexible hardware and software architecture that is highly scalable. System models range from small (48 ports) to large networked configurations of tens of thousands of ports. The same basic hardware and software components are used for all configurations. Individual systems usually vary only in application/transaction processor performance, capacity for additional ports (TMS'), and optional feature software/hardware (for example, Call Progress Detection, Speech Recognition, or Caller Message Recording).

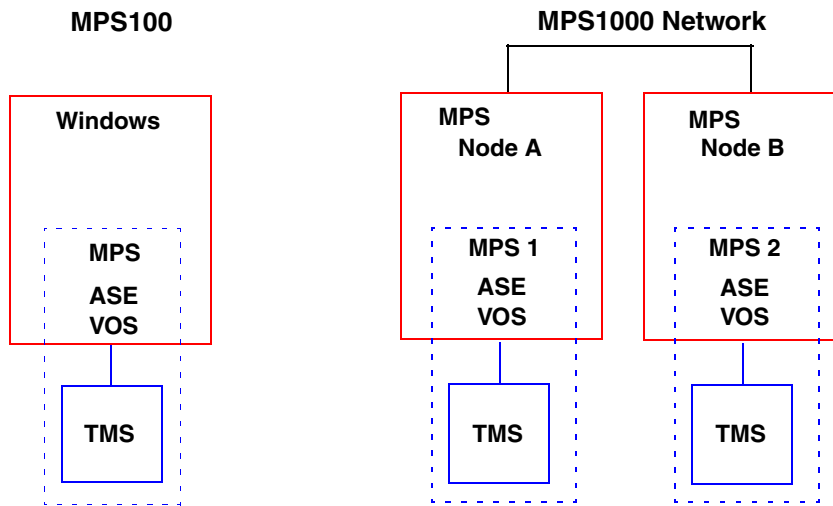
Architecture of the MPS is based on a Sun Microsystems SPARC system processor running the Solaris operating system or an Intel processor running Windows 2000. The system processor is connected to one or more Telephony Media Servers (TMS). The TMS is a flexible platform that provides switching, bridging, programmable resources, memory, and network interfaces to execute a comprehensive set of telephony and media functions.

Each MPS system consists of a Solaris or Windows host node running OS and MPS software, and one or more TMS' responsible for the bulk of the actual telephony processing. One TMS is required for each MPS defined on the node. A multiple node configuration is referred to as the MPS *Network*. The following diagrams illustrate the two basic products available in the MPS system: a single rack-mounted version, known as the MPS100, which is available on the Windows platform only, and a cabinet enclosed networked configuration which relies on the MPS1000 and is available on both the Windows and Solaris platforms. Typically, the MPS100 contains only 2 spans (though it may contain up to 8) and only 1 Digital Communications Controller (DCC) card, and does not support bridging outside the TMS. Conversely, the MPS1000 is the high-capacity model, with 4 TMS' per chassis and up to 4 chassis per cabinet. It can support up to ten thousand ports with the ability to bridge between any two regardless of the chassis the ports are in with respect to each other. This manual deals almost exclusively with the MPS1000.

The flexibility inherent in the product line allows the MPS networks to incorporate numerous different designs. For additional information and configurations, see the *Avaya Media Processing Server Series 1000 Transition Guide*. For information on using the MPS, see the *Avaya System Operator's Guide*.



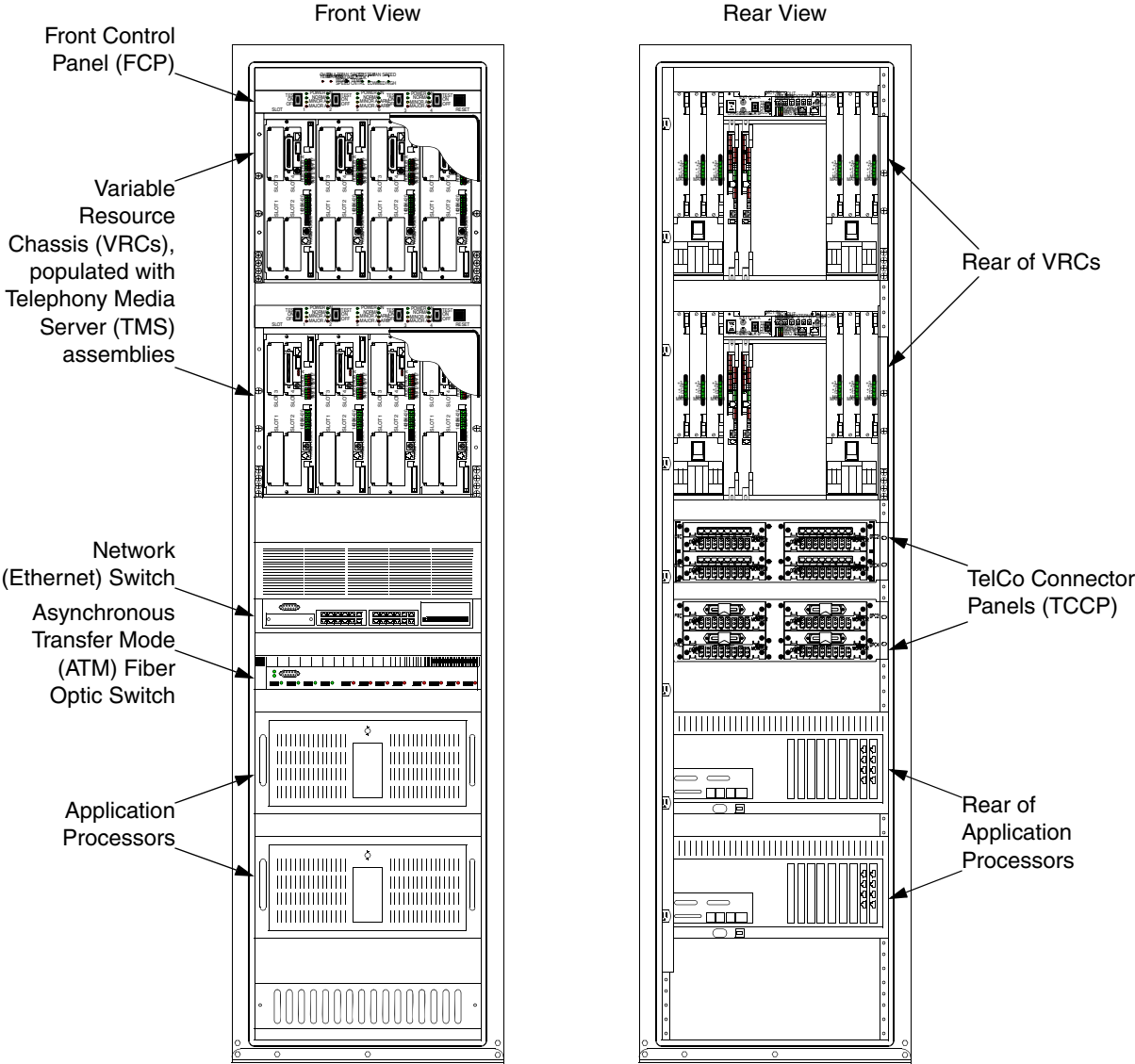
Though the *Avaya Media Processing Server Series 1000 Transition Guide* is typically used by those migrating from a previous version of our transaction processing systems, it also contains information of interest to those new to the product line. Such information should be used in that context only.



Single Media Processing Server 100 and Basic Media Processing Server 1000 Network

Hardware Overview

Typical system hardware includes a SPARC (Solaris) or Intel (Windows) application/transaction processor and related computer components (such as hard drive and RAM) and TMS hardware, including storage for speech and data files, a telephone interface card, network interface cards, power supplies, and various voice processing modules. The major hardware components that make up the MPS1000 are shown in the following illustration (MPS100 information is contained in a separate manual). Each of these is further dissected and discussed in the paragraphs that follow. See the *Avaya Media Processing Server Series System Operator's Guide* regarding details on system monitoring and control and specific analysis of panel switches and LEDs.

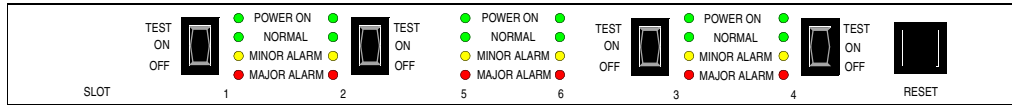




For detailed information on the physical, electrical, environmental, and interface specifications of the Avaya Media Processing Server (MPS) Series, please refer the *MPS Specifications chapter in the Avaya MPS Hardware Installation and Maintenance* manual.

Front Control Panel (FCP)

One FCP is present for each VRC in the system. The FCP provides separate power controls and status indicators for each TMS (by chassis slot).

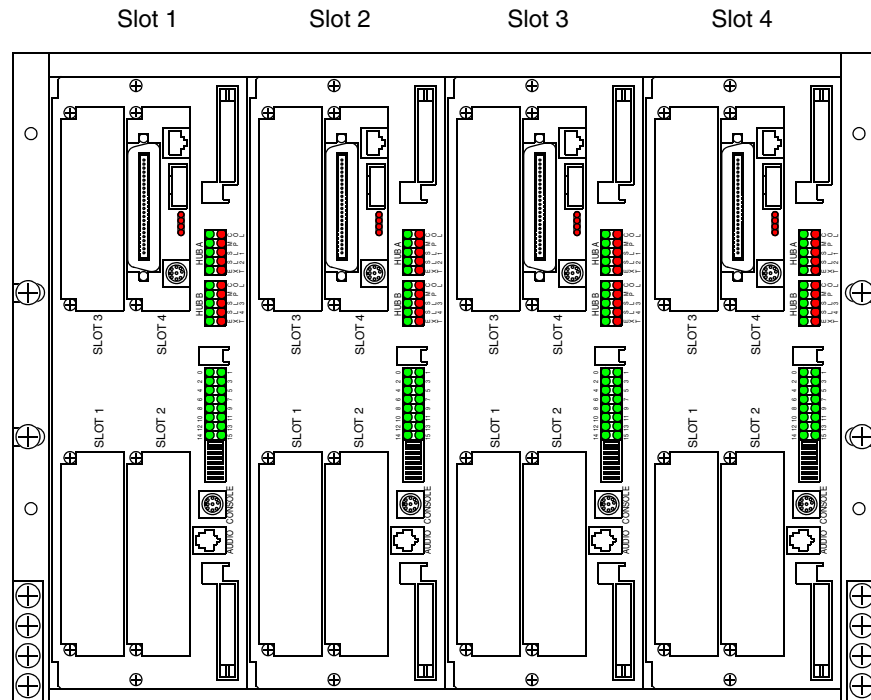


FCP Front View

Variable Resource Chassis (VRC)

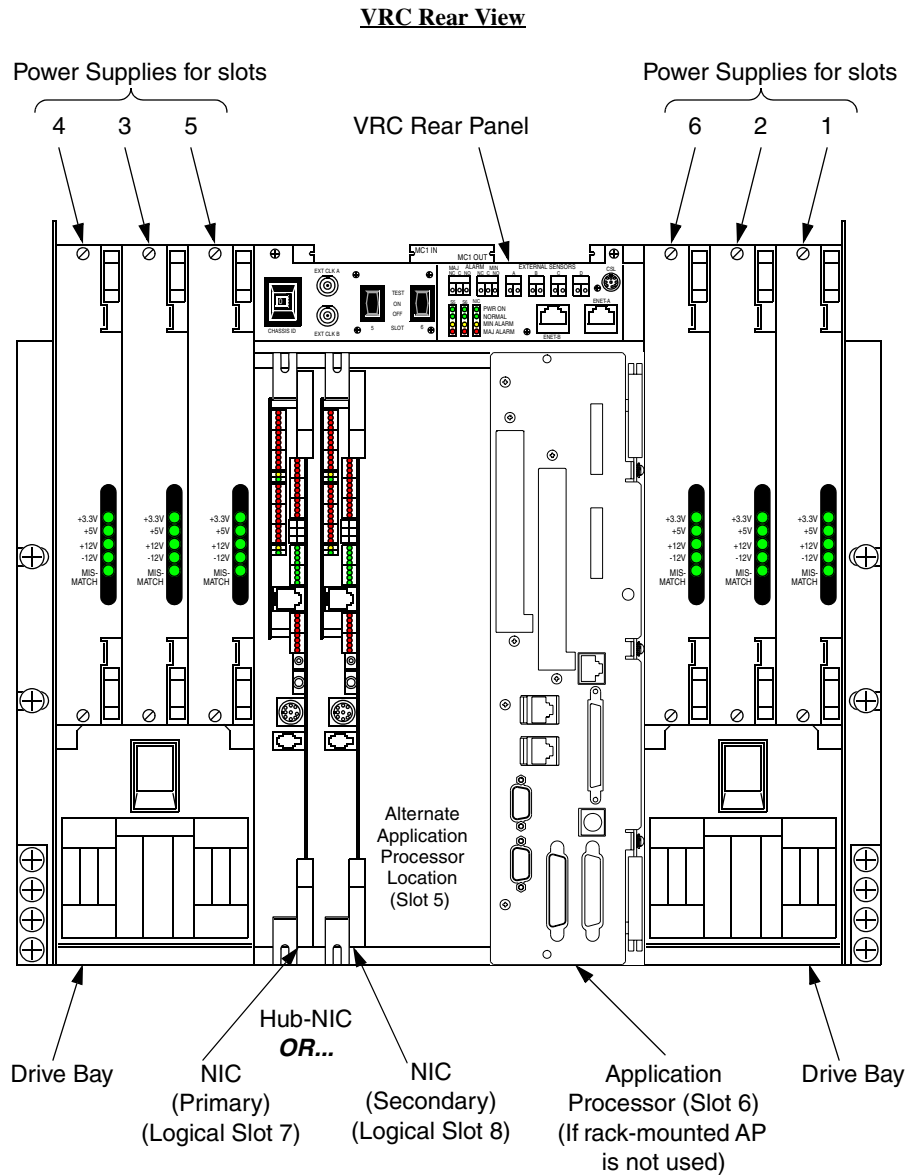
The VRC is a versatile chassis assembly that is used in several Avaya product lines. The VRC has four front and two rear plug-in slots, and contains:

- Up to four TMS assemblies
- One or two application processor board(s) (rear; not present if rack mounted application processor(s) are used)
- Two Network Interface Controllers (NICs) or one Hub-NIC
- Up to six power supplies, one for each populated slot
- Two available drive bays

VRC Front View (Populated with Four TMS?)

The VRC backplane is located midway between the front and rear of the chassis. The backplane contains connectors for the modules that plug into each slot, front and back. The backplane provides connections for:

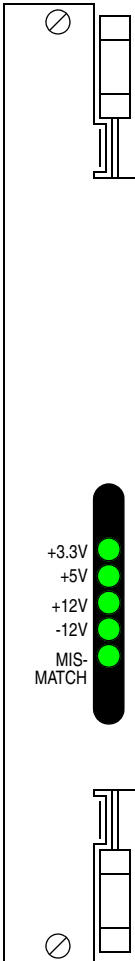
- Inter-module signals
- Power from the power supplies to the module slots
- A Time Delay Multiplexing (TDM) bus for PCM (voice/audio) communications between the TMS assemblies
- Clocking signals for the TDM bus



In multiple chassis and cabinet systems, some VRCs do not contain all the assemblies listed above.

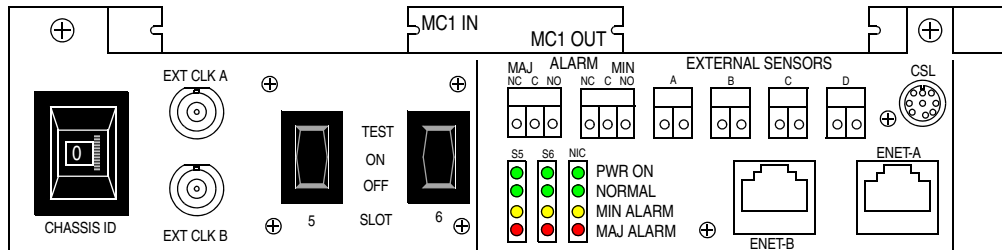
Power Supplies

Each slot in the VRC has a separate power supply dedicated to it. The power supplies are identical and can be installed in any of the six locations for a slot that requires power. The slot that each power supply is associated with is indicated on the decals on the drive bay doors. There is no dedicated power supply for the NIC slot.



VRC Rear Panel

The rear panel of the VRC contains indicators, switches, and connectors for maintenance, configuration, and connection to other system components. The power switches for slots 5 and 6 are also located here, as well as the chassis ID wheel.



Drive Bays

These bays contain the slots for and physical location of the system hard drives when VRC-mounted application processors are used. Generally one drive is present per processor, but additional drives may be added if system performance requires them.

Application Processor

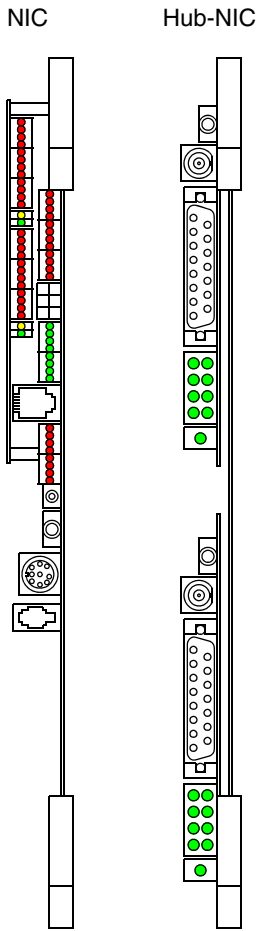
In VRC-mounted configurations, the application processor is a “stripped down” version of a Solaris or Windows computer: it contains the CPU, memory, and printed circuit boards needed for both standard OS functions as well as basic MPS1000 transaction processing. One application processor is present per VRC in slot 6, but if the VRC is populated with multiple TMS’ (which may in turn contain more than one phone line interface card) and large numbers of spans, system performance may be degraded and require the addition of another processor.

In typical rack-mounted configurations, there is one application processor per VRC, and they are mounted at the bottom of the cabinet. This application processor is similar in makeup to a typical Solaris or Windows computer. In either form, an additional application processor may be added where instances of dual redundancy is desired.

Network Interface Controller (NIC) or Hub-NIC

Each VRC in the system contains either two NICs (primary and secondary) or a single Hub-NIC. The Hub-NIC plugs into the NIC slot in back of the VRC, and contains two network hubs for the chassis Ethernet. It is generally used only in single chassis systems. In multiple chassis systems, two NICs are used. In this case a *midplane* board is installed over the backplane connector of the NIC slot, effectively splitting the slot and providing separate connectors for each NIC. The two connectors on the midplane board are logically assigned to slot 7 (primary) and slot 8 (secondary) for addressing.

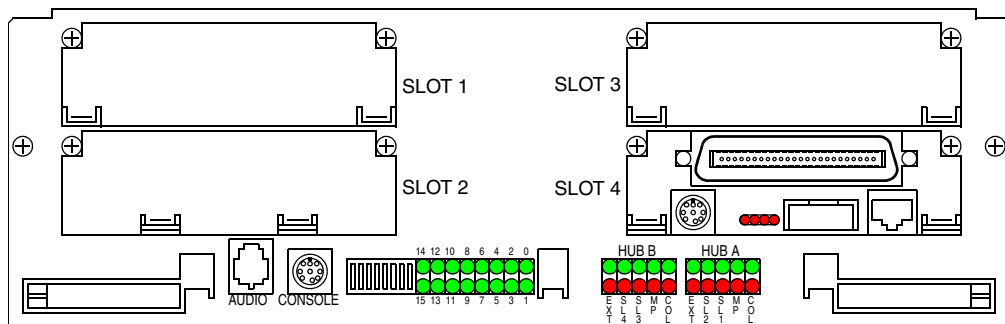
The NICs have additional functionality such as system monitor capabilities, watchdog timer, and alarm drivers, and can interface from the intra-chassis Pulse Code Modulation (PCM) highways to a fiber optic Asynchronous Transfer Mode (ATM) switching fabric. The NICs receive power from any installed power supply that is on.



Telephony Media Server (TMS)

The TMS is the core functional module of the Avaya Media Processing Server (MPS) Series system. It provides a versatile platform architecture for a broad range of telephony functions with potential for future enhancement. The basic TMS assembly consists of a motherboard and mounting plate containing front panel connectors and indicators.

TMS Assembly Front View



The TMS motherboard provides most essential functions for telephony and telephony media management, including network and backplane bus interfaces, local memory, digital signal processors, tone generators, local oscillators, and Phase-Lock Loop (PLL) for Computer Telephony (CT) bus synchronization with other TMS' and the chassis. The motherboard contains a riser board that allows up to four additional modules to be plugged in. The TMS motherboard also contains six Digital Signal Processors (DSPs) which can be configured for communications protocols and to provide resources.

Phone Line Interface

A TMS contains at least one phone line interface card, which can be a single [Digital Communications Controller \(DCC\)](#) (see page 29) or up to three [Analog Line Interface \(ALI\)](#) (see page 30) (a second DCC will be present if Voice over Internet Protocol [VoIP] is installed). Though digital and analog line interfaces cannot be combined in the *same* TMS, *multiple* TMS systems can contain any combination of digital and analog lines in the VRC. Any line can be either incoming or outgoing, and all ports are nonblocking (i.e., any port can be bridged to any other port). The TMS can also be populated with a [Multiple DSP Module \(MDM\)](#) (see page 31), in one or more of the remaining open slots. Although the motherboard has local digital signal processors, the MDM provides additional resources for systems that require them.

A single TMS can support up to eight digital T1 (24 channels/span for a total of 192 lines) or E1 (30 channels/span for a total of 240 lines) spans by using an individual DCC to connect to the Public Switched Telephone Network (PSTN). If some of the lines are used exclusively for IVR resources, one or more spans may be dedicated. Spans dedicated as such are connected directly in clear channel protocol. Supported digital protocols include in-band T1/E1 and out-of-band SS7 and ISDN.

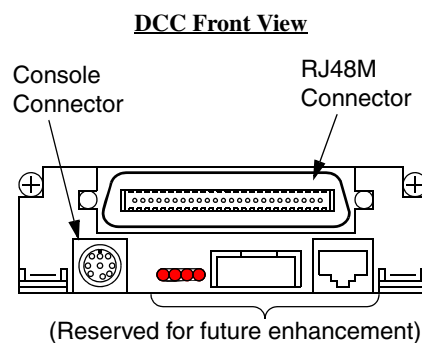
In addition a TMS can support up to 72 analog lines by using three ALI boards (24 lines per ALI). The standard analog interface supports common two-wire loop-start circuits.

Information on configuration and application of phone line protocols and interfaces can be found in the *Avaya Media Processing Server Series Telephony Reference Manual*.

Digital Communications Controller (DCC)

The DCC provides the digital phone line interfaces for the system. It can be plugged into any of the four slots of the TMS. The DCC is dedicated for either a T1 or E1 system, and connects to the PSTN via an RJ48M connector (up to eight spans). The DCC is also capable of interfacing with a telephony network using VoIP. A DCC-VoIP has no telephony connector on the front panel. Only one DCC is typically installed in the TMS, unless the system is also using VoIP, in which case the DCC-VoIP will also be installed. The DCC cannot be combined with an ALI in the same TMS.

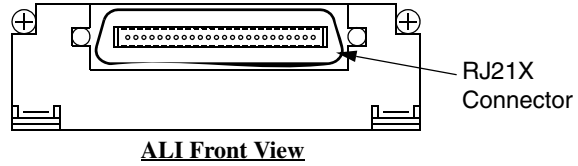
A serial console connector is provided for diagnostic purposes and for verifying and configuring the boot ROM (see [Verifying/Modifying Boot ROM Settings](#) on page 252 for details). Other connectors and indicators are provided on the DCC front panel but are reserved for future enhancement.



Analog Line Interface (ALI)

The ALI provides a phone line interface to the system for up to 24 analog phone lines. It connects to the PSTN via an RJ21X connector on the front panel. The standard analog interface supports common two-wire loop-start circuits. There are no other connectors or indicators on the front of the ALI.

Up to four ALIs can be installed in a TMS, although three is typical since one of the four TMS slots is usually occupied by an MDM. ALIs cannot be combined with a DCC in the same TMS.

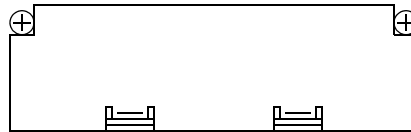


Multiple DSP Module (MDM)

A resource must be available on the system for an application to use it. If the resident DSPs are fully allocated to resources or protocols, capacity for more resources can be added by installing a Multiple DSP Module (MDM) in an open TMS slot and loading the image definitions for the resources required. These resources are in addition to the MPS resource itself. Examples of TMS supported resources are:

- **Player (ply)** - Vocabularies or audio data can be played from local memory on the TMS motherboard.
- **DTMF Receiver (dtmf) and Call Progress Detection (cpd)** - Phone line events such as touch-tone entry, hook-flash, dial tone, busy signals, etc. can be detected.
- **Tone Generator (tgen)** - In lieu of playing tones as vocabularies, DTMF and other tones can be generated.
- **R1 Transmit (r1tx), R1 Receive (r1rx), and R2 (r2)** - Tone generators and detectors to support R1 and R2 protocols.

The MDM contains 12 DSPs for configuration of additional resources. There are no indicators or connectors on the front panel of the MDM. The only visible indication that an MDM is installed in a TMS slot (versus a blank), is the presence of bend tabs near the center of the front bracket that secure it to the MDM circuit board.



MDM Front View

Configuration of resources and protocols is covered in [Base System Configuration](#) on page 64.

System LAN Interface

The TMS interfaces with the system Local Area Network (LAN) via Ethernets using TCP/IP. The chassis Ethernet is connected via the VRC backplane to separate hubs on the chassis NIC or Hub-NIC (see [VRC Rear View on page 24](#)). If there is a failure on the master Ethernet (controlled by the first NIC), the secondary NIC takes control of all Ethernet A, system clocking, and ATM functions. The switchover is virtually instantaneous and the inherent error correction of TCP/IP prevents loss of data.



The redundant Ethernet is only for backup of the primary Ethernet. Ethernet A is the ONLY Ethernet supported between the chassis and the Application Processor. There is no support for dual redundant Ethernet.

Field Programmable Gate Arrays (FPGA) and the Boot ROM

The TMS and the modules that plug into it (i.e., DCC, MDM, and ALI) contain FPGAs. An FPGA is a generic microchip that has no inherent functionality. It contains arrays of generic logic elements (e.g., gates) that are software configurable. The software that configures the FPGA is called an *image*, and the image typically commands the FPGA to assume the functionality of a designed logic circuit. A hardware architecture based on FPGAs is very powerful and flexible because:

- A greater degree of complex logic functionality can be achieved in a relatively smaller board space with fewer circuit components than if dedicated circuit components and hard board wiring were used. This also provides greater circuit reliability.
- Functionality can be enhanced without hardware redesign or even removal and replacement. Upgrades can be done in the field by loading a new image definition.

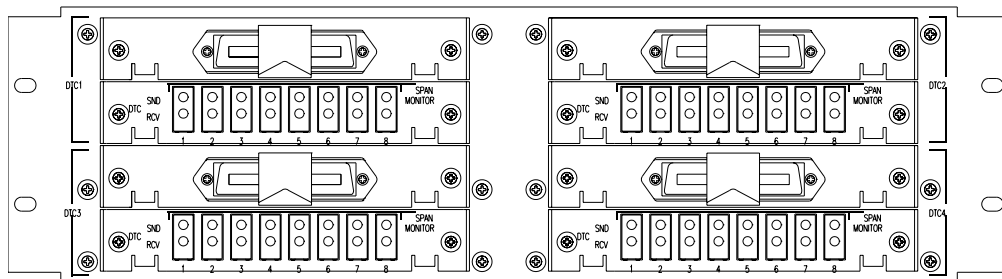
FPGAs are dynamic devices in that they do not retain their image definition when power is removed. The image definition for each device is loaded from an image definition file (*.idf) during the system boot sequence. The TMS contains a boot ROM that statically stores the names of the .idf files for the devices contained on its motherboard and the modules that are plugged in.

Whenever a new system is installed, has components added or replaced, or the system is upgraded, the boot ROM should be verified and, if necessary, modified by Certified Avaya Support Personnel. Details concerning boot ROM verification can be found at [Verifying/Modifying Boot ROM Settings on page 252](#).

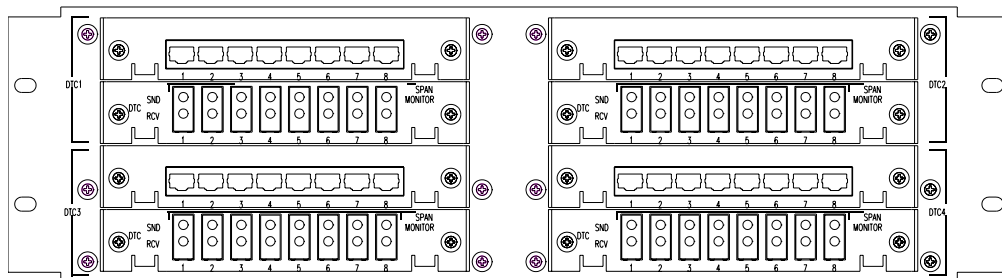
TelCo Connector Panel (TCCP)

The TCCP provides a built-in platform for connecting to the Public Switched Telephone Network (PSTN) and for conveniently breaking out and looping-back spans for monitoring or off-line testing. One TCCP can support up to four TMSs and can be configured with RJ48M or RJ48C connectors for each TMS.

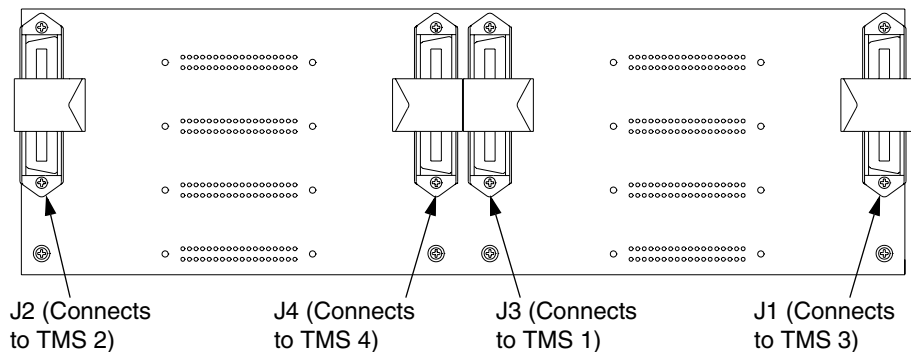
TCCP with RJ48M Interfaces



TCCP with RJ48C Interfaces



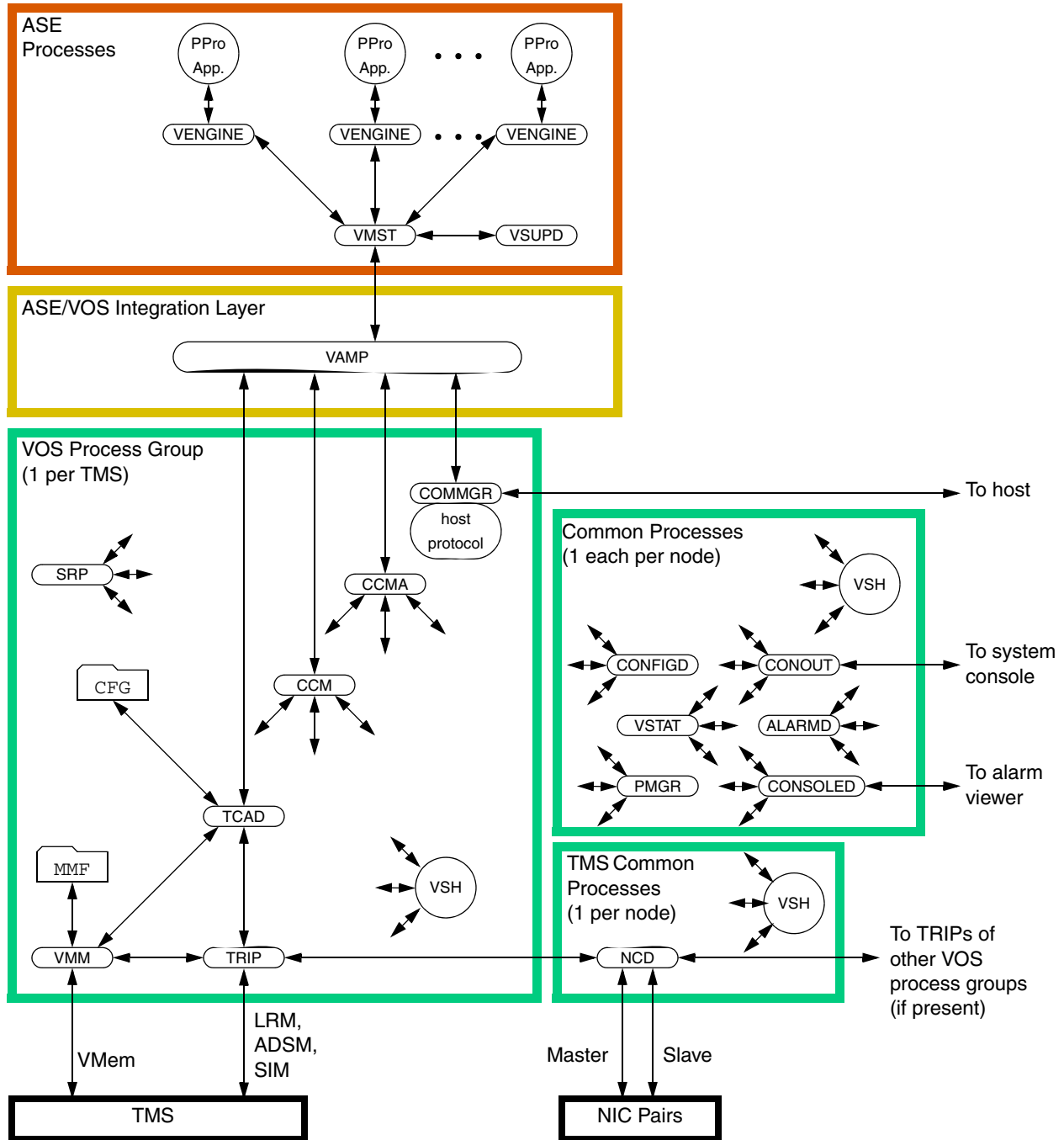
TCCP Rear View



The TCCP is connected to each TMS from the corresponding connector on the TCCP back panel by a direct feed RJ48M cable. In TCCP equipped systems, PSTN connections are made at the TCCP using the RJ48M or RJ48C connectors on the front of the panel. A pair of bantam jacks (SND and RCV) is provided for each span connected to the TCCP. The bantam jacks are resistor isolated and can be used for monitoring only. The bantam jacks cannot be used to create span loop-back connections. Loop-back connections for testing purposes can be made between TMSs or spans using special crossover cables. For details, see the *Avaya Media Processing Server Series 1000 Transition Guide*.

Software Overview

The following illustration shows the functional arrangement of the ASE and VOS processes for MPS release 1.x. Though many of the processes are similar to those of release 5.x, there are several new and revised processes, all of which are described in the paragraphs that follow.



Software Environment

The MPS software components are categorized into two process groups: VOS (Voice Operating Software) and ASE (Application Services Environment).

The VOS software process group comprises the main system software required to run the MPS system. The ASE software process group contains the software required to develop and execute applications.

VOS and ASE software processes have been designed to operate in an open systems Solaris or Windows environment. All speech, telephony, and communications functions run under Solaris or Windows, and require no additional device drivers. VOS uses the standard Solaris or Windows file system for managing all speech/fax data. A set of GUI tools provides for application development and system management.

Some VOS and ASE software processes are common to all MPS components defined on a specific host node; these are located in the GEN subcomponent of the common component on that node (and defined in the file `$MPSHOME/common/etc/gen.cfg`). Other VOS processes are unique to each defined MPS component, and are part of the VOS subcomponent of the MPS component (and defined in `$MPSHOME/mpsN/etc/vos.cfg`). The NCD process, on the other hand, is part of the VOS subcomponent of the `tmscomm` component (and defined in `$MPSHOME/tmscommN/etc/vos.cfg`). This TMS-specific process requires one instance per node; other common processes also require that only a single instance of the process execute on a node. Processes that are unique to each component require an instance of each process be executed for each MPS component defined on the node. When uncommented in their respective `gen.cfg` or `vos.cfg` files, these processes are started by the Startup and Recovery Process (SRP). (For a more comprehensive discussion about SRP, see [SRP \(Startup and Recovery Process\)](#) on page 70.)

Individual applications are executed by means of a separate instance of the ASE process VENGINE for each instance of the application's execution. There are three major types of applications:

- *Call processing applications* are assigned to physical phone lines. A separate instance of both the application and VENGINE process is required for each physical phone line to which the application is assigned.
- *Administrative applications* perform system maintenance functions and support the call processing applications. They are not assigned with physical phone lines. However, they also require a separate instance of VENGINE for each instance of the application.

Applications can communicate with each other by means of shared memory or message passing.

ASE Processes

The Application Services Environment (ASE) process group is comprised of software required to develop and execute applications. ASE processes include:

Process	Description
VENGINE	The application execution process. One VENGINE process is required for each MPS application (call processing, web based, and administrative).
VMST	VENGINE Message Server - Extended. Manages MPS messages related to VENGINE applications. This process also can be used to bridge messages in a multi-MPS environment.
VSUPD	Collects application-specific statistics (as opposed to system statistics).

- VMST, and VSUPD are node-specific processes and require only one occurrence of the process for each host node regardless of the number of components defined on the node.
- VENGINE is an application-specific process. One occurrence of VENGINE must execute for each application assigned to an MPS line.

VENGINE

VENGINE is the application-specific ASE software process. It is responsible for the execution of each occurrence of an application that is assigned to an MPS. One VENGINE process is required to execute for each occurrence of a call processing, web based, or administrative application. Administrative applications are not associated with physical phone lines and perform system maintenance operations and support call processing applications.

Additionally, VENGINE is used to execute all or part of an application while it is under development. It can run all or part of the application so that the logic paths and application functions may be tested.

VENGINE is located in `$MPSHOME/PERIase/bin` on Solaris systems and `%MPSHOME%\bin` on Windows systems, and can be initiated from the command line or by starting an application with the PeriView Assign/(Re)Start Lines tool (see the *PeriView Reference Manual* for more information on the latter). Applications that require ASE processes are located in the `$MPSHOME/mpsN/apps` directory. For additional information about these applications, see [The *MPSHOME/mpsN/apps Directory* on page 140](#). VENGINE makes connections to both these applications and VMST. For additional information on VENGINE, see the *PeriProducer User's Guide*.

VMST

VMST (VENGINE Message Server - Extended) is an ASE software process that

performs message server functions for VENGINE. It funnels VOS messages that have been translated by VAMP to VENGINE processes and service daemons. VMST interprets and supports all pre-existing VMS options, allowing scripts incorporating them to continue functioning under the present release without any modifications.

The advent of the TMS brings about an increase in the number of lines supportable on a single platform, as well as an increase in potential message traffic. In order to handle the increase in addressable lines, this modified version of VMS was created (previously, VMS addressing was limited to a one-to-one correspondence of VMS to CPS/VPS). Though VMST can still act on behalf of a single MPS, VMST can also address the new paradigm by supporting many real or virtual MPS' in a single process (the VMST process assumes the functions of one or more VMS' running on the same node). In addition, VMST:

- eliminates traffic between VMS', since all messages are now passed between threads inside the VMST process.
- supports interapplication communications between the MPS systems (the MPS system to which an application directs a message must be directly connected to the MPS running the application). Inter-VMST traffic is supported as described in [Interapplication/Host Service Daemon Data Exchange on page 215](#).
- supports automatic detection of lost TCP/IP connections (pinging)

The VMST process is located in `$MPSHOME/PERIase/bin` (Solaris) or `%MPSHOME%\bin` (Windows). When used with a single MPS, VMST is started by SRP through the `$MPSHOME/mpsN/etc/ase.cfg` file. When used with multiple MPS' (whether real or virtual), it is started through the `$MPSHOME/common/etc/gen.cfg` file. In addition to VENGINE and VAMP, VMST makes connections to the VSUPD processes



VMST is aliased as `vms` in its SRP startup files, but should not be confused with previous (“non-extended”) versions of VMS.

VSUPD

VSUPD is the ASE software process that is responsible for collecting application-specific statistics. VSUPD is a node-specific process, thus one instance of this process is required for each node regardless of the number of MPS components assigned to the node.



This process need not be run unless application statistics have to be collected and reported.

Each node collects statistics at 15-minute intervals for all applications executing on all MPS' on the node and stores them in the `ASEHOME/stats` directory. On systems with remote nodes, statistics for the four previous 15-minute periods are collected hourly from all other nodes by the one designated for MPS network statistical collection and reporting and transferred to that node's `ASEHOME/stats` directory. VSUPD supports an optional command line argument `-w <secs>`, which specifies the maximum amount of time to wait for phone line responses.

PeriReporter, in conjunction with the individual call processing applications, is used to define the statistical events to be collected and to create and generate reports. For information about PeriReporter, see the *PeriReporter User's Guide*.

VSUPD is started by SRP through the `$MPSHOME/common/etc/gen.cfg` file and located in `$MPSHOME/PERIase/bin` on Solaris systems and `%MPSHOME%\bin` on Windows systems. It makes its connections to VMST.

System statistics are collected by the VSTAT process on a per-MPS basis. For information about the VSTAT process, see [VSTAT on page 50](#).

ASE/VOS Integration Layer

This layer is used to convert and translate messages from the applications to the VOS processes. For PeriProducer applications, this layer communicates with the ASE processes, which in turn communicate with the applications themselves. The Vengine Application Management Process (VAMP) is an interface between the Application Services Environment (ASE) and the Voice Operating Software (VOS).

The VAMP services application requests:

- Consolidate information (lines, resources, etc.) for applications
- Consolidate information for commands issued by applications
- Control line bridging based on Call Progress Detection information
- Process resource control commands which may be directed to different resource providers and have different formats

VOS Processes

The Voice Operating Software (VOS) process group is comprised of the main system software required to run the MPS system. VOS processes can be common (only one instance required per node) or MPS-specific (one instance required per MPS component). This software group consist of the following independently running processes:

Process	Description
ALARMD (Alarm Daemon)	Collects alarm messages, writes them to the alarm log, and forwards them to any running alarm viewers.
CCM (Call Control Manager)	The primary interface between VAMP and the VOS services. Provides request synchronization and resource management.
COMMGR (Communications Manager)	Manages external host communications.
CONFIGD (Configuration Daemon)	System wide configuration process.
CONOUT (Console Output Process)	Relays output from VOS processes to the system console.
CONSOLED (Console Daemon)	Takes messages that would normally appear on the system console and displays them in the alarm viewers. (Solaris only.)
NCD (Network Interface Controller Daemon)	Controls interconnections between multiple TMS platforms attached to the NIC card.
nriod	Daemon responsible for remote input/output.

Process	Description
PMGR (Pool Manager)	Provides resource management, including resource allocation, resource deallocation, and keeping track of resource allocation statistics.
rpc.riod	Daemon responsible for remote input/output (Solaris backward compatibility only).
TCAD (TMS Configuration & Alarm Daemon)	Provides loading, configuration, and alarm functions for TMS.
TRIP (TMS Routing Interface Process)	Acts as a router between the VOS and TMS.
VMM (Voice Memory Manager)	Provides media management services for the VOS.
VSTAT (VPS Statistics Manager)	Provides system (as opposed to application) statistics consolidation and reporting.

ALARMD

ALARMD resides in the GEN subcomponent of the common component. It is responsible for collecting alarm messages, writing them to the alarm log, and forwarding alarms to the MPS alarm viewers. The alarm logs are located in the directory `$MPSHOME/common/log` in the format `alarm.<component_type>.<component_#>.log`, with backup files being appended with the `.bak` extension.

To avoid problems with memory exhaustion and the ALARMD daemon growing out of bounds, alarms can be suppressed from being logged to disk or being transmitted to the viewers (see [Alarm Filtering on page 203](#)). The daemon accepts commands either dynamically during run-time or statically from its configuration file during startup.

ALARMD associations:

- *Connections:* All processes which generate alarms
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* `$MPSHOME/common/etc/alarmd.cfg`
- *SRP Startup File:* `$MPSHOME/common/etc/gen.cfg`



The `alarmd.cfg` file only exists on systems where alarm filtering is instituted at startup (see [The `alarmd.cfg` and `alarmf.cfg` Files on page 99](#)).

The **alarm** command may be used to display the text of alarms, that are broadcast from ALARMD, in a command or VSH window. The PeriView Alarm Viewer is the GUI tool that may be used to select and display this same alarm information.

See the *PeriView Reference Manual* for additional information about the Alarm Viewer and the alarm information that may be obtained with this tool.

You can configure ALARMD to display the year in the timestamps that are added to entries written to the alarm log files (such as info, warning, alarm, and app). By default, the year is not displayed in the timestamp.

The optimum way to enable the display of the year in the timestamps of alarm log entries is to start ALARMD with the command line option **-y**. This can be done by modifying the COMMAND LINE field for ALARMD in the `$MPSHOME/common/etc/gen.cfg` configuration file to include the **-y** command line option. The entry for ALARMD in that file would appear as follows (note that the quotation marks are required):

```
alarmd - - 1 0 "alarmd -y"
```

An alternate method of enabling the display of the year in the timestamps of alarm log file entries is to add either of the following lines to the ALARMD configuration file `$MPSHOME/common/etc/alarmd.cfg`:

```
alarmd showyear on  
alarmd showyear 1
```

Displaying the year in the timestamps of alarm log file entries can be enabled or disabled after ALARMD starts by using VSH to issue the showyear console option with an appropriate argument to ALARMD.

For example, to enable the display of the year in the timestamps of alarm log file entries, issue either of the following commands at a vsh prompt:

```
alarmd showyear on  
alarmd showyear 1
```

To disable the display of the year in timestamps of alarm log file entries, issue either of the following commands at a vsh prompt:

```
alarmd showyear off  
alarmd showyear 0
```



If you want to display the year in the timestamps of alarm log file entries, Avaya recommends using the **-y** command line option in `$MPSHOME/common/etc.gen.cfg` to ensure that the year appears in the timestamp of every alarm written to the log file. If you use either of the other options described above, alarms generated early in the bootup sequence may not display the year in their timestamps.

For additional information about the **alarm** facility, see [System Utilities and Software on page 51](#). **alarm** is located in `$MPSHOME/bin` on Solaris systems or `%MPSHOME%\bin` on Windows systems.

CCM

CCM resides in the VOS subcomponent of the MPS component. Two CCM processes will exist in the VOS subcomponent: CCM and CCMA. CCM manages and controls phone lines and all resources required for interacting with the phone line (caller).

CCMA provides administrative services only, and does not provide phone line related services (i.e., outdial, call transfer, etc.). Configuration is accomplished in one of two ways: process wide or line/application specific. Process wide configuration is setup in `ccm_phoneline.cfg` (for CCM) or `ccm_admin.cfg` (for CCMA).

Line/application specific configuration is achieved by the application by setting up its required configuration when it binds with CCM/CCMA.

The CCM process is primarily responsible for:

- managing the phone line state dynamic
- allocating and deallocating internal and external resources, as well as administering the former
- command queue management and synchronization
- element name parsing for play, record and delete requests
- servicing audio play and record requests
- data input management (touch-tones, user edit sequences, etc)
- third party call control (conference management)
- maintaining call statistics

The CCMA process is primarily responsible for:

- command queue management and synchronization
- element name parsing for delete and element conversion requests
- MMF event reporting
- maintaining statistics

The VSH interface provides the ability to send commands to CCM. For a list of these commands, see the *CCM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

CCM associations:

- *Connections:* VAMP, NCD, TRIP, TCAD, VMM, PMGR
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration Files:*
 - *For CCM:* `$MPSHOME/mpsN/etc/ccm_phoneline.cfg`
 - *For CCMA:* `$MPSHOME/mpsN/etc/ccm_admin.cfg`
- *SRP Startup File:* `$MPSHOME/mpsN/etc/vos.cfg`

COMMGR

COMMGR resides in the VOS subcomponent of the MPS component and provides transaction processing services for the VOS. It enables application programs to communicate with external host computers using a variety of protocols. Though functionally equivalent to pre-existing versions, the release 1.0 COMMGR no longer requires that Virtual Terminals (VTs) be mapped to phone lines.

The `commgr.cfg` file defines the configuration parameters required to communicate with most external hosts. For more information, see [The `commgr.cfg` File on page 144](#).

Host communications functions and protocols are documented in the *Avaya Media Processing Server Series Communications Reference Manual*.

COMMGR associations:

- *Connections:* Protocol server processes
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* `$MPSHOME/mpsN/etc/commgr.cfg`
- *SRP Startup File:* `$MPSHOME/mpsN/etc/vos.cfg`

CONFIGD

CONFIGD is the system wide configuration process. It reads configuration files on behalf of a process and sends this configuration information to the process.



Online reconfiguration *must only take place when the system is idle* (no applications are attached). Unexpected behavior will result if the system is not idle during an online reconfiguration.

CONFIGD associations:

- *Connections:* All VOS processes
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* Not applicable
- *SRP Startup File:* `$MPSHOME/common/etc/gen.cfg`

CONOUT

CONOUT is the VOS process that is responsible for providing output to the system console. On Windows this provides output to the window in which SRP is started. It receives display data from the VOS processes and routes it to the system console.

CONOUT associations:

- *Connections:* Any VOS process sending info to the system console
- *Location:* \$MPSHOME/bin or %MPSHOME%\bin
- *Configuration File:* Not applicable
- *SRP Startup File:* \$MPSHOME/common/etc/gen.cfg

CONSOLED



CONSOLED takes messages that would normally appear on the system console and displays them in an alarm viewer. These messages include:

- system messages
- Zero Administration for Prompts (ZAP) synchronization status alarms

System messages can be generated by the MPS system or the operating system itself.

CONSOLED associations:

- *Connections:* Any process sending info to the system console
- *Location:* \$MPSHOME/bin
- *Configuration File:* Not applicable
- *SRP Startup File:* \$MPSHOME/common/etc/gen.cfg

NCD

NCD is comprised of three distinct logical entities: bridge control; Phase-Lock Loop (PLL) control; and a VSH interface to the NIC board itself. As part of the `tmscomm` component process group, one instance of NCD exists on a node containing TMS'. It interfaces with the TRIP and CCM processes in each VOS on the node, and with embedded processes running on the two chassis NICs (i.e., master, slave).

The NCD Bridge Control Process (NCD BCP) provides a common interface to support bridging between Resource Sets (RSETs) on or between TMS'. NCD BCP orchestrates the setup and teardown of the various bridging configurations supported by the TMS and NIC architecture. NCD BCP also has the ability to construct bridges between a pair of TMS' where the connections are physically hardwired (on a Hub-NIC card), or locked on a Time Space Switch (TSS) on the NIC.

The NCD PLL process provides configuration and control of the timing and clock sources on and between TMS' in a common chassis. NCD PLL is primarily used in small systems that do not have a NIC to provide these functions.

The NCD VSH interface provides the ability to send simple configuration commands to the NIC as well as query the current configuration. For a list of these commands, see the *NCD Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

NCD associations:

- *Connections:* TRIP (local and remote)
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* `$MPSHOME/common/etc/tms/tms.cfg`
- *SRP Startup File:* `$MPSHOME/tmscommN/etc/vos.cfg`

`nriod`

The `nriod` file provides information and access to MPS files for remote PeriView processes in both the Solaris and Windows environments. `nriod` is a system daemon and, as such, only one instance of this process is required for each node.

`nriod` associations:

- *Connections:* Any process communicating with the PeriView Task Scheduler
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* Not applicable
- *SRP Startup File:* `$MPSHOME/common/etc/gen.cfg`

PMGR

PMGR provides pooled resource management of all resources from Resource Provider (RP) processes running on the local node. An example of an RP is the CCM process, which provides lines as resources. An RP registers its resources with PMGR upon initialization. A registered resource can also be pooled applications (used for call handoff, for instance). As applications request resources, PMGR allocates the resources, keeps track of applications and their resources, maintains statistics, and deallocates resources as necessary.

If PMGR cannot allocate a resource locally, it forwards the request to a remote instance of PMGR; the specific instance is determined through round-robin availability. If there are no remote PMGRs available, the request fails. If PMGR dies, it releases all resources that have been allocated; if an RP dies, it must reconnect to PMGR to reregister its resources; if an application dies, allocated resources remain with it: after the application restarts, it queries PMGR for a list of resources currently allocated to the application; it may then use these resources or free them if no longer needed.

PMGR associations:

- *Connections:* Any process that provides resources (RP), applications
- *Location:* \$MPSHOME/bin or %MPSHOME%\bin
- *Configuration File:* \$MPSHOME/common/etc/pmgr.cfg
- *SRP Startup File:* \$MPSHOME/common/etc/gen.cfg

The VSH interface also provides the ability to send commands to PMGR. For a list of these commands, see the *PMGR Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

`rpc.riod`



The `rpc.riod` file provides information and access to MPS files for remote PeriView processes in the SPARC/Solaris environment. `rpc.riod` is a system daemon and, as such, only one instance of this process is required for each node.



This file is maintained for backward compatibility for systems running pre-5.4 software. [*nr iod* on page 45](#) is now included with the system to provide Solaris and Windows functionality.

`rpc.riod` associations:

- *Connections:* Any process communicating with the PeriView Task Scheduler
- *Location:* \$MPSHOME/bin
- *Configuration File:* Not applicable
- *SRP Startup File:* \$MPSHOME/common/etc/gen.cfg

TCAD

TCAD resides in the VOS subcomponent of the MPS component. It provides both

alarm and diagnostic services for the TMS hardware and loading and configuration services for the VOS. This includes:

- loading and configuration of all TMS devices
- a listing of TMS internal resources to the VOS
- alarm generation on behalf of TMS devices by translating TMS alarm code to the correct alarm format used by the alarm daemon (see [ALARMD](#) on page 40).
- diagnostics (System Performance Integrity Tests) which provide information about any device in the TMS. TCAD allows other processes to request information about any device (i.e., request telephony span status).
- logging capabilities for the hardware
- statistics and internal information about TMS devices

TCAD communicates with the TMS via TRIP. This includes sending loading and configuration messages through the Load Resource Management (LRM) port and sending and receiving alarm messages via the Alarm, Diagnostic, and Statistics Management (ADSM) port.

User interface with TCAD is via a VSH command line, which provides the ability to send commands to TCAD.

TCAD associations:

- *Connections:* TRIP, ALARMD, VMM, PMGR, and configuration files
- *Location:* \$MPSHOME/bin or %MPSHOME%\bin
- *Configuration File:* \$MPSHOME/mpsN/etc/tcad.cfg
- *SRP Startup File:* \$MPSHOME/mpsN/etc/vos.cfg

TRIP

TRIP resides in the VOS subcomponent of the MPS component. It is responsible for routing messages between the front end (VOS) and back end (TMS) over the TCP/IP connection. TRIP communicates directly with the LRM, ADSM, and Call SIMulator (SIM) ports of the TMS. TRIP is also responsible for providing the IP and port number of the TMS connected to a VOS. The calling process must identify the particular port on the TMS that it is interested in.

The VSH interface provides the ability to send commands to TRIP.

TRIP associations:

- *Connections:* CCM, VMM, TCAD, NCD, and the LRM, ADSM, and SIM ports of the TMS
- *Location:* \$MPSHOME/bin or %MPSHOME%\bin
- *Configuration File:* \$MPSHOME/mpsN/etc/trip.cfg
- *SRP Startup File:* \$MPSHOME/mpsN/etc/vos.cfg

VMM

VMM resides in the VOS subcomponent of the MPS component and provides media management services for the VOS. When VMM starts it connects to TCAD, TRIP and the VMEM port of the TMS. Once VMM detects that TCAD has configured the TMS, VMM loads the Voice Data Memory (VDM).

The startup time for VMM is minimal and does not delay speak/record requests unless the system is under heavy load. In the case of a record request under heavy load, the TMS buffers that data destined for VMM. Since input/output (I/O) blocking is performed, VMM is capable of servicing all other requests that arrive while prior I/O requests are awaiting completion, eliminating direct impact on other lines.

The VMM process is primarily responsible for:

- loading and managing VDM
- loading and managing media MMF files both system wide and application specific (playback and record)
- creating and managing hash tables of element names
- performing hash lookups on behalf of CCM
- performing on-line updates and deletes
- receiving data for ethernet based Caller Message Recording (CMR)
- maintaining maximum workload constraints and related queuing of pending I/O operations
- maintaining media access related statistics (reference counts and cache hits, for example)

The VSH interface provides the ability to send commands to VMM. For a list of these commands, see the *VMM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

VMM associations:

- *Connections:* CCM, TRIP, TCAD, and the VMEM port of the TMS
- *Location:* \$MPSHOME/bin or %MPSHOME%\bin
- *Configuration File:* \$MPSHOME/mpsN/etc/vmm.cfg
- \$MPSHOME/mpsN/etc/vmm-mmf.cfg
- *SRP Startup File:* \$MPSHOME/mpsN/etc/vos.cfg

VSTAT

VSTAT is the VOS software process that is responsible for collecting host, phone line and span system statistics. It resides in the VOS subcomponent of the MPS component.

Statistics are collected at each host node in 15-minute intervals and stored in the `MPSHOME/mpsN/stats` directory. Statistics for the four previous 15-minute periods are collected hourly by the node designated for MPS network statistical collection and reporting, converted to binary files, and moved to the `ASEHOME/stats` directory of that node. The same process occurs on single-node systems.

System statistics are collected by the VSTAT process and application statistics are collected by the VSUPD process. VSUPD is a member of the ASE software process group (see [VSUPD on page 38](#)). PeriReporter is used to create and generate reports based on these statistics. For information about PeriReporter, see the *PeriReporter User's Guide*.

•



VSTAT commands are intended to be issued by the Solaris `cron` or Windows scheduling facility and not at the VSH command line.

VSTAT associations:

- *Connections:* All processes which generate alarms
- *Location:* `$MPSHOME/bin` or `%MPSHOME%\bin`
- *Configuration File:* Not applicable
- *SRP Startup File:* `$MPSHOME/mpsN/etc/vos.cfg`

System Utilities and Software

In addition to the previously defined software processes, an array of system utilities and graphical tools is available to the MPS system operator and network administrator. These include:

Utility	Description
alarm	Textually displays alarms that were processed by the alarm daemon (see ALARMD on page 40). PeriView's Alarm Viewer may be used to display this same information in a GUI format.
dlog	Generic Debug-Logging. ¹ An interface that provides additional command options to multiple VOS processes.
dlt	Diagnostics, Logging, and Tracing (Daemon). Provides these capabilities for the TMS. ¹ Also used when executing call simulations (see Call Simulator Facility on page 195).
log	Textually displays low-level system process messages used for diagnostic purposes.
PeriProducer	Used to create and edit Avaya applications in a GUI environment.
PeriReporter	Collects, stores, and reports statistical data for the MPS network.
PeriStudio	Used to create and edit MMF files.
PeriView	A suite of GUI tools used to control and administer the MPS network. Included in this set of tools is: the PeriView Launcher, Application Manager, Activity Monitor, Alarm Viewer, File Transfer Tool ² , Task Scheduler ² , SPIN ¹ , PeriReporter Tools, PeriStudio, PeriProducer, PeriWWWord, PeriSQL, and PeriDoc.
PeriWeb	Used to create web-based applications and to extend typically IVR applications to the Internet.
vsh	Text command shell interface utility. Up to 256 VSH windows may be active at any one time.

1. Intended for use only by Certified Avaya Support Personnel

2. Not available at present on Windows.

alarm

alarm is the text-based utility used to display the alarms that are broadcast by ALARMD, the alarm daemon. **alarm** is a non-interactive application that simply displays the alarm message text received from the ALARMD process running on the MPS node with which **alarm** is currently associated. This translation facility uses the

alarm database to convert system and user-created messages to the proper format that may then be displayed and logged. If alarm filtering has been implemented through ALARMD, then **alarm** only receives those that pass the filter (ALARMF filtering has no affect on it since alarm “attaches” directly to ALARMD).

Alternatively, the *Alarm Viewer* may be used to display this same alarm information. The Alarm Viewer is a GUI tool accessible by means of the PeriView Launcher. Refer to the *PeriView Reference Manual* for additional information.

If the **alarm** process is unable to establish an IPC connection to ALARMD, it will periodically retry the connection until it succeeds. This functionality permits the Alarm Viewer to be invoked before starting the MPS system itself and allows for any startup messages to be viewed. Consequently, the Alarm Viewer for systems equipped with a graphics-capable console is invoked as part of the normal startup process providing for the automatic display of alarms (including normal startup messages) as they are generated during this period of time. See the *Avaya Media Processing Server Series System Operator’s Guide* for information on system startup and monitoring.

dlog

Debug logging is typically used by Certified Avaya Support Personnel. It is not frequently necessary to interact with **dlog** from an end-user’s perspective.

Although DLOG is not process-specific, a process name must be specified to invoke any of the commands. The processes that are configured to use DLOG options include CCM/CCMA, COMMGR, VAMP, PMGR, TCAD, TRIP, and VMM. The process name is substituted for the standalone **dlog** string in the command line options. The VSH interface provides the ability to interact with these processes. For a list of these commands, see the *DLOG Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

dlt

The DLT process provides:

- diagnostics (system performance integrity tests) which provide information about any device in the TMS. DLT allows other processes to request information about any device (i.e., request telephony span status)
- logging capabilities for the hardware (including line-based logging)
- statistics and internal information about TMS devices
- an interface for call simulation

DLT is used primarily by Certified Avaya Support Personnel and programmers. To initiate the DLT process, open a command window on the node you wish to monitor and enter the **dlt** command. Connections to TRIP and TCAD are attempted: if these connections are successful, the **dlt** prompt appears in the command line. For a list of these commands, see the *DLT Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

log

log is the text-based utility used to display messages sent between MPS processes. It monitors message traffic among selected VOS processes and is used for diagnostic purposes. This utility has a command line user interface.

log is an interactive application. It accepts commands from the terminal, maintains a history event list similar to that maintained by VSH (the MPS shell used for user interaction with VOS processes), and allows for simplified command entry and editing. For additional information refer to this manual's section about [vsh](#) on page 60.

log accepts the same command line options defined for any VOS process. These options may be used to determine the MPS with which **log** communicates and the method by which the messages are to be displayed. Further, a command line option may be used to determine the status of active logging requests when the **log** utility loses the IPC connection to the remote process responsible for implementing those logging requests. The utility is also able to log messages between processes that are not registered with SRP.

PeriProducer

PeriProducer is the software tool used to create, maintain, and test interactive applications for MPS systems in a GUI environment. It also provides a graphical application generation and testing environment that supports all aspects of an application's life cycle.

These applications are invoked by means of the Application Manager tool (APPMAN) accessible through PeriView. Generally, an MPS system runs multiple lines concurrently, and these lines are used to run different applications or multiple instances of the same application. For additional information about APPMAN see the *PeriView Reference Manual*.

The following is a list of the major functions that are available for processing caller transactions. An application can use some or all of these features:

- speaking to callers in recorded and/or synthesized speech
- accepting input from the caller using touch tone, speech recognition, or speech recording
- concurrently interfacing to multiple hosts
- processing information via computation
- accessing local files and databases
- sending or receiving a fax
- controlling phone lines
- processing exceptions
- recording caller messages

Generally, PeriProducer should be run on a separate development workstation. Should it be necessary to run it on a workstation that is also actively processing phone calls, PeriProducer should be used only during times of low system activity. Processing-intensive activities (e.g., testing logic paths, implementing resource usage, etc.) may impact the overall performance of the MPS system.

PeriProducer provides features that are used to verify the performance and functionality of an application either before or after it is placed into a production environment. While under development, application execution is accurately simulated within the PeriProducer software environment on the development workstation. A set of diagnostic functions allows the developer to view the internal workings of an application during the simulation.

When assigned to a line and started, the processing of an application is managed by the VOS VENGINE process (see [VENGINE on page 36](#)). VENGINE is also used while developing an application to execute all or part of the application so that the logic paths and application functions can be tested.

For additional information about using PeriProducer to create and maintain applications designed to execute in the MPS environment, refer to the *PeriProducer User's Guide*.

PeriReporter

PeriReporter is the tool used for collecting, storing, and reporting statistical data for the MPS network. It allows a point-and-click specification of multiple report formats for each statistics record type. A report is viewed as a set of columns, with each column representing an application or system-defined statistical counter. Each row of cells corresponds to a time interval recorded in a statistics file.

PeriReporter consists of three tools:

Tool Name	Description
PeriConsolidator	This program gathers all system and application statistics and consolidates them into 15 minute, hourly, daily, weekly, monthly and yearly files. PeriConsolidator is configured in the crontab ¹ and set to run at a convenient time once a day, preferably when the MPS system load is relatively light.
PeriDefiner	This program is a graphical utility which is used to set up the contents and the display of a specific report. After a report definition is created and saved it can be generated via the PeriReporter component of the tool.
PeriReporter	This program is a graphical utility which is used to generate reports. The report (created in PeriDefiner) must be specified, along with the date and the consolidation type, after which it can be generated and printed.

1. Functionality similar to **crontab** has been added to the Windows operating system through the Avaya software installation.

The PeriReporter tool typically resides only on the node that is designated as the site for statistical collection and reporting. Therefore, in a multi-node environment, the PeriReporter tool only displays and is available on the statistics node.

For more information on using PeriReporter Tools and configuring it for use in single and multi-node environments, see the *PeriReporter User's Guide*.

PeriStudio

PeriStudio is a software tool used to create, manage, and edit audio elements for MPS systems. Audio elements serve a variety of purposes in the voice processing environment, including providing verbal information, messages, voice recordings, touch-tones for phone line control, sound effects, music, etc. In the PeriStudio editor, audio elements may be initially recorded, as well as edited in any way germane to audio processing (e.g., volume levels, frequency range, duration of silent periods, etc.). Included with the tool is a GUI-based audio (MMF file) editor, file management and interchange facilities, and advanced audio signal processing capabilities.

Primarily, PeriStudio is used for:

- recording audio from a variety of sources (microphone, tape, line source, and other audio data format files).
- playing back recorded vocabulary elements for audible verification.
- editing all or portions of the recorded data (cut, paste, delete, scale length, etc.).
- importing and exporting audio items from or to other multimedia format files.
- performing advanced audio signal processing (equalization, normalization, mixing, filtering, etc.) of recorded elements to improve the sound quality.
- performing batch editing and processing on multiple elements in a single operation for obtaining consistent vocabularies as well as saving time.

Support is provided for both digital and analog environments, and digital and analog elements may be stored in the same multi-media (vocabulary) file. Audio files created in other software environments may also be imported into PeriStudio.

In order to provide a complete audio processing environment, an audio cassette tape player, an external speaker and a telephone handset are recommended. The cassette player is used to input recordings of speech to be digitized and processed for use on an MPS system. The telephone handset is used to verify the speech quality of audio elements as heard by system callers. The handset can also be used to record new speech elements directly to the editor. The external speaker is useful during editing and any subsequent audio processing operations to determine the effect of signal modifications made by the user.

Generally, PeriStudio should be run on a separate development workstation. Should it be necessary to run it on a workstation that is also actively processing phone calls, PeriStudio should be used only during times of low system activity. Processing-intensive activities (e.g., digitizing elements, adjusting their lengths, etc.) may impact the overall performance of the MPS system.

For additional information about using PeriStudio to create, edit and manage audio elements in the MPS environment, refer to the *PeriStudio User's Guide*.

PeriView

PeriView provides a suite of self-contained graphical tools used for MPS system administration, operation, and control. PeriView also provides access to several other distinct applications. Each tool is invoked independently and displays its own tool subset.

The Launcher is PeriView's main administrative tool. It provides a palette from which to select the various tools and applications. For a detailed description of PeriView and the use of its tool set, refer to the *PeriView Reference Manual*. For information on the daily activities typically conducted with PeriView, see the *Avaya Media Processing Server Series System Operator's Guide*.

Tool Name	Description
PeriView Launcher	The PeriView Launcher is used to define the MPS network's composite entities, to graphically portray its hierarchical tree structure, and to launch other PeriView tools.
Application Manager	The Application Manager (APPMAN) is used to associate applications with phone ports. Using APPMAN, you may invoke and terminate applications, associate and disassociate them from phone ports, configure application run-time environments and line start order, and access supporting application maintenance functions. MPS component and application status can also be elicited from this tool.
Activity Monitor	The Activity Monitor is used to monitor the states of phone line activity and linked applications within the network. Activity is depicted by a set of graphs in near real time. Host and span status may also be monitored from this tool.
Alarm Viewer	The Alarm Viewer is used to view live and logged alarms. A filtering mechanism provides for selectively displaying alarms based on specified criteria in the viewer. A logging facility provides for the creation of user-defined history-oriented Alarm Log Files.
File Transfer	The File Transfer tool is used to copy files across the MPS network. Transfer capability provides for movement of a single file, a group of files, or a subdirectory tree structure. <i>This tool is not available on the Windows operating system.</i>
Task Scheduler	The Task Scheduler tool provides a mechanism for defining and scheduling processes that are to be performed as either a single occurrence or on a recurrent basis. <i>This tool is not available on the Windows operating system.</i>

Tool Name	Description
SPIN	SPIN (System Performance and Integrity monitor) is a diagnostic tool used to monitor interprocess and intercard communications to facilitate the identification of potential problems on MPS systems. <i>SPIN is intended for use primarily by Certified Avaya Personnel.</i>
PeriReporter	PeriReporter provides statistics and reports management functions for the MPS network. It generates predefined reports and collects and reports user-defined application statistics. (For additional information, see PeriReporter on page 55.)
PeriStudio	PeriStudio is used on MPS and stand-alone workstations to develop and edit vocabulary and sound files for voice applications. (For additional information, see PeriStudio on page 56.)
PeriProducer	PeriProducer is used on Avaya Media Processing Server (MPS) Series and stand-alone workstations to create and support interactive applications. (For additional information, see PeriProducer on page 54.)
PeriWWWord	Use PeriWWWord, the PeriWeb HTML Dictionary Editor, to create and maintain dictionaries (directory structure containing the HTML fragments) of Words (HTML fragments) and their HTML definitions (HTML tags) for PeriWeb applications. <i>Available as part of PeriWeb (see below) on Solaris platforms only!</i>
PeriSQL	PeriSQL is used to create, modify, and execute Structured Query Language (SQL) SELECT commands through a graphical interface. PeriSQL can be used as a stand-alone utility or with the PeriProducer SQL block.

PeriWeb

PeriWeb is used to both build new applications to take advantage of the Web, and also extend existing IVR applications to the Internet user community. While IVR applications use the telephone as the primary input/output device, World Wide Web (WWW) browsers can provide an alternate visual interface for many types of transaction oriented applications. PeriWeb software facilitates this access mode with minimum changes. A user of a WWW browser initiates a “call” to an application by clicking a hypertext link. PeriWeb “answers” the call and routes it to the proper application. The application normally responds with a request to generate a greeting, but PeriWeb translates this into a dynamic hypertext document and sends it to the browser (caller). The user enters responses through forms or image maps, and PeriWeb delivers these responses back to the application.

Standard PeriPro IVR applications connect callers to MPS systems, where recorded voice prompts guide them to make service selections and enter information using touch tones or spoken words. The MPS responds to the caller using recorded prompts, generated speech, or fax output, as appropriate. For existing Avaya customers with IVR applications, PeriWeb software provides Internet access with minimal changes to the application programs. This leverages existing investment in application logic and host/database connectivity and transaction processing.

For customers with existing PeriProducer applications, PeriWeb adds:

- access to the World Wide Web
- an environment that does not require application logic changes for access to basic features (that is, IVR supported interactive transactions using a Web browser)
- enhanced Web presentation without changes to application logic and processing

In summary, its features allow PeriWeb to:

- co-exist with standard WWW servers, such as HTTPD, but does not rely upon them
- incorporate network-level security based on WWW encryption and authentication standards
- support standard HTML tagging formats created with a text editor or Web publishing tool
- perform Web transactions directly from the internet or through a relay server
- support the Keep-Alive feature of the HTTP1.1 protocol
- support the PUT method for publishing new HTML pages
- support standard and extended log file generation
- enable Web-aware applications for enhanced presentation on the World Wide Web using Web-oriented features
- support multiple languages for interaction content
- support Java based applications for browsers with Java capability

For information concerning PeriWeb details, see the *PeriWeb User's Guide*. For information on performing PeriPro IVR programming, see the *PeriProducer User's Guide*.

vsh

vsh is a text-based command shell which provides access to MPS processes. For both Windows and Solaris, **vsh** is modeled after the Solaris **cs**h in regard to input processing and history, variable, and command substitutions. **vsh** may be invoked from any command line. Up to 256 MPS shells may be in use at one time.

If only one component is configured in the `vpshosts` file for the node on which **vsh** is initiated, the default MPS shell prompt indicates the current component type and component number (that is, the component that is local to the node) as well as the node from which the tool was launched. If more than one component is configured for the node, a component list displays showing all components configured in the `vpshosts` file for that node, including those that are remote to the node (if any). Select a component by entering its corresponding number.



If **vsh** is invoked on an Speech Server node, the component list always displays first, regardless of the contents of the `vpshosts` file.

To display a list of components configured for a node, enter the **comp** command at any time. This command identifies the currently configured components along with their status. “Local” indicates the component is connected to the present node. “Remote” indicates the component is connected to another node in the network. Select a component by entering its corresponding number (“common” is not a selectable *component* entry).

```
cmdtool - /bin/csh
peri@tmsi03 {2} vsh
Configured components are:

   [1]   #common.0/tmsi03      <local/up>
   [2]   #common.0/dire09     <remote/up>
   [3]   #vps.3/tmsi03        <local/up>
   [4]   #vps.4/tmsi03        <local/up>
   [5]   #vps.1/dire09        <remote/up>
   [6]   #tmscomm.1/tmsi03    <local/up>
   [7]   #tmscomm.1/dire09    <remote/up>

Enter number (displayed in []) of component desired -> 4
Default component set to [#vps.4,vos/tmsi03]
vsh#vps.4,vos/tmsi03 {1} -> comp
Configured components are:

   [1]   #common.0/tmsi03      <local/up>
   [2]   #common.0/dire09     <remote/up>
   [3]   #vps.3/tmsi03        <local/up>
   [4]   #vps.4/tmsi03        <local/up>
   [5]   #vps.1/dire09        <remote/up>
   [6]   #tmscomm.1/tmsi03    <local/up>
   [7]   #tmscomm.1/dire09    <remote/up>

Enter number (displayed in []) of component desired -> 5
Default component changed to [#vps.1/dire09]
vsh#vps.1,vos/dire09 {1} -> hostname
tmsi03
```

vsh/comp Commands Example



Any native Solaris or Windows commands entered in an MPS shell are issued to the local node *regardless of the current component*. For example, if the current component is `mps1` and `dire09` is the name of the current node, but the MPS shell were launched on node `tmsi03`, `ls` lists the files in the directory on `tmsi03`, *not* on `dire09`. To identify the local node when connected to a component remote to that node, enter the `hostname` command at the prompt.

See [The *vpshosts* File on page 93](#) for information about this configuration file. See the *Avaya Media Processing Server Series System Operator's Guide* for additional information on command line interaction and control.

This page has been intentionally left blank.



2

Base System Configuration

This chapter covers:

1. **Base System Configuration**
2. **System Startup**
3. **User Configuration Files**
4. **The `MPSHOME` Directory**

Base System Configuration

The Avaya Media Processing Server (MPS) series system setup procedures involve installing and configuring the operating system and proprietary system software. The installation includes system facilities, and preconfigured `root`, administrative, and user accounts. The accounts are set up to run the operating system and define any required shell variables.

The software installation procedure creates the MPS Series home directory and places all files into the subdirectories under it. The `MPSHOME` variable is used to identify the home directory, and is set by default to `/opt/vps` for Solaris systems and `%MPSHOME%\PERIase` on Windows systems.

During system initialization, the various MPS processes reference configuration files for site-specific setup. Files that are common to all defined MPS systems are located in the directory path `$MPSHOME/common` (`%MPSHOME%\common`). Files that are specific to an MPS are located in their own directories under `$MPSHOME/mpsN` (`%MPSHOME%\mpsN`), where N indicates the particular numeric designation of the MPS. On Solaris systems, the files that comprise the software package release are stored in `$MPSHOME/packages` (symbolic links to these packages also exist directly under `MPSHOME` directory): on Windows systems, these files are stored in the MPS home directory. Not all packages and files exist on all systems: this chapter deals with those which are found in most basic MPS designs.





See the *Avaya Media Processing Server Series System Operator's Guide* for a more detailed discussion of the directory structure. See *Installing Avaya Solaris Software on the Avaya Media Processing Server Series* and *Installing Avaya Windows Software on the Avaya Media Processing Server Series* for matters regarding package installations.

System Startup

When started, the MPS software sets several system-wide parameters and commences the Startup and Recovery Process (SRP).

For information about configuration and administration files common to all MPS systems defined on a node, see [The *MPSHOME/common/etc* Directory on page 88](#). For information about component-specific configuration and administration files common to all MPS' defined on a node, see [The *MPSHOME/mpsN/etc* Directory on page 142](#). Information regarding TMS-specific processes can be found at [The *MPSHOME/tmscommN* Directory on page 138](#).

The startup files described in the following table are discussed further later in this chapter:

Startup File	Description
 s20vps.startup	Script that executes when the Solaris node boots. It is installed by the PERImps package. This script sets several Solaris Environment Variables and starts SRP (Startup and Recovery Process) (see page 70). This file is stored in the <code>/etc/rc3.d</code> directory. See Manually Starting and Stopping SRP on page 70 for more information about this script.
 s30peri.plic	Script that executes upon Solaris node bootup and starts the Avaya license server. Licenses are required for some Avaya packages to run. This file is installed by the PERIplic package in the <code>/etc/rc3.d</code> directory. For additional information on Avaya licensing and this file, see %MPSHOME%\PERIplic - /opt/vps/PERIplic on page 134 and the Avaya Packages Install Guides .
 vpsrc.sh vpsrc.csh	Defines MPS Solaris Environment Variables used by the Solaris shells <code>sh</code> and <code>csh</code> . These files perform the same function, but are specific to each shell type. The files are stored in the <code>/etc</code> directory.
 perirc.sh perirc.csh	<p>The <code>perirc.csh</code> and <code>perirc.sh</code> files resides in the <code>\$MPSHOME/PERI<package>/etc</code> directory. They contain the default environment variables that are common to the package.</p> <p>Do not edit these files! They are subject to software updates by Avaya. If a customer site must add to or modify environment variables, set the site-specific environment variables in the <code>siterc.csh</code> and <code>siterc.sh</code> files.</p> <p>The <code>vpsrc.csh</code> and <code>vpsrc.sh</code> files is responsible for executing the <code>perirc.csh</code> and <code>perirc.sh</code> files, which contain the environment variables specific to the products that are installed on a node.</p>



Startup File	Description
siterc.sh siterc.csh	<p>The <code>siterc.csh</code> and <code>siterc.sh</code> files are designed to contain site-specific environment variables. When these files exist on an MPS node, they reside in the following directory path: <code>\$MPSHOME/common/etc</code>.</p> <p>These files do not necessarily have to exist. Also, they can exist and be empty. If these files do not exist, you need to create them to enter site-specific environment variables. If they do exist, edit the file to include site-specific environment variables.</p> <p>The <code>vpsrc.csh</code> and <code>vpsrc.sh</code> files on the MPS node are responsible for executing the <code>siterc.csh</code> and <code>siterc.sh</code> files (if they exist). The values of the environment variables set in these files take precedence over the default values set in the <code>perirc.csh</code> and <code>perirc.sh</code> files.</p>
hosts	<p>Defines all systems associated with a particular MPS. The node names identified in all other configuration files must be included in this file. On Solaris systems, this file is stored in the <code>/etc</code> directory. On Windows systems, it is stored in the directory <code>\Winnt\System32\drivers\etc</code>. (See The <i>hosts</i> File on page 83.)</p>

Solaris Startup/Shutdown

When a Solaris system boots, it executes various scripts that bring the system up. The system software is started at *run level 3* by means of the `S20vps.startup` script file. The licensing mechanism is started by the `S30peri.plic` script, also at this level.

For a reboot, Avaya has altered the command to first perform a controlled shutdown, then bring the system up gracefully. A message displays that the original Solaris **reboot** command has been renamed to **reboot.orig**.



You can “flush” the memory on your system before rebooting by entering the `reset` command from the ROM prompt. This ensures there are no processes still in memory prior to the system coming back up.

The **halt** command has also been modified by Avaya to perform a controlled shutdown by taking down system processes and functions in the proper sequence and timing. If the **halt** command has been executed and the system does not respond, execute the **halt.orig** command instead.

The table that follows contains detailed Solaris and MPS startup and shutdown configuration information. For complete instructions on starting and stopping a node/software/system, see the *Avaya Media Processing Server Series System Operator's Guide*.

System Initialization and Run States

Scripts	Run Control Files	Run Level	Init State	Type	Use This Level	Functional Summary
/etc/rc0.d	/sbin/rc0	0	Power-down state	Power-down	To shut down the operating system so that it is safe to turn off power to the system.	Stops system services and daemons; terminates all running processes; unmounts all file systems
/etc/rc1.d	/sbin/rc1	1	Administrative state	Single user	To access all available file systems with user logins allowed.	Stops system services and daemons; terminates all running processes; unmounts all file systems; brings the system up in single-user mode
/etc/rc2.d	/sbin/rc2	2	Multuser state	Multuser	For normal operations. Multiple users can access the system and the entire file system. All daemons are running except for the NFS server daemons.	(Expanded functionality - see footnote that follows for details)
/etc/rc3.d	/sbin/rc3	3	Multuser with NFS resources shared and Peri software	Multuser	For normal operations with NFS resource-sharing available and to initiate any Avaya software startups.	Cleans up sharetab; starts nfsd; starts mountd; if the system is a boot server, starts applicable services; starts snmpd; (if PERIsnmp is not installed).
/etc/rc4.d	/sbin/rc4	4	Alternative multuser state	—	This level is currently unavailable.	—
/etc/rc5.d	/sbin/rc5	5	Power-down state	Power-down	To shut down the operating system so that it is safe to turn off power to the system. If possible, automatically turn off system power on systems that support this feature.	Runs the /etc/rc0.d/K* scripts to kill all active processes and unmount the file systems
/etc/rc6.d	/sbin/rc6	6	Reboot state	Reboot	To shut down the system to run level 0, and then reboot to multuser state (or whatever level is the default - normally 3 - in the inittab file).	Runs the /etc/rc0.d/K* scripts to kill all active processes and unmount the file systems
/etc/rcS.d	/sbin/rcS	S or s	Single-user state	Single-user	To run as a single user with all file systems mounted and accessible.	Establishes a minimal network; mounts /usr, if necessary; sets the system name; checks the root (/) and /usr file systems; mounts pseudo file systems (/proc and /dev/fd); rebuilds the device entries for reconfiguration boots; checks and mounts other file systems to be mounted in single-user mode

Mounts all local file systems; enables disk quotas if at least one file system was mounted with the quota option; saves editor temporary files in /usr/preserve; removes any files in the /tmp directory; configures system accounting and default router; sets NIS domain and ifconfig netmask; reboots the system from the installation media or a boot server if either /.PREINSTALL or /AUTOINSTALL exists; starts various daemons and services; mounts all NFS entries

Windows Startup/Shutdown

The Avaya Startup Service is installed with the PERIglobl package. During bootup, the services manager loads the Avaya Startup Service, along with other required subsystems.

The Avaya Startup Service reads a file name `vpsboot.cfg` from the system's `\winnt` directory. The format of the file is as follows:

- A '#' character introduces a comment until the end-of-line.
- Each line of text is considered to be a self-contained command line suitable for starting an application.
- The program being invoked must support the insert `@term@ -X <mutex_name>`, which is the *termination synchronization mutex*. The process polls this mutex, and when it is signaled, the process exits. The mutex is signaled when the service is stopped. Significant events are logged to the file `vpsboot.log` in the system's `\winnt` directory.



The following information is for use by Certified Avaya personnel only:

- If a service is stopped and started from the **Services** entry in the **Control Panel**, it again attempts to execute any commands listed in its configuration file.
- The command line option `show` (entered via the **Control Panel — Services**) allows the window associated with the started commands to be visible.
- The general mechanism for preventing Avaya software from starting at boot time is as follows:
 - Access administrative privileges
 - Choose **Control Panel — Services**.
 - Select **Avaya Startup Service** and click on the **Startup** button.
 - In the new popup, change the **radio box** setting from **Automatic** to **Manual**.
 - When the system is restarted, the Avaya software does not start. To restore automatic startup, follow the same procedure and restore the **Automatic** setting.

For Windows systems, the following services used in MPS operations are started at boot time. Each service is installed by the indicated package.

Service	Installation Package
Avaya Startup Service	PERIglobl
Avaya RSH Daemon	
NuTCracker Service	PERIgrs
Avaya License Service	PERIplc
Avaya VPS Resources SNMP Daemon	PERIsnmp
SNMP EMANATE Adapter for Windows	
SNMP EMANATE Master Agent	

Service	Installation Package
PeriWeb	PERIpweb

SRP (Startup and Recovery Process)

SRP (the Startup and Recovery Process) is the parent of all MPS software processes. It is responsible for starting and stopping most other software processes, and for polling them to ensure proper operation. It also restarts abnormally terminated programs.

One instance of SRP runs on each MPS node to control the systems associated with that node. As SRP finishes starting on each node, an informational alarm message is generated indicating that the system is running.

SRP has its own configuration file that provides for control of some internal functions. For information about this file, see [The `srp.cfg` File on page 89](#).

Each MPS node contains two classes of software processes, each of which has its own set of configuration files processed by SRP:

- The VOS (Voice Operating Software) process group is comprised of the core system software for running the MPS system (see [VOS Processes on page 39](#)).
- The ASE (Application Services Environment) process group is comprised of software to execute call processing and administrative applications (see [ASE Processes on page 36](#)).

In addition to controlling processes specific to each MPS system, SRP manages a common MPS (i.e., *virtual MPS*), which is used to run processes requiring only one instance per node. This includes system daemons, such as ALARMD.



Currently, the SRP is capable of starting pot approximately 300 applications.

Manually Starting and Stopping SRP

Normally, SRP is automatically started at boot time. If SRP has been stopped, it can be manually restarted.



If it is necessary to control the starting and stopping of SRP, it is first necessary to disable the operations of the `S20vps.startup` script. To do this, become `root` user and place an empty file with the name `novps` in the `$MPSHOME` directory. To manually start SRP on Solaris systems, execute the following command:

```
/etc/rc3.d/S20vps.startup start
```

Starts the MPS system software. This command can be used to restart SRP.

To shut down the MPS software, execute the following command:

```
/etc/rc3.d/S20vps.startup stop
```

Stops the MPS software without stopping the Solaris software.



Do not use the Solaris `kill` command to stop SRP!



To manually start SRP on Windows systems, follow the menu path Start—Settings—Control Panel—Services—Avaya Startup Service—Start.

To shut down the MPS software, follow the menu path Start—Settings—Control Panel—Services—Avaya Startup Service—Stop. You must have administrative permissions to perform these actions.



Do not use the Windows task manager to kill SRP!

VPS Topology Database Server (VTDB)

Many processes require information about available MPS systems and the processes running on each node. This information is collected via the VPS Topology Database (VTDB), which is used internally to store information about the MPS network.

The default well-known port used by other processes for SRP interaction on any node is 5999. The default port used by the VTDB library for SRP interaction is 5998. These default ports are intended to suit most configurations, and in most cases, these numbers should not be modified. To override these defaults, appropriate specifications must be made in the Solaris `/etc/services` or the `Winnt\system32\drivers\etc\services` file on Windows.



If changes are made to any port entries in these files, SRP must be stopped and restarted for the changes to take effect (see [Manually Starting and Stopping SRP on page 70](#)).

Restart of Abnormally Terminated Programs

SRP can restart programs that have either terminated abnormally or exhibited faulty operation. Abnormal termination is detected on Solaris systems via the SIGCHLD signal, or by proxy messages from remote copies of SRP that received a SIGCHLD signal. On Windows, a separate thread is started for each child process that SRP starts. This thread blocks on monitoring the process handle of the child process; when that handle becomes signalled by the kernel that the child process has terminated, the thread initiates the same child-termination processing that is instituted by SRP under the Solaris SIGCHLD signal handler. In either case, SRP restarts the process.

If the problem process were in the VOS software process group, a synchronization phase is entered. That is, all other processes in the VOS process group are notified that a process has terminated and should reset as if they were being started for the first time. SRP restarts the process that exited and all processes in the VOS software process group are allowed to begin operation.

Faulty operation is detected by means of the ping messages that SRP sends to processes in the VOS group. If successive ping messages fail to generate replies, SRP considers the process to be in an abnormal state and kills it. At that point, the system behaves as if the process exited abnormally.

Communication with VOS Processes



For Solaris-based systems, *multicast ping*ing is available as a subsystem within the IPC library. The implementation of multicast pinging is similar to that of *unicast* IPC-connection *pinging*, except that a ping transmission interval may be specified. All pinging configuration is done for SRP. VOS processes that receive pings cannot be configured for these actions. (This is handled within callbacks defined by the IPC library.)



For Windows systems, only unicast pinging is available.

In Solaris systems unicast or multicast pinging can be performed by any process whenever it is necessary to ping remote connections. The unicast method should be used when pinging a single remote connection, or a small number of remote connections. Multicast pinging should be employed if there is a need to ping many remote connections.

The following are the SRP configuration parameters used to configure multicast pinging:

Parameter	Description
Multicast Group IP	Internet Protocol address used for multicasting. The specified value must be in standard Internet dotted-decimal notation. It must be greater than or equal to 224.0.1.0 and less than or equal to 239.255.255.255. The IPC subsystem defines 225.0.0.1 as the default.
	SRP command line -x mpip=<dotted-decimal-IP>
	srp.cfg MPip=<dotted-decimal-IP>
Multicast Group port	IPC port used for multicasting. The specified value must be greater than or equal to 1025 and less than or equal to 65535. The IPC subsystem defines 5996 as the default.
	SRP command line -x mpport=<#>
	srp.cfg MPport=<#>
Multicast period	Time period between data transmissions. This value is specified in milliseconds and must be greater than the value given by the macro ITM_RESOLUTION_MS as defined in the ipcdefs.h file. (This value is set to 10.) The IPC subsystem defines 15000 as the default (i.e. a transmission period of 15 seconds).
	SRP command line -x mpperiod=<#>
	srp.cfg MPperiod=<#>
	VSH console option (to SRP) srp ipctimeout mping=<#>
	This method should only be used when pinging is not currently active (i.e., if SRP was started with either a -p or a -zp command line argument, or pinging was turned off via a -ping=off console option while SRP was running).
Maximum outstanding requests	Maximum number of unanswered ping requests to listener processes before the SRP server is notified of the fault. The specified value supplied must be greater than 0. The IPC subsystem defines 3 as the default.
	SRP command line -x mpmaxout=<#>
	srp.cfg MPmaxout=<#>

SRP Configuration Command Line Arguments

The SRP command line arguments are described below. Command line options for SRP are not typically used since it is started automatically on bootup. However, command line options *do* override options in the `MPSHOME/common/etc/srp.cfg` file.

```

srp          [-a] [-c] [-d] [-e] [-f <class>] [-g <#>]
               [-h] [-i <pri>] [-j <pri>] [-k <#>] [-l]
               [-n] [-p] [-q <#>] [-r <#>] [-s <#>]
               [-t <#>] [-u <#>] [-v <#>]
               [{-y|-z}[deklmprstTx]]
    
```

-a	Sets aseLines startup delay in seconds. Default is 3.
-c	Truncates the log file.
-d	Generates debugging output to the console. (This is the same as the -yd option.)
-e	Enables extended logging. (This is the same as the -ye option.)
-f <class>	Sets default VOS priority class. Currently not supported on Windows. Setting should not be changed on Solaris.
-g <#>	Size of the swap space low-water mark in megabytes.
-h	Displays command line options.
-i <pri>	Default ASE application priority. Currently not supported on Windows. Setting should not be changed on Solaris.
-j <pri>	SRP priority. Currently not supported on Windows. Setting should not be changed on Solaris.
-k <#>	Size of the swap space high-water mark in megabytes.
-l	Disables logging. (This is the same as the -zl option.)
-n	Disables restarting VOS processes after termination (This is the same as the -zn option.) This is primarily used for diagnostics and debugging.
-p	Disables pinging. (This is the same as the -zp option.)
-q <#>	Number of seconds for the runaway period. Default is 600.
-r <#>	Number of times that a process can restart (after exiting abnormally) within the runaway period set by the -q option. After the process has restarted the specified number of times within the given runaway period, no more restarts are attempted. Default is 3.
-s <#>	Log file size limit. The default maximum size is 5000000 bytes.
-t <#>	Proxy timeout. Times proxy messages, and determines the frequency of ping messages, between (remote) instances of SRP
-u <#>	Disk low-water mark, specified in megabytes.
-v <#>	Disk high-water mark, specified in megabytes.

```

srp          [-a] [-c] [-d] [-e] [-f <class>] [-g <#>]
               [-h] [-i <pri>] [-j <pri>] [-k <#>] [-l]
               [-n] [-p] [-q <#>] [-r <#>] [-s <#>]
               [-t <#>] [-u <#>] [-v <#>]
               [{-y|-z}[deklmprstTx]]

```

```

-y[deklmprstTx] Enables (-y) or disables (-z) the following functions:
-z[deklmprstTx]

```

```

d => debugging
e => extended logging
k => killAll protocol
l => logging
n => VOS process restarting
p => ping
r => registry debugging
s => state change logging
t => timestamping of external debugging (output of -d or
-yd)
T => extended timestamping wherever timestamping is performed
(i.e. through -yt, log file entries, or state change logging), and
where extended timestamping indicates milliseconds in addition to
the existing month/day/hour/minutes/seconds.
x => generating alarms for processes that exit normally

```

```

-y y          If you start SRP with the -y y option/argument pair, the
                timestamps of entries made into the srp.log and
                srp_state.log files will contain the year. If the year is enabled
                in the timestamp and the timestamp is enabled by the -yt
                option/argument pair, the year also appears in the timestamps that
                are added to debug output sent to the console and vsh.

```

You can permanently enable the year in the timestamp by doing one of the following:

- Add the following entry into `$MPSHOME/common/etc/srp.cfg`:

```
showYearInTimestamp=on
```

- Modify the line in the `/etc/rc3.d/S20vps.startup` file that starts SRP to add the **-y y** command line option. For example, change the line

```
cd ${VPSHOME}; srp >/dev/null 2>&1 &
```

to

```
cd ${VPSHOME}; srp -y y >/dev/null 2>&1 &
```

VSH Shell Commands

Once SRP is running, the VSH interface can be used to send commands that display status information or affect the current state of the system. To send commands to individual MPS systems, they must be sent through SRP.

To facilitate this, SRP supports a syntax construction that allows multiple commands

to be specified in a single entry intended for one or more MPS systems. Therefore, it is important that the particular component intended to receive a given command be clearly specified on the command line.

In general, the syntax of the command line takes the form of the name of the category for which the command is intended, followed by a pound symbol (#), the component type, a period, and the component number to which the command is being issued. For example, **vos#mps3** refers to the VOS software process group on MPS number 3. This information is preceded by the **srp** command and followed by an argument: thus, a complete command example based on the above is **srp vos#mps3 -status**.



The component IP address can be substituted for the node name (identifier) when issuing SRP commands.

The syntax and argument format for a VSH SRP command are shown below:

srp	obj -arg[=val] [obj -arg[=val] [obj ...]]
obj	An object (i.e. command destination) controlled by SRP, optionally specified with a component and node identifier. Any unrecognized command is compared to the process names in the applicable <code>vos.cfg</code> , <code>ase.cfg</code> , or <code>gen.cfg</code> file for a match. An object can be one of any of the following specifications:
componentX	Component. Includes (typically for MPS systems) <code>common</code> , <code>oscar</code> , <code>mps</code> , and <code>tmscomm</code> , or <code>compX</code> generically. X is a component specification: if not included, it is assumed that the component is the one on which <code>vsh</code> is logged in. A command issued with this object returns all instances of the argument applicable to the component only.
subcomponentX	Includes <code>vos</code> , <code>ase</code> , <code>gen</code> , and <code>hardware</code> . X is a component specification: if not included, it is assumed that the component is the one on which <code>vsh</code> is logged in. A command issued with this object returns all instances of the argument applicable to the subcomponent only.
component spec subset	A subset of a standard component specification in the general form <code><compType>.<comp#></code> , <code><subcompType>/<compIP></code> and where <code><compType></code> is any of the objects given in componentX , <code><comp#></code> is a component number, <code><subcompType></code> is any of those shown in subcomponentX , and <code><compIP></code> is a dotted decimal IP address.
subcomponent spec	A subcomponent specification in the general form <code><subcompType>.<comp#></code> , where <code><subcompType></code> is any of those shown in subcomponentX and <code><comp#></code> is an associated component number.

srp	obj -arg[=val] [obj -arg[=val] [obj ...]]	
obj	process	A subset of a full thread specification starting with a process name in the form of <pName> (<gName>) {<svcType> : <svcIDlst>}, and where <pName> is a VOS, ASE, or GEN process name, <gName> is a Group Name (intended to allow a process, such as a daemon, to segregate the processes that were connected to it, and treat a specific group of them in the same way), <svcType> is a Service Type (for example, CCM provides a service of managing phone lines, and its Service Type is SVCTYPE_PHONE [defined as "phone"]), and <svcIDlst> is an identifier or list of identifiers corresponding to <svcType> (in these instances, phone lines are associated with CCM, thus the <svcIDlst> would be any applicable phone line number, and the pairing would be, for example, {phone:1}).
	app	The set of lines associated with the applications bound to the current MPS. Except for "Line" commands, remainder affect all applications on system.
	none	Command is intended for SRP itself.
-arg[=val]	SRP arguments always begin with a dash ("-"), and arguments that take values must use the format -arg=val (rather than -arg val) because an arg specified without a dash prefix is interpreted as a new (unknown) command, and val not prefaced with an equal sign is also treated this way. The list of arguments that SRP recognizes for each of the command destinations is as follows. Note that if an argument is sent to a group object, it affects all lower-level objects belonging to the named object. For example, sending -kill to the vos object kills all VOS processes. The following arguments are available to all destination objects:	
	status	Displays current information about the named object. See SRP Status on page 81 .
	ping	Toggles the ping flag for the named object. Takes a value equal to a process name or, for app object, a line number.
The following argument is valid for all destination objects except for components which support the <code>hardware</code> subcomponent and which have a target of "hardware" (or, for legacy instances, "cps"):		

srp	obj -arg[=val] [obj -arg[=val] [obj ...]]	
-arg[=val]	kill	Kills the named object.
	The following arguments are valid for all destination objects <i>except</i> for components which support the <code>hardware</code> subcomponent and which have a target of "hardware" (or, for legacy instances, "cps"); where the target is SRP itself; or where no target is specified:	
	stop	Stops the specified object (no restart).
	start	Starts the specified object.
	The following argument is available <i>only</i> to the objects <code>mps</code> , <code>common</code> , and <code>comp</code> :	
	alarm	Causes SRP to generate a test alarm message to the alarm daemon, with the target object as the source component of the alarm.
	The following argument is valid for all destination objects <i>except</i> for components which support the <code>vos</code> subcomponent and which have a target of a VOS process; components which support the <code>ase</code> subcomponent and which have a target of an ASE process; and components which support the <code>gen</code> subcomponent and which have a target of a GEN process:	
	gstatus	Similar to the <code>status</code> command but displays information about the process groups as a whole instead of about individual processes. (See SRP Status on page 81.)
	The following argument is available to these destination objects only: <code>mps</code> , <code>common</code> , <code>comp</code> ; and components which support the <code>vos</code> subcomponent and which have a target of <code>vos</code> or a VOS process:	
	reboot	Completely shuts down the process or group and restarts it with commensurate reinitialization.
	The following argument is available to these destination objects only: components which support the <code>vos</code> subcomponent and which have a target of <code>vos</code> ; and components which support the <code>gen</code> subcomponent and which have a target of <code>gen</code> :	
	restart	Similar to performing the <code>stop</code> and <code>start</code> arguments.
	The following arguments are available only to the destination object of components which support the <code>ase</code> subcomponent and which have a target of <code>app</code> :	
-arg[=val]	startLine stopLine killLine	Starts/stops/kills application assigned to line specified by a value equal to its line number. Stopping an application puts it into an EXITED state: killing an application stops it then restarts it.

srp	obj -arg[=val] [obj -arg[=val] [obj ...]]
Examples:	srp vos#mps1 -kill Forcibly terminates all VOS processes on MPS number 1.
	srp vos#mps1 -status ase#mps2 -gstatus Sends the status command to the VOS software process group on MPS number 1 and the group status (gstatus) command to the ASE process group on MPS number 2. (See SRP Status on page 81 for sample output from the status commands.)
	srp app -killLine=111 Stops then restarts the application assigned to line 111 of the MPS associated with the VSH command line.

You can use a console option to enable displaying the year in the timestamps of entries made to the `srp.log` and `srp_state.log` files. To add the year, do one of the following:

- Add the following entry to the `$MPSHOME/common/etc/srp.cfg` file:

```
showYearInTimestamp=on.
```

- Issue the following command at a vsh prompt:

```
vsh {1} -> srp -showYearInTimestamp=on
```

If you want to disable displaying the year in the timestamp, issue the following command:

```
vsh {2} -> srp -showYearInTimestamp=off
```

To see a full list of options available to SRP, enter **srp -options** at a **vsh** command line.

Because unrecognized names are compared to the MPS and process names in the `vos.cfg`, `ase.cfg`, and `gen.cfg` files, SRP substitutes known values from the current **vsh** component. For example, if **vsh** is logged on to the `common` component on `tms2639`, the command **srp gen.0 -status** is the same as the command **srp gen#common.0/tms2639 -status**; thus, the former can be used as a shorthand version of SRP commands.

SRP Status

The following example of the SRP **status** command shows information from all MPS systems and components associated with node `tms1000`. The **gstatus** report produces a summarized version of the status report and includes any remote components defined for the node (in this case MPS number 1 on node `xtc9`).

```

cmdtool - /bin/csh

vsh.11@tms1000 {21} -> srp -status

NODE:PORT      USER      PID      LINE STATE      ENTERED STATE  FLAGS  CMDLINE
tms1000:5999   root      376      -   RUNNING    Apr 04 15:00:34  C  srp

Component: #common.0,gen/tms1000

tms1000:32791  root      377      -   RUNNING    Apr 04 14:56:19  C  alarmd
tms1000:32796  root      378      -   RUNNING    Apr 04 14:56:19  C  configd
tms1000:32793  root      379      -   RUNNING    Apr 04 14:56:19  C  conout
tms1000:32798  root      381      -   RUNNING    Apr 04 14:56:20  C  rpc.riod
tms1000:32798  root      382      -   RUNNING    Apr 04 14:56:20  C  nriond
tms1000:32800  root      383      -   RUNNING    Apr 04 14:56:20  C  consoled
tms1000:32871  root      384      -   RUNNING    Apr 04 14:56:21  C  pmgr

Component: #vps.11,vos/tms1000

tms1000:32853  root      401      -   RUNNING    Apr 04 14:58:29  C  trip
tms1000:32850  root      400      -   RUNNING    Apr 04 15:00:02  C  tcad
tms1000:32856  root      402      -   RUNNING    Apr 04 15:00:07  C  vmm
tms1000:32860  root      403      -   RUNNING    Apr 04 15:00:29  C  ccm
tms1000:32876  root      404      -   RUNNING    Apr 04 14:56:44  C  commgr
tms1000:32865  root      405      -   RUNNING    Apr 04 14:56:44  C  vstat

Component: #vps.11,ase/tms1000

tms1000        root      619      -   RUNNING    Apr 04 15:00:29  C  mxvmt
tms1000        root      620      -   RUNNING    Apr 04 15:00:29  C  vmst

Component: #vps.11,ase/tms1000

tms1000        peri      621      73  RUNNING    Apr 04 15:00:30  V
applicationone-release6-caller (applicationone-release6-caller)
tms1000        peri      632      169 RUNNING    Apr 04 15:00:32  V
applicationtwo-release6-receiver (applicationtwo-release6-receiver)

Component: #tmscomm.1,vos/tms1000

tms1000:32809  root      385      -   RUNNING    Apr 04 14:56:24  C  ncd

vsh.11@tms1000 {22} ->

```

```

cmdtool - /bin/csh

vsh.11@tms1000 {22} -> srp -gstatus

COMPONENT      GRP      PC      STATE      ENTERED STATE  MISC
#common.0/tms1000  srp      -   RUNNING    Apr 04 15:00:34  pid(376) pri(55)
#common.0/tms1000  common  -   RUNNING    Apr 04 14:56:20
#common.0/tms1000  vos      0  ABSENT     Apr 04 14:56:20
#common.0/tms1000  gen      6  RUNNING    Apr 04 14:56:20
#vps.11/tms1000   vps      -   RUNNING    Apr 04 15:00:34
#vps.11/tms1000   vos      6  RUNNING    Apr 04 15:00:29
#vps.11/tms1000   ase      2  RUNNING    Apr 04 15:00:34
#vps.11/tms1000   app      2  RUNNING    Apr 04 15:00:34  applications(2)
#vps.11/tms1000   gen      0  ABSENT     Apr 04 14:56:20
#tmscomm.1/tms1000 tmscomm -   RUNNING    Apr 04 14:56:24
#tmscomm.1/tms1000 vos      1  RUNNING    Apr 04 14:56:24
#tmscomm.1/tms1000 gen      0  ABSENT     Apr 04 14:56:20
#vps.1/xtc9       vps      -   REMOTE     Apr 04 14:56:20

vsh.11@tms1000 {23} ->

```

Call Control Manager (CCM/CCMA)

Startup parameters for CCM can be specified as command line options in the `MPSHOME/mpsN/etc/vos.cfg` file for the component CCM controls (see [The `vos.cfg` File on page 143](#)). These options apply to the current instance of CCM, and cannot be overridden directly from a command/shell line. If the parameters to CCM need to be changed, the system must be stopped, the `vos.cfg` file edited, and the system restarted. Configuration options available to CCM and CCMA are contained in [The `ccm_phoneline.cfg` File on page 151](#) and [The `ccm_admin.cfg` File on page 155](#), respectively.

The command line options for CCM are shown below:

ccm	[-c <class>] [-d <debug_obj>] [-s <num_list>]
-c <class>	Specifies whether CCM provides administrative (admin) or Telephony & Media Service (tms) services. The default for this option is tms .
-d <debug_obj>	Enables debugging from startup. All debugging is written to the default file <code>\$MPSHOME/common/log/ccm.dlog</code> . The following debug objects are supported: LINE, ERROR, STARTUP, ALL.
-s <num_list>	Specifies the service IDs/lines that CCM controls. This option is only required when the class is tms ; it is ignored for admin class. This option has no default.



The `-d` option should only be used to enable debugging of errors that happen before the system is up (i.e., before being able to enable debugging via `vsh`). The `-d` option is typically used for debugging administrative application bind issues in CCM.

Startup Files

The `hosts` File

The `hosts` file associates network host names with their IP addresses. Also, optionally, an alias can be included in these name-number definitions. The [first line](#) of the file contains the internal loopback address for pinging the local node. The [section that follows](#) this can be edited to add or delete other nodes recognized by the present one. You must be `root` user or have administrative privileges to edit the file.

The subsequent sections of the file contain [chassis numbering](#) and [LAN information](#). Each node contains entries for the hostname `vps-to-dtc`, `tmsN`, and `nicN` (where `N` denotes a specific TMS or NIC number). These "N" numbers are the only items that may be altered in this section of the `hosts` file. The IP addresses of these entries must not be edited by the user.



In this file, the term `dtc` is the same as TMS of release 1.X MPS terminology.

The [final section](#) of the file contains diagnostic PPP (Point-to-Point Protocol) communication addresses. The entries for `ppp-DialIn` and `ppp-DialOut` also must not be altered.

For Solaris systems, this file is stored in the `/etc` directory. For Windows systems, it is stored in `C:\Winnt\system32\drivers\etc`.

Example: `hosts`

```
127.0.0.1          localhost

# use www.nodomain.nocom line for systems not in a domain
# ctx servers to tms resource cards, private LAN
#
10.10.160.62      tms1000 loghost
10.10.160.42      is7502
10.10.160.3       vas1001
10.10.160.104    periblast
192.84.160.78    cowbird
192.84.161.17    pc105r
#
#192.168.101.200  scn1   scn1-to-tms   loghost www.nodomain.nocom
192.168.101.201  scn2   scn2-to-tms
192.168.101.202  scn3   scn3-to-tms
#
# If the VPS/is is connected to a network, change
# the above IP address and name as desired.
# When changing the VPS-is nodename, change all occurrences of
# VPS-is in this file. Remember to change update /etc/ethers also
#
#tms resource cards, private LAN
#
```

Example: hosts (Continued):

```
192.168.101.1    vps to dte ctx to dte
192.168.101.2    tms11
192.168.101.3    tms3
192.168.101.4    tms4
192.168.101.7    nic1
#
# IP Addresses associated with ctx chassis nbr 2
192.168.101.11   tms5
192.168.101.12   tms6
192.168.101.13   tms7
192.168.101.14   tms8
192.168.101.17   nic3
#
# IP Addresses associated with ctx chasis nbr 3
192.168.101.21   tms9
192.168.101.22   tms10
192.168.101.23   tms11
192.168.101.24   tms12
192.168.101.27   nic5
#
# IP Addresses associated with ctx chasis nbr 1 qfe ports
192.168.102.1    scn1qfe0
192.168.103.1    scn1qfe1
192.168.104.1    scn1qfe2
192.168.105.1    scn1qfe3
#
# IP Addresses associated with ctx chasis nbr 2 qfe ports
192.168.110.1    scn2qfe0
192.168.111.1    scn2qfe1
192.168.112.1    scn2qfe2
192.168.113.1    scn2qfe3
#
# IP Addresses associated with ctx chasis nbr 3 qfe ports
192.168.118.1    scn3qfe0
192.168.119.1    scn3qfe1
192.168.120.1    scn3qfe2
192.168.121.1    scn3qfe3
#
#
192.84.100.1     ppp-DialIn
192.84.100.2     ppp-DialOut
```

Entry	Description
localhost	Internal loopback address for pinging the same machine.
loghost	Local machine name (tms1000 in this example) precedes this entry, which in turn is preceded by its IP address.
vps-to-dtc tmsN nicN ppp-DialIn ppp-DialOut	Internal LAN designations. Do not edit these lines.
scnNqfeX	IP addresses associated with TMS chassis number N and QFE port numbers represented by X. Do not edit these lines.

User Configuration Files

The `.xhtrahostsrc` File

The `$HOME/.xhtrahostsrc` file lists the names of nodes where user access may be required. A node should be listed in this file if pertinent status information may be required of it, and the node is not already included in the `vpshosts` file. The `.xtrahostsrc` file identifies any nodes, other than those that are defined in the `vpshosts` file, which are to be displayed in the PeriView tree. An example of a node you may want to add to this file is a PeriView Workstation node. To implement this functionality, the `.xtrahostsrc` file needs to reside in the `$HOME` directory of the user that launched the PeriView tool (`$HOME/.xtrahostsrc`).

To display nodes in the tree that are not identified in the `vpshosts` file, create this file and place it in the user's home directory. Entries in this file must follow this format:

```
<node name><space or tab><yes or no>
```

One of the keywords `yes` or `no` must appear after each node name, following a space or tab. This indicates whether or not SRP is configured to run on the node. The state of the node displays in PeriView's tree only if SRP is configured as `yes`. Only one node is allowed per line.

The following is an example of this file:

Example: `.xhtrahostsrc`

```
$1  
#  
kiblet yes  
sheltimo yes  
frankie no
```

In this example, all three nodes will appear in PeriView's tree when so expanded, but only `kiblet` and `sheltimo` display their states. Node `frankie` always appears black (state unknown) because SRP is not configured to run there.



The first line in this file must contain only the string "`$1`". In some circumstances, this must be added manually.

For more information on this file and the states of nodes as displayed in PeriView, please see the *PeriView Reference Manual*.

The MP\$HOME Directory

The MPS system installation process creates a home directory and several subdirectories beneath it. On Solaris systems, this is represented as \$MP\$HOME (/opt/vps by default). On Windows systems, this is indicated as %MP\$HOME%.



See the *Avaya Packages Install Guides* and the *Avaya Media Processing Server Series System Operator's Guide* for more information about the home and subdirectories.

The relevant subdirectories (from a configuration standpoint) are identified in the following table, and described in greater detail later in this chapter.

MP\$HOME

Directory	Description
common	Contains files common to all MPS components associated with a particular node. (See The MP\$HOME/common Directory on page 88 for more information.)
packages	Contains the actual released software and sample configuration files. This directory is referenced by means of symbolic links in /opt/vps in the format PERIxxx (where xxx represents a package acronym). (See The \$MP\$HOME/packages Directory on page 125 for more information.)
PERIxxx	Individual packages of actual released software and configuration files. <i>These packages are located directly under %MP\$HOME%.</i> Use the <i>Table of Contents</i> to locate each package by name.
tmscommN	Contains files used for bridging between and within MPS components. (See The MP\$HOME/tmscommN Directory on page 138 for more information.)
mpsN	Contains files unique to each MPS, where N denotes the particular MPS number. One mpsN directory exists for each defined on the node with which it is associated. (See The MP\$HOME/mpsN Directory on page 139 for more information.)



On Solaris systems, if the defaults are not used, only the symbolic links to the Avaya packages exist in /opt/vps.



On Windows systems, if the defaults are not used, the specified target directory contains a Avaya subdirectory with the common and mpsN component directories, the distribution directory, and the bin executables directory.

The `MPSHOME/common` Directory

The `$MPSHOME/common` (`%MPSHOME%\common`) directory contains files common to all MPS components on a node. The subdirectories of relevance under `common` are described in the following table.

MPSHOME/common	
Directory	Contents
<code>etc</code>	Configuration, administration, and alarm database files. Contains a subdirectory structure of files that are generated from within PeriView and are common to all defined MPS components
<code>log</code>	Log files common to all defined MPS components. These files include <code>filexfer.log</code> , <code>sched.log</code> , <code>*.dlog</code> , <code>alarm*.log</code> , <code>srp.log</code> , and <code>srp_state.log</code> .

The `MPSHOME/common/etc` Directory

The `$MPSHOME/common/etc` (`%MPSHOME%\common\etc`) directory contains configuration and administration files common to all MPS components associated with the node. These files are used during system startup and are also responsible for ensuring the continual operation of the MPS system. This directory also contains the PeriView configuration and administration files.

These files are identified in the following table and further described in the passages that come afterward. Subdirectories used for the purpose of containing files generated by PeriView are also generic to the entire MPS system, and are described in the following table. For information about the files in these subdirectories, refer to the *PeriView Reference Manual*.

MPSHOME/common/etc	
File/Subdirectory	Description
<code>srp.cfg</code>	Defines the configuration parameters for the Startup and Recovery Process (SRP).
<code>vpshosts</code>	Lists all components known to the local node and the nodes to which those components are configured.
<code>compgroups</code>	Allows modification of default node for any process group listed in the <code>vpshosts</code> file.
<code>gen.cfg</code>	Lists ancillary Solaris processes started at boot time.
<code>global_users.cfg</code>	Lists the user names who have global view privileges in the PeriView GUI applications.
<code>alarmd.cfg</code>	Defines filter set files to be loaded and processed upon system startup for this daemon. If no such files exist, or they are not to be started automatically, then the <code>alarmd.cfg</code> file is not present.

MPSHOME/common/etc

File/Subdirectory	Description
<code>alarmf.cfg</code>	Defines filter set files to be loaded and processed upon system startup for this daemon. If no such files exist, or they are not to be started automatically, then the <code>alarmf.cfg</code> file is not present.
<code>pmgr.cfg</code>	Defines pools to which resources are allocated and configures resources that belong to that pool. Also enables/disables debug logging.
<code>periview.cfg</code>	Defines configuration parameters for Periview.
<code>/tms</code>	Contains configuration files copied over from the PERItms package. These include the <code><proto>.cfg</code> , <code>sys.cfg</code> , and <code>tms.cfg</code> files.
<code>/ents</code>	Contains the names of domains created by the Periview Launcher.
<code>/grps</code>	Contains the names of groups created by the Periview Launcher.
<code>/snps</code>	Contains the names of snapshots created by the Periview Launcher.
<code>/packages</code>	Contains the names of File Transfer Packages created by the Periview Launcher.
<code>/images</code>	Image files for Periview and its tools.

The `srp.cfg` File

SRP, the process that spawns all other processes in the MPS system, has its own configuration file, `srp.cfg`, which allows control of certain internal parameters. This file is stored in the `$MPSHOME/common/etc` directory for Solaris systems, and in the `%MPSHOME%\common\etc` directory on Windows-based systems.

As included in the system software, this file contains only comments that explain the syntax of the available parameters. If this file does not exist at the time of system startup, or if there are no actual commands, all parameters are assigned default values. Detailed descriptions of these parameters are provided in the table [SRP Configuration Variables](#) on page 90.



When a new `srp.cfg` file is installed, it does not overwrite an existing one. This allows modifications in the older file to be retained. The modifications, if any, must be manually added to the new file, and then the new file must be copied to the `common/etc` directory.

Example: srp.cfg

```
# Note that options in this file will be overridden by command line options to srp
#
# vosProcRestart      = 1 (default) - restart procs that terminate
#                    = 0 - do not restart procs that terminate
# vosKillAll          = 1 (default) - kill all procs if one terminates
#                    = 0 - use MT_RESTART protocol if a proc terminates
# vosFlushQueue       = 1 (default) - flush queues for VOS procs
#                    = 0 - do not flush queues for VOS procs
# alarmOnExit         = 1 procs that exit should generate an alarm
#                    = 0 (default) procs that exit should not generate an alarm
# maxLogSize          = maximum-size-of-log-file (bytes) (default=1000000)
# defAseAppPri        = default-ae-apps-priority (default=0)
# srpPri              = srps-priority (default=55)
# vosPriClass         = default-vos-process-priority (default=3)
# runawayLimit        = number-restats-allowed-in-runaway-period (default=3)
# runawayPeriod       = time-before-allow-more-SIGCHLDs (seconds) (default=600)
# proxyTimeout        = timeout-for-proxy-messages (seconds) (default=30)
# ping                = 1 (default) pinging on
#                    = 0 pinging off
# cdebug              = 1 debugging on
#                    = 0 debugging off
# log                 = 1 logging on
#                    = 0 logging off
# elog                = 1 extended logging on
#                    = 0 extended logging off
# swapLWM             = swap low water mark
# swapHWM             = swap high water mark
# diskLWM             = disk low water mark
# diskHWM             = disk high water mark
# statelog            = 1 (default) state logging on
#                    = 0 state logging off
# MPip                = multicast-group-IP (default set by IPC="225.0.0.1")
# MPport              = multicast-group-port (default set by IPC=5996)
# MPperiod             = multicast pinging period (default set by IPC=15000ms)
# MPmaxout             = maximum outstanding multicast ping responses (default=3)
# aseLineStartDelay   = delay between startup of last ASE process and first ASE LINES process
#                    (default=2s;specified in seconds)
# regdisp              = display format for "registry" and "lookup" commands
#                    = v (default) for a vertically-oriented listing
#                    = h (old style) for a horizontally-oriented listing
#
```

SRP Configuration Variables






Variable	Description
vosProcRestart	Enables or disables the automatic restarting of terminated VOS processes. If this parameter is set to 1 (the default), restarting is enabled. If it is set to 0, terminated processes are not restarted. This should be modified only by Certified Avaya personnel.

SRP Configuration Variables

Variable	Description
vosKillAll	Informs SRP whether it should invoke the normal restart synchronization method for subcomponent processes, or if it should kill and restart all VOS processes in the event that any one process dies. If this variable is set to 1 (the default), all processes are forced to terminate. If it is set to 0, RESET messages are used to synchronize VOS processes. Some software products (like MTS) need the RESET protocol instead.
vosFlushQueue	Sets IPC message queue flushing. This is the same as the IPC <code>-q</code> command line option. 0 means the queue does not get flushed. 1 (the default) allows flushing. This clears all transmit queues upon receipt of an MT_RESET message from SRP (used during group resynchronization when <code>vosKillAll</code> is not enabled).
alarmOnExit	Enables or disables alarm generation for processes (including applications) that exit normally. The default is 0 (alarms are not generated for normal terminations). 1 allows alarms to be generated.
maxLogSize	Specifies the maximum size (in bytes) of the SRP log files in bytes. The default size is 1MB.
defAseAppPri srpPri vosPriClass	Determines the usage of real-time priorities. Settings should not be changed.
runawayLimit runawayPeriod	Limits the number of times a process can exit abnormally within a specified period before further attempts to restart it are aborted. This is useful for avoiding infinite restarts to processes that can't run properly because external intervention is required (e.g., malfunctioning hardware, poorly made configuration files, etc.). The defaults are 3 times within 600 seconds (10 minutes).
proxyTimeout	Times proxy messages, and determines the frequency of ping messages, between (remote) instances of SRP. Default is 30 seconds.
ping	Enables or disables ping message exchange between SRP and other VOS processes. 1 (enabled) is the default; 0 disables this function.
cdebug	Enables or disables external logging (debugging). 1 (on) is the default; 0 (off) disables this function.
log	Enables or disables logging to the file <code>srp.log</code> . 1 (on) is the default; 0 (off) disables this function.
eelog	Enables or disables extended logging to the file <code>srp.log</code> . 0 (off) is the default; 1 (on) enables this function.



SRP Configuration Variables

Variable	Description
 swapLWM	Sets the swap space low watermark. When the current swap space resource use reaches the high watermark, SRP generates an alarm. If the swap space usage drops below this low watermark level, SRP generates another alarm. When an argument is supplied, it specifies the low watermark alarm threshold as a percentage.
swapHWM	Same as swapLWM, but for the high watermark.
diskLWM	Same as swapLWM, but for the current disk resource.
diskHWM	Same as diskLWM, but for the current disk resource's high watermark.
stateLog	Enables or disables state change logging for all SRP object state changes in the file <code>srp_state.log</code> . SRP object logging is enabled (1) by default; 0 disables state logging.
 MPip	Specifies the multicast group IP address. The value supplied must be in standard Internet dotted-decimal notation, and within the range 224.0.1.0 through 239.255.255.255, inclusively. The default is 225.0.0.1.
 MPport	Specifies the multicast group port number for IPC. The value supplied must be within the range 1025 through 65535, inclusively. The default is 5996.
 MPperiod	Specifies the multicast period, in milliseconds, between transmissions. This value must be greater than 10ms. The default is 15000, which provides a transmission period of 15 seconds.
 MPmaxout	Specifies the maximum number of outstanding ping responses from a listener process before the SRP server is notified of the fault. The value supplied must be greater than 0. The default value is 3.
aseLineStartDelay	The time, in seconds, between the final ASE process entering the RUNNING state and the spawning of the first ASELINE process as defined through the <code>aseLines.cfg</code> file (default is 2 seconds).
regdisp	Formats the output of the <code>registry</code> and <code>lookup</code> commands to be either horizontally (<code>h</code>) or vertically (<code>v</code>) displayed. The default is <code>v</code> (vertical).

The `vpshosts` File

After the `srp.cfg` file is read, the `vpshosts` file is processed. This file is stored in the `$MPSHOME/common/etc` directory for Solaris systems, and in the `%MPSHOME%\common\etc` directory on Windows systems.

The `vpshosts` file lists all components configured for the MPS network. Each component is identified by its component number, the name of the node where it resides, and the component's type. It is required that this file exist on each node in the network. Typically, the file's contents are the same across all nodes; however, this will vary in instances where additional component information is desired on a particular node.

The `vpshosts` file is created/updated on a node, automatically, during the system installation procedure. The file only needs to be edited to include components that have not been installed on the node and reside on other nodes in the network. By default, a node is only aware of those components (in the MPS network) that are explicitly defined in its `vpshosts` file. You must edit a node's `vpshosts` file to make the node aware of components that are installed on a different node.

A node name specified as a dash (-) implies the local node. For each component defined for the local node in the `vpshosts` file, a corresponding directory must exist in the `$MPSHOME` directory for Solaris systems, and in the `%MPSHOME%` directory on Windows systems. For example, if four MPS components are defined in the `vpshosts` file, the following subdirectories must exist: `$MPSHOME/mps1` (`%MPSHOME%\mps1`), `mps2`, `mps3`, and `mps4`. They may be renumbered, if desired. If the MPS components are renumbered, the node must be rebooted in order for the changes to take effect. The file also contains an entry for the `tmscomm` component.

The following is an example of the `vpshosts` file:

Example: `vpshosts`

```

$1
#
#vpshosts
#
#       This file was automatically generated by vhman.
#       Wed Apr 26 19:16:25 2000
#
# COMP   NODE       TYPE
110      -           mps
1        -           tmscomm
56       tms3003         mps

```



The first line in this file must contain only the string "`$1`". If this line is missing, it must be added manually.

The `vpshosts` file is copied over from the `MPSHOME/PERIglobl/etc` package and updated by means of the `vhman` command, issued from the command line. The `vhman` command can also be used to add or delete components from an existing `vpshosts` file. In general, there is no need to execute this command because the system comes preconfigured from the factory.

```
vhman          [-c <#>] [-t <type>] [-h <name>]
                [-H <name>] [-a | -d] [-q] [-n] [-f]
```

<code>-c <#></code>	Numeric designation of component.
<code>-t <type></code>	Type of component. Valid values include <code>mps</code> , <code>tmscomm</code> , <code>oscar</code> , <code>ctx</code> and <code>mts</code> .
<code>-h <name></code>	Host name associated with the component entry. A dash ("-") specifies the local host (which is the default).
<code>-H <name></code>	The host name of the <code>vpshosts</code> file to change. The default is to assume the local host. This option allows you to change a <code>vpshosts</code> file remote to the node <code>vhman</code> is being run from.
<code>-a</code>	Adds the specified component to the <code>vpshosts</code> file.
<code>-d</code>	Deletes the specified component from the <code>vpshosts</code> file.
<code>-q</code>	Quiet mode. In this mode, <code>vhman</code> does not display status or error messages.
<code>-n</code>	Disables display of the <code>vpshosts</code> column headings.
<code>-f</code>	Forces the current <code>vpshosts</code> file to be the latest version.



The `vhman` functionality can be executed in a GUI environment by using the `xvhman` tool.

PeriView needs to be configured with the information for all nodes that it is to control. This command would be issued on a PeriView node for the purpose of reconfiguring node names and component numbers. If the node configuration is changed, PeriView must be restarted.

For specific information about the `vpshosts` file (including editing and updating) and `xvhman`, refer to the *PeriView Reference Manual*.

The compgroups File

The `compgroups` file allows any of the groups (subcomponents) of any of the components listed in the `vpshosts` file to reside on a node different from the node hosting the component. If an entry in the `compgroups` file exists, it changes the meaning of the entry in the `vpshosts` file to the specified value. For example, if the `vpshosts` file has `mpsX` configured on `nodeY`, the `compgroups` file allows, for instance, the `vos` subcomponent of `mpsX` to reside on `nodeZ` instead of on `nodeY`. Otherwise, this file typically contains only descriptive comments. This functionality is rarely used.

During installation on a Solaris system, this file is copied over from the `$MPSHOME/PERIglobl/etc` directory. The file is stored in the `$MPSHOME/common/etc` directory for Solaris systems, and in the `%MPSHOME%\PERIglobl\etc` directory on Windows systems.

The following is an example of the `compgroups` file:

Example: compgroups

```
#
# Example compgroups file.
#
# Proc group can be one of VOS, ASE, HARDWARE, GEN.
#
# PROCGRP          ALTHOST

VOS                WWWW
ASE                XXXX
HARDWARE           YYYY
GEN                ZZZZ
```



A different default host can be specified for any process group. If an entry for a particular group is missing, or if the file itself is missing, the default meaning of "-" (local host) is used.

The gen.cfg File

The gen.cfg file lists ancillary software processes that are to be started upon system initialization. These are commands and custom software that SRP must monitor. Processes in this file are common to all components on a node and require only one instance to be present thereto. *If adding any additional (user-defined) processes be sure they meet this criteria.*

This file is stored in the \$MPSHOME/common/etc directory for Solaris systems, and in the %MPSHOME%\common\etc directory on Windows systems.

During installation, this file is copied over from the \$MPSHOME/PERIglobl/etc or %MPSHOME%\PERIglobl\commonec\etc directory. The file, as used by SRP, is read from common/etc.

The following is an example of this file on a Windows system. The Solaris version of the file follows immediately thereafter:



Example: gen.cfg

```
$3
#
# Example gen.cfg file.
#
# All executables listed in this file should support the
# Windows convention for srp-triggered termination. If you do not
# know what this means, please do not add any entries to this file.
#
# NAME          NODE    PORT    is-VOS-CLASS    PRI    COMMAND LINE
#
alarmd          -      -      1                0      alarmd
alarmf          -      -      1                0      alarmf
configd        -      -      1                0      configd
conout         -      -      1                0      conout
nriod          -      -      1                0      nriod
screendaemon   -      -      0                0      screendaemon
pmgr           -      -      1                0      pmgr
#vsupd         -      -      0                0      vsupd
#periweb       -      -      0                0      periweb
#proxy         -      -      0                0      "proxy -S ccss
                    -L cons -l info
                    -k 10 -n"
pbootpd        -      -      0                0      pbootpd
ptftpd         -      -      0                0      ptftpd
psched         -      -      0                0      "psched -run"
cclpd          -      -      1                0      cclpd
```


Example: gen.cfg

```

$3
#
# Example gen.cfg file.
#
# NAME          NODE      PORT      is-VOS-CLASS  PRI  COMMAND LINE
#
alarmd          -        -          1              0    alarmd
alarmf          -        -          1              0    alarmf
configd        -        -          1              0    configd
conout         -        -          1              0    conout
rpc.riod       -        -          0              0    rpc.riod
nriod         -        -          1              0    nriod
#screendaemon  -        -          0              0    screendaemon
consoled       -        -          0              0    consoled
pmgr           -        -          1              0    pmgr
#vsupd        -        -          0              0    vsupd
#periweb      -        -          0              0    periweb
#proxy        -        -          0              0    "proxy -S ccss
-L cons -l info -k 10 -n"

```

Field Name Description

NAME	Shorthand notation by which that process is known to SRP, vsh , and any other process that attempts to connect to it by name (essentially the process' well-known system name).
NODE	Node name the process is running on. A dash (-) indicates the local node.
PORT	Specifies the well-known port the process uses for IPC communication with other processes. If a dash is present, it indicates that the system fills in the port value at run time. A static port number only needs to be assigned for those processes that do not register with SRP, and must not conflict with the port numbers configured in the Solaris <code>/etc/services</code> file.
is-VOS-CLASS	Indicates whether or not the process uses IPC (1 is yes, 0 is no). By default, any processes listed in older versions of <code>gen.cfg</code> are classified as not using IPC (set to 0).
PRI	Real-time (RT) priority. This field is currently not used on Windows. A 0 indicates that the process should be run under the time-sharing priority class.

Field Name	Description
COMMAND LINE	Actual command line (binary) to be executed. Command line arguments can be specified if the command and all arguments are enclosed in quotes (see <code>proxy</code> in examples above). The normal shell backslash escape mode (" <code>\</code> ") may be used to embed quotes in the command line. A command with a path component with a leading slash is assumed to be a full path designation and SRP makes no other attempt to locate the program. If the command path doesn't begin with a slash, SRP uses the (system) <code>PATH</code> environment variable to locate the item. Avaya package installations add the various binary location paths to this environment variable during their executions.



The first line in a `gen.cfg` file must contain only the string "`$3`". In some circumstances, this must be added manually.



For Windows systems, only certified programs may be added to the `gen.cfg` file. Consult your system administrator before adding program names to this file.

The `global_users.cfg` File

The `global_users.cfg` file lists the users who have global view privileges in PeriView's APPMAN and Monitor tools. On Solaris systems, this file can only be modified by a user with `root` privileges. On Windows systems, this file should only be edited by users with administrative privileges.

This file is stored in the `$MPSHOME/common/etc` directory on Solaris systems, and in the `%MPSHOME%\common\etc` directory on Windows systems. The following is an example of this file:

Example: `global_users.cfg`

```
#
# global_users.cfg
#
# The user names in this file will have global view privileges.
#
# format:
#     globalUser=username
#
globalUser=peri
```

For specific information about PeriView and data views, refer to the *PeriView Reference Manual*.

The `alarmd.cfg` and `alarmf.cfg` Files

These files contain a reference to any filter set file that is to be instituted upon system startup. Filter sets are used to limit the types and number of alarms that are passed by the daemons for eventual display by the alarm viewers, or to initiate some other action in response to receiving alarms that satisfy certain criteria. For additional information, see [Alarm Filtering on page 203](#). The `addflt` command is used to enable a filter set file; the `clearflt` command to disable it. References in these configuration files must include the full path name to the filter set file unless it resides in the `MPSHOME/common/etc` subdirectory. In that case the name of the file itself is sufficient. In the example below, the filter set file `filter_set.flt` exists in `/home/peri`. Only one filter set file may be active at a time. This file must be created for and only exists on systems taking advantage of alarm filter sets at startup.

Example: `alarm*.cfg`

```
#  
addflt /home/peri/filter_set.flt
```



Filter sets, though standard ASCII files, should be appended with the `.flt` extension.

The pmgr.cfg File

This file sets parameters used by Avaya's Pool Manager process. The Pool Manager provides resource management of all registered resources on the local node (a registered resource can also be a pool of resources). During installation, this file is copied over from the \$MPSHOME/PERIglobl/etc or %MPSHOME%\PERIglobl\commonetc\etc directory.

Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. The following is an example of the default file installed with the system. See the table that follows for a more detailed explanation of each entry.

Example: pmgr.cfg

```
#
# Configuration file for PMGR process
#

#
# Enables debugging to a file
#
dlogDbgOn FILE,ERR
#dlogDbgOn FILE,GEN
.
.
.

#
# Defines a new pool called 'poolname'.
#
#defpool poolname
#

defpool line.in
defpool line.out

#
# Configures the resources that belongs in each pool
#

cfgrsrc line.in,phone.1-24.vps.*
cfgrsrc line.out,phone.25-48.vps.*
```



In theory any dlog command that supports the debug objects ERR and GEN can be entered in the configuration file. In practice, only those commands in the following table are. Though these commands are shown in this document prefaced with pmgr, the actual configuration file entry can be entered without the acronym.

Field Name	Description
dlogDbgOn	<p>Enables debugging to a file for errors only (ERR, the default) or all output (GEN) for this process. This file is located in <code>MPSHOME/common/log</code> as <code>pmgr.dlog</code> by default. The file name/location can be changed via the <code>pmgr dlogfile</code> command; the default size of 100k can be changed through the <code>pmgr dlogfilesize</code> command.</p> <p>Debug output can also be sent to a capture buffer, <i>but should not be sent to <code>STDERR</code>, which is intended for Certified Avaya personnel only.</i></p>
defpool	<p>A descriptive character string which identifies a particular pool of resources. This string must never start with the @ character due to an application programming conflict. The default values for this file are <code>line.in</code> (all inbound lines) and <code>line.out</code> (all outbound lines).</p>
cfgrsrc	<p>Defines the resources that make up each pool identified by the <code>defpool</code> entry above. The general format for a resource configuration is <code>cfgrsrc <poolname>, <resource_class>.<resource_instance>.<resource_comp_type>.<resource_comp_ID></code> (this last entry typically being a number). Wildcards (*) may be used for the resource instance and component ID. By default the <code>line.in</code> pool contains phone line numbers 1 through 24 on any MPS on the node; <code>line.out</code> maintains the same configuration for lines 25 through 48. These values should be adjusted to fit the number of lines, MPS', and expected call usage on each system.</p>

The only other command that might typically be set in this file is `pmgr allocRetry`. This configures the number of allocation retries that should be made before sending a failure back to the application if an allocation fails. The default value is 3.

For details on this and any other PMGR command, see the *PMGR Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

The periview.cfg File

The `periview.cfg` file defines configuration parameters for the PeriView Launcher, which is PeriView's main administrative tool. This file is stored in the `$MPSHOME/common/etc` (`%MPSHOME%\common\etc`) directory, and is not typically edited by the user. *Such editing may impede or surcease operation of the PeriView GUI.* For specific information about PeriView, refer to the *PeriView Reference Manual*. The following is an example of this file:

Example: periview.cfg

```
# This file contains the configuration information for the
# periview launcher pertaining to the applications it should
# launch, and the images and menu strings used to designate them.
#
# Format:
# Menu string      Image host      Image path/file      Command
# Send tree data  Send ipc data    Send view data      Send login data
#
# where 'Menu string' is a quote enclosed string for the
# launch menu.
# 'Image host' is the host where the image 'Image
# path/file' is located.
# '-' will indicate the current host.
# 'Image path/file' is the path and filename of the
# image. If no path is given, the path
# $MPSHOME/common/etc/images is assumed.
# The file must be in xpm format.
# 'Send tree data' is 'yes' if the tree's data should
# be sent to the application, and 'no'
# otherwise.
# 'Send ipc data' is 'yes' if the ipc timeout value
# should be sent to the application,
# and no otherwise.
# 'Send view data' is 'yes' if the view value should
# be sent to the application, and no
# otherwise.
# 'Send login data' is 'yes' if the login should be sent
# to the application, and no otherwise.
#
#-----
"Application Manager..." - appman.64xpm      appman      yes yes yes yes
"Activity Monitor..." - monitor.64xpm    monitor     yes yes yes yes
"Alarm Viewer..." - alarmview.64xpm  alarmview   no no no no
"File Transfer..." - filexfer.64xpm  filexfer    no no no no
"Task Scheduler..." - sched.64xpm     peri_schedule no no no no
"SPIN..." - spin.64xpm      spin       yes yes yes yes
"PeriReporter Tools..." - prpttools.64xpm PrptLaunch  no no no no
"Peri Studio..." - peristudio.64xpm peristudio  no no no no
"Peri Producer..." - pproi.64xpm     peripro     no no no no
"PeriWWWord..." - periwwword.64xpm pwwword     no no no no
"PeriSQL..." - perisql.64xpm   perisql     no no no no
"Online Documentation..." - onlinedoc.64xpm peridoc.bat no no no no
```



The Windows version of the `periview.cfg` file does not contain entries for "File Transfer", "Task Scheduler", or "PeriWWWord".

The MPShOME/common/etc/tms Directory

This directory contains the configuration files installed with and copied over from the PERItms package (see [%MPShOME%\PERItms - /opt/vps/PERItms](#) on page 134). Included are several protocol configuration files referenced by the `tms.cfg` file. These `<proto>.cfg` files are not detailed in this manual but instead can be found in the *Avaya Media Processing Server Series Telephony Reference Manual*. This directory is referenced by the system for the files to process during configuration.

The `sys.cfg` File

This file specifies parameters used by Avaya's Server Address Resolution Protocol (SARP). This protocol is used by software on MPS nodes to resolve internet addresses for connecting to TMS' and NICs.

A copy of the default `sys.cfg` file is maintained in the `MPShOME/PERItms/site-cfg` subdirectory. The system reads and processes the `sys.cfg` file located in the `MPShOME/common/etc/tms` subdirectory. Any customizing or changes should be made to this file: if it is necessary to revert to a "clean" version of the file, copy the `sys.cfg` file in the `/site-cfg` subdirectory to the `/tms` subdirectory, then proceed to make modifications as required.

Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. See the table that follows for a more detailed explanation of each.

Example: `sys.cfg` Sheet 1 of 2

```
#
# File for configuring Periphonic's server address resolution protocol (SARP)
# used by software on CTX nodes to resolve Internet Addresses for connecting
# to DTCs and Network Interface Cards (NIC's).
#
#
# Port Number
#
LRMPORT 30000
ADSMPORT 30001
SIMPOR 30002
VMEMPORT 30003
SARPPORT 30010
```

Example: sys.cfg Sheet 2 of 2

```
# EnetA broadCastIP
# Synopsis:
#     Specify the broadcast IP address of the network connected
#     from this host to ENET-A of the DTC's and NICs.
#     Default is 192.168.101.255
#
#     broadcastIP = broadcast IP address from
#
#
ENET-A 192.168.101.255

# iRepeat n
# Synopsis:
#     Specify the interval for repeating UDP SARP broadcasts
#     while in the initial period (iPeriod) occurring after
#     after a ctx node start up. After the initial period expires
#     broadcasts will be repeated at the repeat interval.
#     Default initial repeat is 10 seconds.
#
#     n = number of seconds between broadcasts
#
iRepeat 10

# iPeriod n
# Synopsis:
#     Specify the duration of the initial period which occurs after
#     ctx node start up. During this period UDP SARP broadcasts will
#     be repeated at the iRepeat rate on the networks listed above.
#     After expiration of the initial period, broadcasts will repeat
#     at the repeat interval. Default iPeriod is 600 seconds (10 minutes).
#
#     n = number of seconds of broadcasting every iRepeat seconds
#
iPeriod 600

# repeat n
# Synopsis:
#     Specify the interval for repeating UDP SARP broadcasts
#     after expiration of the initial period (iPeriod).
#     Default repeat interval is 60 seconds.
#
#     n = number of seconds between broadcasts
#
repeat 60
```



In this file, the term `dtc` is the same as TMS of release 1.X MPS terminology.

Field Name	Description
Port Number	Numbers assigned for the Load Resource Management (LRM), Alarm, Diagnostic, and Statistics Management (ADSM), Call SIMulator (SIM), Voice Memory (VMEM), and SARP ports.
EnetA broadCastIP	Specifies the broadcast IP address of the network connected from the host node to ENET-A of the TMS' and NICs. The default address is 192.168.101.255.
iRepeat n	Specifies the interval in n seconds for repeating UDP SARP broadcasts while in the initial period (iPeriod - see next). After the initial period expires broadcasts are iterated at the repeat interval. The default initial repeat is 10 seconds.
iPeriod n	Specifies the duration in n seconds of the initial period. This is the period of time which occurs after an MPS node start up. During this period UDP SARP broadcasts are repeated at the iRepeat rate (see above) on the networks listed at EnetA broadCastIP. After expiration of the initial period, broadcasts are iterated at the repeat interval (see next). The default iPeriod is 600 seconds (10 minutes).
repeat n	Specifies the duration in n seconds of the interval for repeating UDP SARP broadcasts after expiration of the initial period (iPeriod - see above). The default repeat interval is 60 seconds.

The `tms.cfg` File

The sections that follow, contain printouts from a sample `tms.cfg` divided into sections. They are presented in the same order of appearance as in `tms.cfg`. In addition to the configuration settings, the file contains narrative descriptions (comments) explaining the purpose of the configuration variables and settings in that section. This document provides more detailed explanations and references between sections to show relationship of entries throughout the file.

System Description Section

The system description section (`[SYSTEM]`) contains definitions for resource set profiles (`RSET_PROFILE`) and system parameters (`PARAM SYS_`).

The first uncommented line in the section contains the string `[SYSTEM]` to indicate the section. The uncommented lines that follow, contain the RSet profile definitions (one per line). Each RSet profile definition contains the string `RSET_PROFILE=` followed by the RSet profile name, the anchor resource class name, and additional resource class names, if any. The RSet profile name is also used in the [Resource Set Table Section on page 118](#). The *anchor* is the resource to which all other resources in the RSet are connected. The anchor will typically be a system device such as a phone line, however any resource can be the anchor. The class name (see [TMS Resource Definition Section on page 108](#)) is always followed by `:1` (additional numbers are reserved for future enhancement).

System Parameters

System parameters are used to override the default settings in the TMS. A comment is usually used to describe the effects of the parameter setting, followed by the uncommented line that assigns the value to the parameter. The statement line has the syntax `PARAM SYS_<parameter> = <value>`. The `PARAM` string is a keyword to indicate that a parameter is to be set. The `SYS` string indicates it is a system parameter (other types of parameters can be specified).

The system parameter `SYS_outdial_method` defines the resources the TMS uses when generating tones. Generated tones include DTMF, FAX, and Call Progress. The two options available are player `OUTDIAL_PLY` and tone generator `OUTDIAL_TGEN`. The default is `OUTDIAL_PLY`.

The system parameter `SYS_coding_law` is used to set the system (backplane) coding law. The default is `ulaw`. If `SYS_coding_law` has been changed from the default, it needs to be set to the network law (`ulaw` or `alaw`).

The system parameter `SYS_NETCODINGLAW` sets the network coding law. This parameter is used to override the default setting of the `ulaw` or `alaw` encoding on the DCC. By default, the DCC coding law is set based on the Phone Line Interface (PLI) on the TMS. For an E1 card, the default is `alaw` and for T1 card, the default is `ulaw`. To override the defaults, set `SYS_NETCODINGLAW` to either `ulaw` or `alaw` based

on the site requirements. If this parameter is set incorrectly, it can affect audio quality. For more information about `SYS_coding_law`, `SYS_NETCODINGLAW` and audio quality issues, refer to the section *MPS 2.1 Audio Click Prevention* in the document *Avaya Media Processing Server Series Speech Server 6.0.1 Reference Guide*.

System Description Section

```

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
; FILENAME :: $Id: tms-mps1000.cfg,v 1.1.2.7 2002/03/13 15:02:50 clnroom Exp $
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

;*****
;
; S Y S T E M   D E S C R I P T I O N   S E C T I O N
;
;*****
;
; This section specifies the system param definition area. It
; describes the resource set profiles and system parameters such
; as law to use for the MPS system
;
[SYSTEM]

; rset profiles are defined here and referenced in the line definition section
; defined below. These profiles specify how to build an rset and what resources
; are to be added.
;
; The command format is :
;
;   RSET_PROFILE = <RsetProfileName> default:1 <resource_classname:num_of_instances> ...
;
;
RSET_PROFILE = MPSLine   LINE:1 player:1 dtmf:1           ; Inbound rset profile

;
; System Parameters
;
; The following section contains system parameters. Any parameter defined here will
; override
; its hard coded default value in the TMS.
;
; Coding Law for the System.
;
PARAM SYS_coding_law = ulaw           ; define law of box

;
; Should outdialing try to use a player or a tone generator first.
;
PARAM SYS_outdial_method = OUTDIAL_PLY

```

TMS Resource Definition Section

The TMS resource definition section starts with resource configurations ([RSRC_CONFIG]). Each configuration is defined by a configuration name (CONFIG_NAME) and a set of resource classes to load ([CLASS]). The resource classes define the resources that will be loaded for the respective CONFIG_NAME. There can be multiple resource configurations. Each one must have a unique name and contain at least one class definition, but can contain more.

The first part of each resource configuration starts with an uncommented line containing the string [RSRC_CONFIG] to indicate the section. The next uncommented line contains the string CONFIG_NAME = <config_name>. The <config_name> value is an arbitrary name, but note that this name is also used in the [DTC Map Section](#) on page 112.

Each resource class definition starts with a line containing [CLASS], followed by separate lines containing COUNT = <number>, CLASS_NAME = <class_name>, and CDF = <cdf_file_name>.

Default settings for the class can be set under Default Params. The keyword PARAM is followed by the parameter and value.



System hardware limitations should be considered when configuring the COUNT value in the [CLASS] definition section. See [Resource Limitations](#) on page 111.

TMS Resource Definition Section

```
;*****  
;  
; T M S   R S R C   D E F I N I T I O N   S E C T I O N  
;  
;*****  
;  
; R S R C   C O N F I G U R A T I O N  
;  
; The following section defines the configurations that may be used  
; for a tms in this MPS system.  The configuration is referenced by  
; the configuration name.  
;  
; [RSRC_CONFIG]  
; CONFIG_NAME = BasicConfig  
;  
; This section is used to define the resources that should be loaded.  
; (players, recorders, asr, fax) for this configuration. This will  
; allow additional dtmf,cpd,tgen,r2 resources to be loaded as well  
; the ones specified in the proto.cfg files.  
;  
; This section specifies the configuration definition file (CDF) to use  
; and the class name (optional) to assign to the created line resources.  
; Count specifies the number of resources requested to be loaded  
; configuration. This number will be checked against the number of  
; licences available in order to load this system.  
;  
; If the class name is specified here it will override any class name  
; specified in the CDF file.  
;  
; Parameters specified here will override any parameters specified  
; in CDF file.  
;  
; Mode definition is done at this level. Each set of configuration parameters  
; specifies the values to set the paramters to for mode 0. This mode is used as the  
; default mode for a resource. When specified here it will override  
; the system defaults for the resources created.  
;  
; There may be more than one of class of resource loaded for this configuration  
; and there will be one of these class definitions for each type loaded.  
;  
; Each section will start with [CLASS]  
;  
;*****
```

TMS Resource Definition Section (Continued):

```
[RSRC_CONFIG]
CONFIG_NAME = BasicConfig

[CLASS]
COUNT = 210 ; number of DTMF resources to load
CLASS_NAME = dtmf; 210 / 30 per dsp = 7 Dsp
CDF = dtmf.cdf

[CLASS]
COUNT = 64 ; number of tone generators to load
CLASS_NAME = tgen; 64 / 32 per dsp = 2 Dsp
CDF = tgen_us.cdf

[CLASS]
COUNT = 60 ; number of CPD resource to load
CLASS_NAME = cpd ; 60 / 30 per dsp = 2 Dsp
CDF = cpd_us.cdf

[CLASS]
COUNT = 210 ; number of players to load
CLASS_NAME = player; 210 / 30 per dsp = 7 Dsp
CDF = okiply.cdf

[CLASS]
COUNT = 0 ; number of recorders to load
CLASS_NAME = oki_recorder; 0 / 20 per dsp = 0 Dsp
CDF = okirec.cdf

[CLASS]
COUNT = 0 ; number of recorders to load
CLASS_NAME = pcm_recorder; 0 / 15 per dsp = 0 Dsp
CDF = pcmrec.cdf

[CLASS]
COUNT = 0 ; number of recorders to load
CLASS_NAME = pcm_fullldup_rec; 0 / 15 per dsp = 0 Dsp
CDF = pcmrec2.cdf

[CLASS]
COUNT = 0 ; number of recorders to load
CLASS_NAME = oki_fullldup_rec; 0 / 15 per dsp = 0 Dsp
CDF = okirec2.cdf

[CLASS]
COUNT = 0 ; number of conference PORTS to load
CLASS_NAME = conference;
CDF = conf.cdf ; 0 / 16 ports per dsp = 0 Dsp

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Resource Limitations

There are hardware limitations to the classes of resources and the quantity (count) that can be loaded, based on the number of Digital Signal Processors (DSP) in the system. The TMS motherboard contains six DSPs and each MDM installed contains 12 additional DSPs. The limitations are generally not a factor, however, they need to be considered when configuring the system. Configuring unnecessary resource classes and counts can degrade system performance and occupy DSPs that are needed for other resources.

Consider all sources of resource class configuration. In addition to the common `tms.cfg` file, resources can be loaded as part of a phone line protocol. (See [Protocol Configuration Files](#) on page 123.) The `<proto>.cfg` file contains the same [CLASS] definition section as the `tms.cfg` file.

Protocols are assigned on a per span basis in the protocol package definitions section. (See [Line Card Protocol Package Definitions](#) on page 114.) The number entered in the COUNT statement in the [CLASS] definition section means that number of resources will be loaded for each span the protocol is assigned to, in addition to those defined in the `tms.cfg` file.

Example:

- A COUNT of 30 `tgen` resource classes is entered in the `tms.cfg` file.
- A COUNT of 24 `tgen` resource classes is entered in the `att_winkstart_proto.cfg` file.
- The `att_winkstart_proto.cfg` is assigned to four spans in the protocol package definition section.

A total count of 126 tone generator resources will be loaded.

The following table lists the available protocols and the resource classes that comprise the protocol.

Protocol	Configuration File (*_proto.cfg)	Resource Classes						
		r1tx	r1rx	dtmf	tgen	ply	cpd	r2
ATT Winkstart	<code>att_winkstart_proto.cfg</code>	no	no	yes ¹	yes ²	no ²	no ³	no
Feature Group D	<code>fgd_eana_proto.cfg</code>	yes	yes	no	no	no	no ³	no
CB Ground Start	<code>cb_grndstart_proto.cfg</code>	no	no	no	yes ²	no ²	yes	no
CB Loop Start	<code>cb_loopstart_proto.cfg</code>	(TBD)						
Net5 ISDN	<code>isdn_net5_proto.cfg</code>	no	no	no	yes ²	no ²	no ³	no
National ISDN	<code>isdn_national_proto.cfg</code>	no	no	no	yes ²	no ²	no ³	no
R2 Saudi	<code>r2_saudi_proto.cfg</code>	no	no	no	yes ²	no ²	no ³	yes

1. The `dtmf` resource is required only if DNIS collection is enabled.

2. Either a `tgen` or a `ply` resource may be used for outdialing and generating call progress.

3. The `cpd` resource is optional.

The following table lists the quantity of each resource that can be loaded per DSP. The Class Name column contains the exact string that should be entered in the CLASS_NAME statement. The Configuration Definition File column contains the name that should be entered in the CDF line. The Count/DSP is the number of that resource a DSP can provide. Each resource loaded occupies a DSP, whether there is only one instance, or up to the limit a DSP can handle. If the count exceeds the limit by only one, another DSP will be loaded to handle the instance, and that DSP will not be available for other resources.

Class Name	Configuration Definition File (*.cdf)	Count/DSP
dtmf	dtmfrx.cdf	30
tgen	tgen_us.cdf	32
tgen	tgen_uk.cdf	32
r1_mf_rx	mfr1rx.cdf	30
r1_mf_tx	mfr1tx.cdf	32
oki_player	okiply.cdf	30
oki_recorder	okirec.cdf	20
oki_fulldup_rec	okirec2.cdf	16
pcm_player	pcmply.cdf	30
pcm_recorder	pcmrec.cdf	30
pcm_fulldup_rec	pcmrec2.cdf	16
cpd	cpd_us.cdf	30
cpd	cpd_uk.cdf	30
r2	r2.cdf	12

In the preceding *Example*, 126 tgen resources are loaded by the configuration. Since a DSP can provide up to 32 tone generators, four DSPs are occupied by tgen as a result of those configuration entries.

DTC Map Section



The term DTC stands for Digital Trunk Controller. It is synonymous with TMS.

The MPS relative configuration begins with the DTC map section. The DTC map section ([DTCMAP]) is used to define the physical location of each TMS in the MPS by its chassis and backplane slot (BPS) position, and the primary and secondary VOS subcomponents to which they are assigned ([BIND]).



On the back of each VRC, there is a number selector that defines the number of the chassis. (See [VRC Rear Panel on page 26](#).) Ensure that these are uniquely set (starting at 0) for each chassis in the system. This number corresponds to the chassis number to be used in the `[DTCMAP]` section of the `tms.cfg` file.

A logical TMS number (TMS1, TMS2, etc.) is assigned to each TMS in the system. Each TMS must have a primary VOS subcomponent bound to it. Typically, TMS1 is bound to VOS1, TMS2 is bound to VOS2, and so on. If a redundant or backup MPS node is used in the system, the MPS components on that node are also aliased to secondary VOSs. Typically TMS1 is bound to VOS101, TMS2 is bound to VOS102, and so on.

Under the `Config` column, is the configuration name (`[CONFIG_NAME]`) for each TMS. This defines the configuration definition to use for each TMS. (See [TMS Resource Definition Section on page 108](#).)

There should always be uncommented `BIND` statements for each NIC in the chassis. Only the `Chassis Num` and `Chassis Slot` (i.e., 7 and 8) should be entered, with the remaining columns each containing a dash (-). If the chassis contains a Hub-NIC, there is no need for NIC bind statements, or they can be commented out.

DTC Map Section

```

;*****
;*****
; MPS relative configuration starts here
;*****
;*****
;
; This section will defines the TMS's in the MPS system.
; It assigns each TMS to a primary and secondary controlling MPS.
; The bound MPS will load and configure the associated
; TMS as a result of the following bind commands.
; TMS number must be from 1 to max TMS number.
;
;*****
[DTCMAP]
;-----
;      Chassis   Backplane   TMS      Primary   Secondary
;      Num       Slot (BPS)   Num      VOS Comp# VOS Comp#  Config
;-----
BIND  1          1           1         1         -         BasicConfig
BIND  1          2           2         2         -         BasicConfig
BIND  1          3           3         3         -         BasicConfig
BIND  1          4           4         4         -         BasicConfig

BIND  1          7           -         -         -         -
BIND  1          8           -         -         -         -

BIND  2          1           5         5         -         BasicConfig
BIND  2          2           6         6         -         BasicConfig
BIND  2          3           7         7         -         BasicConfig
BIND  2          4           8         8         -         BasicConfig

BIND  2          7           -         -         -         -
BIND  2          8           -         -         -         -
;

```

Line Card Protocol Package Definitions

A protocol must be specified for each span in the system. One LOAD statement is use to specify the protocol for each span, and spans cannot be split (i.e., protocols cannot be specified for individual lines). The LOAD statements are entered in tabular format under the commented headings. The fields are described below.

- The TMS Num field contains the unique TMS number specified under the *DTC Map Section* on page 112.
- The PLI Slot is the slot containing the phone line card (DCC or ALI).
- The Span Num is the span this LOAD statement is assigning the protocol to.
- The svc-type field is required and the string is ISDN, SS7, or CAS.
- The MpsNum, Outline, and Pool/Class fields are used for configuration of legacy systems that are not otherwise configured to use the

Pool Manager. (See *Pool Manager (PMGR)* on page 288.) These fields shall each contain a dash (-) for all spans on the MPS.

- The Protocol Pkg field contains the name of the protocol configuration file (*_proto.cfg).

Line Card Protocol Package Definitions

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This section is used to define the protocol packages to load
; to the line cards in this MPS system. For DCC cards there is a
; protocol package specified for span of the DCC card.
;
; TMS num - tms number (from [DTCMAP] section above)
;
; PLI slot - slot line card is plugged into on the TMS
;
; Span Num - this is the span number for a DCC card. For an
; analog card this is not applicable and a dsh will appear there
;
; Service type - ascii string that is returned to app in
; responses to GetInCall and Get OutLine containers
;
; mpsNum - the Vps number, if applicable, that these lines are
; plugged into to and the associated lines.
;
; Outline - The user specifies the lines of the span/card that are
; outbound lines via the following specification:
;
;           [<s>-<e>] | * |
; where:
;           s = start line number
;           e = end line number
;           * = all lines
;           - = no lines
;
; The above specification references lines s to e (inclusive) relative
; to span span_num.
;
; The user may specify that all lines for a particular span be placed
; in a pool by use of *.
;
; If the card is an ALI card the span number will be "-"
;
; Pool/class - the class name to use when creating the resource
; pool.
;
; Protocol package is file having TMS resources needed to support
; the requested protocol.
;

```

Line Card Protocol Package Definitions (Continued):

;	TMS	PLI	Span	svc-type	MpsNum	Outline	Pool/class	Protocol
;	Num	Slot	Num					Pkg

LOAD	1	4	1	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	2	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	3	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	4	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	5	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	6	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	7	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	1	4	8	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	1	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	2	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	3	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	4	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	5	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	6	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	7	CAS	-	-	-	att_winkstart_proto.cfg
LOAD	2	4	8	CAS	-	-	-	att_winkstart_proto.cfg

MPS Line Definition Section

The [VPS_LINE_DEF] section is used to map physical TMS lines to logical line numbers. Generally, one LINE statement is used to map each physical span in the system. The LINE statements are entered in tabular format under the commented headings. The fields are described below.

- The MPS from:to field contains the range of logical line numbers mapped to the lines of the physical span.
- The TMS Num field contains the unique logical TMS number defined under [DTC Map Section on page 112](#).
- The PLI Slot Num field contains the TMS slot number (1-4) of the line card (DCC or ALI) where the physical span resides.
- The Span:channel field contains the span number on the DCC, a colon (:), and the starting channel number (always 1).

MPS Line Definition Section

```

;
; M P S   L I N E   D E F I N I T I O N   S E C T I O N
;           R s e t   C r e a t i o n
;
; This section maps the controlling VPS's lines to the physical
; lines on the associated TMS.  This causes the creation of
; rsets - one for each line mapped.  The entries are specified as follows:
;
; lines : These are the VPS line numbers that are to be
;         mapped.
; TMS Num : This is the TMS number.
; PLI Slot Num : This is the slot on the TMS which references
;              the card.
; Span   : This is the span number for a DCC card.  If
;         an analog card then this is not applicable
; channel : This is the start channel (instance) for the
;         mapping.

[VPS_LINE_DEF]
;-----
;      MPS      TMS   PLI Slot  Span:channel
;      from:to  Num   Num
;-----
;
;           MAP TMS 1 lines
;
;
LINE   1:24     1     4           1:1
LINE   25:48    1     4           2:1
LINE   49:72    1     4           3:1
LINE   73:96    1     4           4:1
LINE   97:120   1     4           5:1
LINE  121:144   1     4           6:1
LINE  145:168   1     4           7:1
LINE  169:192   1     4           8:1

LINE   1:24     2     4           1:1
LINE   25:48    2     4           2:1
LINE   49:72    2     4           3:1
LINE   73:96    2     4           4:1
LINE   97:120   2     4           5:1
LINE  121:144   2     4           6:1
LINE  145:168   2     4           7:1
LINE  169:192   2     4           8:1

```

Resource Set Table Section

The [RSET_TABLE] section is used to create custom resource sets for individual lines or a range of lines. If this section is defined, it will use the specified RSET_PROFILE for creating the RSet for that line or range of lines. The RSET statements are entered in tabular format under the commented headings. The fields are described below.

- The MPS Line Num contains the range of logical line numbers that were mapped in the *MPS Line Definition Section* on page 116.
- The TMS Num field contains the unique logical TMS number defined under *DTC Map Section* on page 112.
- The Rset_Profile Name field contains the name used in the RSET_PROFILE definition under *System Description Section* on page 106.

RSET Table Section

```
; this is a custom configuration table - if these are not specified here
; then the default rset profiles will be used and the lines above will
; be used to build the rsets.
;
[RSET_TABLE]
;-----
;      MPS Line   TMS   Rset_Profile
;      Num       Num   Name
;-----
RSET =   1:192     1     MPSTLine
RSET =   1:192     2     MPSTLine
```

Synclist Configuration Section

The synclist section is used to define the source(s) of timing and synchronization for the computer telephony (CT) bus on the local node. It is a prioritized list for maintaining CT bus operation using the failure redundancy features inherent in the MPS architecture. (See *TMS Computer Telephony (CT) Bus Clocking* on page 265.)

Synclist Section

```

; SYNCLIST SECTION
;
; This section is to specify the SYNCLIST for Reference Source A and
; Reference Source B.
;
; Each line can specify the sync list for a particular BPS (Back plane slot )
; The order in which the sync list is specified will be the order in which
; the TMSs will try to synchronize with the network.
; For example, if the sync has to be obtained from span 5 and then span 2, then
; the REF_SRC line should specify span 5 before span 2 in the list.
;
; NOTE: The Sync List for a particular Reference Source should all be on the
; SAME CHASSIS. It can exist on more than one BPS, but the order is important.
;
; In HUBNIC (NICLESS) MODE, if the span list is specified on more than one BPS,
; only the list specified on the first BPS is used. All others are ignored.
; Also, if both Ref Source A and Ref Source B are being specified in this mode,
; they have to be on different BPS as a TMS cannot drive both the ref sources.
; It can either drive RefSrc A or RefSrc B.
;
; Format of the sync command line
;
; [SYNC_LISTS]
; REF_SRC      A/B      Chassis      Bps      Sync S:C:D-Range
; REF_SRC      A        1            1        4:0:1-5
; REF_SRC      B        2            1        4:0:1 4:0:2
;
;
; [SYNC_LISTS]
; -----
;                RefSrcChBPSSpansList
; -----
REF_SRC_A      1  1  4:0:1-8
REF_SRC_A      1  2  4:0:1-8
REF_SRC_A      1  3  4:0:1-8
REF_SRC_A      1  4  4:0:1-8

REF_SRC_B      2  1  4:0:1-8
REF_SRC_B      2  2  4:0:1-8
REF_SRC_B      2  3  4:0:1-8
REF_SRC_B      2  4  4:0:1-8

```

The comments contained in the synclist section provide some recommendations and guidelines for configuring the synclist. The following is an expanded explanation.

The first uncommented line in this section contains the string [SYNC_LISTS], to define the section to the startup scripts. Each subsequent (uncommented) line will define the prioritized list of clocking sources to use. The entries are in tabular format and the required fields on each line are:

- The string REF_SRC
- The reference source being defined (i.e. A or B)
- The chassis number
- The backplane slot (BPS)
- The TMS slot number (or DCC), card, and device number delimited by colons (:).

For example, in the preceding sample, the first uncommented line after the [SYNC_LISTS] line is

```
REF_SRC      A      1      1      4:0:1      4:0:3-8      4:0:2
```

This configuration states that the first timing source to be used for REFCLK_A resides on chassis 1, BPS 1, slot 4, card 0, device 1. The slot is the slot number as labeled on the front of the TMS. The card number is always 0 (additional numbers are reserved for future enhancement). The device number is the span on the DCC. A range of devices, or spans, can also be specified as shown in the second field (4:0:3-8).

If the current clock source becomes disabled for any reason, the selection process starts at the beginning of the list to obtain a valid source, rather than proceeding to the next specified source in the list. For example, if the source of REFCLK_A is currently span 8 on DCC4, the clock selection process starts checking at DCC4, span 1 (first on the list) and runs through the list, instead of going directly to DCC4 span 2 (last on the list).

Although there are no absolute limitations or rules to building the synclist, there are recommendations for achieving the best degree of failure redundancy.

- In a multiple chassis system, the sources of REFCLK_A and REFCLK_B should be obtained from different chassis.
- In a single chassis system, the sources of REFCLK_A and REFCLK_B should be obtained from different BPSs (TMSs).
- A separate REF_SRC line should be used to define the list of sources from each chassis BPS.
- All available clock sources should be listed.

For analog systems (and for testing digital systems for which there is no operating span available or connected), the sync clocks are obtained from oscillators on the TMS mother board(s). (See *TMS Computer Telephony (CT) Bus Clocking* on page 265.) To specify a local oscillator from a TMS, enter “-1” in the <slot>:<card>:<device> fields, as follows:

```
REF_SRC      A      3      1      -1:-1:-1
```

The above statement specifies the source of REFCLK_A as the local oscillator on chassis 3, BPS1 (TMS in chassis 3, slot 1).

tms.cfg Major Section Functional Summary

tms.cfg File Field	Description
[SYSTEM]	Resource sets (singularly, <code>rset</code>) are defined here and referenced later in the file at the [RSET_TABLE] (see below) of the Line Definition Section. The procedure to build an <code>rset</code> is included in the header information. A resource set is a group of parameters that can later be referenced by name.
System Parameters	A group of parameters applied to the TMS as a whole and which override same such hard coded values in the TMS.
[RSRC_CONFIG]	Defines the configurations, referenced by name, that can be used in a TMS on the MPS component. The parameters for this definition are explained in the lines of the file that follow, and defined in the [CLASS] section (see next table entry).
[CLASS]	Defines the parameters for each [RSRC_CONFIG] to use (see above). This includes the resources to be loaded (in addition to the those defined in the <proto>.cfg files), each of which has a corresponding [CLASS] definition. Included in this definition are the number of specified resources to be loaded; the configuration file (*.cdf) to use; and an optional class name for reference. Parameters specified in [CLASS] override those of the *.cdf file.
[DTCMAP]	Defines the actual TMS system configuration parameters. This information is made up of three major subsections: BIND; DCCLOAD (currently not supported); and LOAD. The format and architecture of each is explicitly spelled out in the contents of the file immediately preceding each subsection definition table.
[VPS_LINE_DEF]	Maps the MPS lines to the physical lines (spans) on the TMS. Definition table contents are spelled out immediately preceding the section to which they apply.
[RSET_TABLE]	Defines custom configurations, and references the [SYSTEM] section where <code>rsets</code> were built earlier. If no table is present <code>rsets</code> are built using default profiles in conjunction with the information provided at [VPS_LINE_DEF] (see above).
[SYNC_LISTS]	Specifies the order in which the TMS' attempt synchronization with the network. The fashion in which the sync list is ordered determines the execution of this process. The sync list for a particular reference source must include entries for the same chassis only.

Protocol Configuration Files

Protocol configuration is defined by a `<proto>.cfg` file. One of these files is required for each protocol. A protocol is assigned to any number of spans (not individual lines) via the [Line Card Protocol Package Definitions on page 114](#). The `<proto>.cfg` file name is used in the `Protocol Pkg` field of that section of the `tms.cfg` file.

Each protocol configuration file contains two sections:

- The `[SPAN_CLASS]` section defines the resource set for the entire span(s) that will use this protocol. The value of `COUNT` should be the number of lines in a span (i.e., 24 for T1 or 30 for E1).
- The `[CLASS]` section specifies the resources class(es) to be used to implement the protocol. These are resource classes that are *always* used with this protocol. This section is configured the same way as the `[CLASS]` definition section of the `tms.cfg` file. (See [TMS Resource Definition Section on page 108](#).) The value of `COUNT` should be the number of lines in a span (i.e., 24 for T1 or 30 for E1).



System hardware limitations should be considered when configuring the `COUNT` value in the `[CLASS]` definition section. See [Resource Limitations on page 111](#).

```
=====
; proto.cfg
;=====
;
; This file is used to define the set of resources required to load
; in order to perform a particular protocol.
;
; S P A N   C L A S S
; the span class is a special class of resource for the proto.cfg file.
; it specifies the information used to load the span.  If more than
; one span class section is specified the first one found will be used
; and subsequent specifications will be ignored
[SPAN_CLASS]
COUNT = 24                ; number of resources of this class to load
CDF = ISDN.cdf             ; block = TIM, CPD, DTMF, TGEN
;<mode 0 param definition>

;*****
; R E S O U R C E   C L A S S   D E F I N I T I O N
; This section is used to define the protocol specific resources that
; should be loaded. This will allow additional dtmf,cpd,tgen,
; r2 resources to be loaded as well the ones specified in the proto.def
; files.
; This section specifies the configuration definition file (CDF) to use
; and the class name (optional) to assign to the created line resources.
; If the class name is specified here it will override any class name
; specified in the CDF file.
; Parameters specified here will override any parameters specified
; in CDF file.
; Mode definition is done at this level. Each set of configuration parameters
; specifies the values to set the paramters to for mode 0. This mode is used as the
; default mode for a resource. When specified here it will override
; the system defaults for the resources created.
; There will be one of these sections for each required resource.
;*****
;[CLASS]
;COUNT = <n>                ; number of resources of this class to load
;CLASS_NAME = <classname>    ; class name to use for this resource.
;CDF = <protocol>_<block>.CDF ; block = TIM, CPD, DTMF, TGEN
;<mode 0 param definition>
;
```

The \$MPSHOME/packages Directory



(This section applies to Solaris systems only)

This directory contains the actual installed Avaya software packages and default configuration files. The subdirectory naming conventions and subdirectories located in this directory are listed in the following table. In a typical configuration, not all subdirectories are present. Only the packages with configuration issues *not* covered in a user's manual are presented here. For a list of manuals, please use the *Reference Material* link available in PeriDoc.



The X-convention represents the numerical version of a package software release.

Avaya Software Packages

Symbolic Link in /opt/vps	As Found in \$MPSHOME/packages	Contents
PERIase	aseX.X.X	Directories and files specific to ASE.
PERIbrdge	brdgeX.X.X	Directories and files used for bridging calls in the system.
PERIcmpat	cmpatX.X.X	Shared libraries only necessary for PeriView release 5.X and MPS release 1.X compatibility.
PERIdist	distX.X.X	Used in distributing information from a source location to destination nodes in the MPS network; installs the web server and Perl scripts used for this and by PeriDoc, maintains related log files, installs a file compression utility.
PERIdocb	docbX.X.X	Software in support of PeriDoc, the comprehensive resource used to access Avaya on-line reference material and documentation.
PERIfw	fwX.X.X	Installs system library that enables platform-independent process execution.
PERIgase	gaseX.X.X	Global ASE shared libraries only used between release 5.X and MPS release 1.X.
PERIglobl	globlX.X.X	Current globally accessed directories and files including libraries and binaries used by all other packages.
PERIhostp	hostpX.X.X	Directories and files used in communicating with host computers. Protocol files are not detailed in this manual but instead can be found in the <i>Avaya Media Processing Server Series COMMGR Reference Manual</i> .
PERImps	mpsX.X.X	Directories and files used by MPS processes and utilities.

Avaya Software Packages

Symbolic Link in <code>/opt/vps</code>	As Found in <code>\$MPSHOME/packages</code>	Contents
PERIview	periviewX.X.X	Directories and files used by PeriView and its tools.
PERIperl	perlX.X.X	Integrates the Perl programming language into the Avaya software suite. Also sets the environment variable for <code>MPSHOME</code> .
PERIplic	plicX.X.X	Directories used in Avaya package licensing.
PERItms	tmsX.X.X	Directories and files used by TMS processes and utilities.

For sake of clarity and discussion, and as highly recommended by Avaya, this section uses `$MPSHOME` as the default root directory for the `packages` subdirectory. However, it is important to note that during installation, a user can elect to specify a directory name of their own choosing. If a user-specified distribution directory other than `/opt/vps` has been chosen, the released software packages reside in `/distdir/packages`, where `distdir` is the name of the user-specified distribution directory.

The subdirectories in `$MPSHOME/common` and `$MPSHOME/mpsN` look to the files located in this directory by means of symbolic links. This provides for control over the released software version used by the MPS system. If a user-specified distribution directory other than `/opt/vps` has been chosen, the symbolic links follow the path `/distdir/packages/version`, where `distdir` is the name of the user-specified distribution directory and `version` is a version of any Avaya software package installed on that system. The symbolic links themselves *always* exist in `/opt/vps`.

See the *Avaya Media Processing Server Series Solaris System Operator's Guide* for more information about these subdirectories.

`%MPSHOME%\PERIase - /opt/vps/PERIase`

This directory contains the Application Services Environment (ASE) software. ASE is the runtime environment for PeriProducer. By default, the system sets the `ASEHOME` variable to `/opt/vps/PERIase` on Solaris systems and `%MPSHOME%/PERIase`. The `stats` directory holds the application statistics, collected globally by the `VSUPD` process for all MPS components defined on a node. The configuration files of concern are both located in the `PERIase/etc` subdirectory.

The `/etc/ase.conf` file

This file has entries in the form of `name: value` and specifies where some commonly referenced ASE directories are located. It also defines the shared memory configuration. Currently the following named file entries are used for establishing directory relationships:

ase.conf File Field	Description
<code>MasterDBase</code>	The location of the database master file.
<code>LinkDir</code>	Default location for LINK programs.
<code>StatsDir</code>	Location of applications raw statistics files generated by VSUPD.
<code>CopyDir</code>	Location of statistics folders stubs.
<code>WebRoot</code>	Reserved for future enhancement.
<code>AseCoreDir</code>	Location for <code>vcore</code> files generated by VENGINE if it core dumps.
<code>VexLinkDirs</code>	Default location for vexlink linking.



To prevent problems caused by a modem connection loss, **amu** redirection to a device will not work unless the `ase.conf` file enables this functionality by setting the `AmuRedir` variable to `tty`. This can be accomplished by uncommenting the applicable line of the file.

Example: ase.conf

```
# Location of ASE elements in the format
#   element: dir
# Full path can be defined using HOME and other env variables
#
MasterDBase:${ASEHOME}/etc/VASDBlist
LinkDir: ${ASEHOME}/link
StatsDir:${ASEHOME}/stats
CopyDir: ${ASEHOME}/copy
WebRoot: ${ASEHOME}/web
AseCoreDir:${ASEHOME}/tmp
VexLinkDirs:.;${ASEHOME}/link
#AmuRedir:tty

#
# Shared Memory variables
# SharedMemory ---> Shared memory directory (file-based SM)
# ShMemorySegments ---> Maximum number of shared memory segments
# ShMemoryUpperLevelItems ---> Maximum number of Upper Level Items
# ShMemoryRequests ---> Maximum number of diff requests ( DELAY, WAIT ) and SETs
#
ShMemorySegments:          99
ShMemoryUpperLevelItems:  150
ShMemoryRequests:         2048
#
# Only for file-based shared memory
SharedMemory:              ${ASEHOME}/shm
ConstSharedMemory:        ${ASEHOME}/shm
#
```

Shared memory configuration is established in the second half of the file. The particulars for these entries are delineated in the section prior to their definitions. By default, shared memory is generated to support no more than 99 segments (shared folders in PeriPro), 150 total 01 level items in Linkage Section (e.g. 99 original 01 level items + 51 01 level items with REDEFINES clauses), and 2048 total outstanding DELAY with REQID and WAIT requests. The default configuration can be changed by modifying these variables and restarting the system.



If the number of segments is increased, the entries

```
set shmsys:shminfo_shmseg=101
set shmsys:shminfo_shmmni=101
```

in the file `/etc/system` also must be changed commensurately.

If the file contains the uncommented entry `SharedMemory: dirname`, an application's shared memory is implemented on top of files residing in the directory specified by `dirname`. If the entry does not exist, is commented out, or the directory cannot be accessed, the application's shared memory is implemented on top of the Solaris shared memory facility. On Windows machines, shared memory always resides on top of the file system.

File Based Shared Memory (FBSM) is not transient by default (as Solaris shared memory is) and is **not** removed or cleared when all applications exit or when the machine reboots (unless the FBSM is located in `/tmp` on Solaris). VENGINE's command line option `-K` is **ignored** for FBSM.

Use the following `vassm` utility option to remove a specified shared memory item:

```
vassm -c -i <itemname>
```

Normally, all Constant Shared Memory (CoSM) items reside in the directory specified by the `SharedMemory` entry in this file. However, they can be placed into a separate directory if the `ase.conf` file has the second entry `ConstSharedMemory: constdir`, where `constdir` represents this separate directory.



While the `SharedMemory dirname` entry must always reside on a local partition and have read/write permissions, the `constdir` directory may be located on a mounted read-only file system. This allows CoSM items to be shared over a distributed environment. Note that all CoSM initialization must be performed on the machine where the files are physically located.



Both the `dirname` and `constdir` directories can reside on a shared file system.

ASE relies on certain keywords defined in this file. However, a developer can add any arbitrary `name:value` pair to the file and extract it in an application by using the `get-configuration CALL` function. For more information on using `CALL` functions, and for information on vexlinking, see the *PeriProducer User's Guide*. To find out more about Shared Memory, see the *PeriView Reference Manual*.

The `/etc/services` File

The `services` file contains a list of processes (i.e., the services), some of which may be accessed by call processing or administrative applications. The file defines the port and protocol associated with each service and is used by the ASE (Application Services Environment) software process group.



VMST is aliased as `vms` in this file, but should not be confused with previous (“non-extended”) versions of VMS.

The following are sample excerpts of this file, followed by an [explanatory table](#):



Example: services

```
# Service      Port(s)  Protocol
#
#####
# Attention :
#   ports here are NOT TCP/IP ports,
#   but rather are 'handles' to VMS/vvpethers !!
#
#   All ports have to be in increasing order and by default must
#   be less than 509.  If more handles are required, use the
#   'vmst -p xxxx' command line option to increase the limit.
#####
vms             1-10
periweb        11-14    linfo rissue fissue
#pweb          12-14    linfo
periq          16       linfo
htmls          17       linfo
bankcore       18-20    linfo
sbrm           21-40    ping
commdaemon     41-60
vsupd         65
tcap           68       kick
ccss           69       kick
xsp            70       vps
amu            81-85
#
# PeriPro related services
#
peripro        101-110
vemul          111-120
timedaemon    121-130
screendaemon  131
#
vtcpd         132-135  linfo
#
# CTI/CTAP Related Services
ctapsrp       200
cti            201-205
csrout        206-210
sntry         211-215
#
# Oracle daemon
#
sqlclnt       221-230
#
# Custom made services
#
# starting from 241
test          240
```



Example: services

```

# Service      Port(s)  Protocol
#
#####
# Attention :
#   ports here are NOT TCP/IP ports,
#   but rather are 'handles' to VMS/vvpethers !!
#
#   All ports have to be in increasing order and by default must
#   be less than 509.  If more handles are required, use the
#   'vmst -p xxxx' command line option to increase the limit.
#####
vms            1-10
periweb       11          linfo rissue fissue
htmls         12-15       linfo
periq         16-17       linfo
bankcore      18-20       linfo
sbrm          21-22       ping
commdaemon    23-24
jsb           61-64       linfo rissue fissue
vsupd         65
sip           67          kick
tcap          68          kick
ccss          69          kick
xsp           70          vps
clipsr2       71-75       linfo rissue
amu           81-85
#
# PeriPro related services
#
peripro       101-110
vemul         111-120
timedaemon    121-130
screendaemon  131
#
vtcpd         132-145  linfo
#
#
# CSS 4.0.0 Related Services (201-220)
cti           201-205
# IPML/ICM (CSVAPI and HDX) IVR SCCS (RSM)
csrsm         206-210
#
# Screen-Pop to TAPI Server M1/DMS
cstapi        211-215
#
# IVR.DLL Interface for SCCS Connected to DMS100
cstapisccs    216-220
#
# CSS 3.3.1 Resources not supported in CSS 4.0
#ctapsrp      200
#csrouter     206-210
#sntry        211-215
#
#
# Oracle and other dbase
#
sqlclnt       221-237
corbaclnt    238          linfo rissue fissue
dcomclnt     239          linfo rissue fissue
#
# Custom made services
#
# starting from 241
test          254

```

Variable	Description
Service	Specifies the process name.
Port(s)	Identifies the system ports from which each process may be accessed. The port numbers represent internal handles to the VMST/VPETHER processes (i.e., they are not TCP/IP ports). The numbers must be less than 509, must be unique in this file, and must not conflict with the port numbers configured in the <code>vos.cfg</code> and the Solaris <code>/etc/services</code> files. The entries should be specified in increasing order of the port numbers. If this file is changed, all instances of VMST/VPETHER and the services must be restarted.
Protocol	Defines the protocol to use when accessing each process.

`%MPSHOME%\PERIbrdge - /opt/vps/PERIbrdge`

The PERIbrdge package is responsible for building the `tmscomm` component (see [The `MPSHOME/tmscommN` Directory on page 138](#)) and installing the Network Interface Controller Daemon (NCD) process (see [NCD on page 45](#)). The `vos.cfg` SRP startup file located in the `etc` subdirectory is copied to the `MPSHOME/tmscommN/etc` directory, where it is processed during system startup. Make production changes to that file only. Should the need arise to revert to a "clean" (original) version, *copy (do not cut)* the file from this package to the `MPSHOME/tmscommN/etc` directory.

The actual process executable file and the script used to create the `tmscomm` component are located under `MPSHOME/bin`. Do not make any changes to these files.

`%MPSHOME%\PERIdist - /opt/vps/PERIdist`

The PERIdist package contains utilities used for distributing information from a source location to destination nodes in the MPS network and processing Speech Server related log files. It also contains Perl scripts and a web server used by PeriDoc, the comprehensive online reference material and documentation resource. By default, the system sets the PERIDISTHOME variable to `/opt/vps/PERIdist` on Solaris systems and `%MPSHOME%\PERIdist` for Windows.

The apache directory contains files related to the web server. These files typically should not be edited. However, to specify nodes, after the initial installation, which are allowed to distribute files to the corresponding destination (PERIdist) node, edit the `apache/conf/httpd.conf` file. Details on this step are contained in the *Avaya Packages Install Guides*.

The `\etc` subdirectory of this package contains configuration files that determine the location to which Speech Server related log files are written. It also contains utility configuration files. Information on these files are contained in the *speech recognition* resource guides listed at the *Reference Material* link in PeriDoc.

`%MPSHOME%\PERIglobl - /opt/vps/PERIglobl`

This directory contains globally accessed software used by all other packages. On Solaris, this package's `/etc` subdirectory contains configuration files copied to `$MPSHOME/common/etc` during system setup (see [The MPSHOME/common/etc Directory on page 88](#)). On Windows, these files, with the exception of `compgroups`, are located in the `\commonetc\etc` subdirectory of this package and copied to `%MPSHOME%\common\etc`. The `compgroups` file resides in the Windows `\etc` subdirectory of the package and does not get copied over. The configuration files included are:

- `compgroups`
- `comptypes.cfg`
- `gen.cfg`
- `pmgr.cfg`
- `srp.cfg`
- `vpshosts`

These files should be used as backups for their deployed versions which were copied to the `MPSHOME/common/etc` directory. Make production changes to those files only: should the need arise to revert to a "clean" (original) version, *copy (do not cut)* the file from this package to the `MPSHOME/common/etc` directory.

The `misc` subdirectory of the PERIglobl package contains the `alarm.uts` file. This file contains the information for all predefined alarms in the system and is used to build the alarm database in `MPSHOME/common/etc`. This file should *not* be edited: to add or delete alarms from the database, use the PeriView Alarm Manager (see the *PeriView Reference Manual* for more information).

`%MPSHOME%\PERIview - /opt/vps/PERIview`

This directory contains the software used to run PeriView, the suite of graphical tools used for MPS system administration, operation, and control. This package's `etc` subdirectory contains configuration files copied to `MPSHOME/common/etc` during system setup (see [The `MPSHOME/common/etc` Directory](#) on page 88). The configuration files included are:

- `global_users.cfg`
- `periview.cfg`
- `ptlimages.cfg`

These files should be used as backups for their deployed versions which were copied to the `MPSHOME/common/etc` directory. Make production changes to those files only: should the need arise to revert to a "clean" (original) version, *copy (do not cut)* the file from this package to the `MPSHOME/common/etc` directory.

Complete information concerning PeriView can be found in the *PeriView Reference Manual*.

`%MPSHOME%\PERIplic - /opt/vps/PERIplic`

This directory contains the software necessary to run licensed Avaya packages. By default, the system sets the `plichOME` variable to `%MPSHOME%\PERIplic` for Windows (no such variable is set on Solaris). Though there is no configuration file directly related to `PERIplic`, certain options can be invoked when running the license server (`pllcd`). For details on this licensing mechanism, including file locations and options, see the *Avaya Packages Install Guides*.

`%MPSHOME%\PERItms - /opt/vps/PERItms`

This directory contains the software used by the Telephony Media Server (TMS) for system and parameter configuration. The `PERItms` subdirectory structure is described in the following table. Only those subdirectories directly related to configuration issues in the context of this manual are included. Some of the files identified in the `PERItms/cfg` subdirectory (`%MPSHOME%\PERItms\cfg` and `opt/vps/PERItms/cfg`) and `%MPSHOME%\PERItms\images` directory are documented in the sections that follow this one. Those files located in the `PERItms/site-cfg` subdirectory (`%MPSHOME%\PERItms\site-cfg` and `opt/vps/PERItms/site-cfg`) are discussed in more detail at [The `MPSHOME/common/etc/tms` Directory](#) on page 103.

PERItms

Subdirectory	Contents
cfg	Protocol configuration and definition files, as well as system level TMS software and hardware configuration files. Protocol files are discussed in the <i>Media Processing Server Series Telephony Reference Manual</i> . The remaining files include the <code>ali_triplets.cfg</code> , <code>atm_drv_triplets.cfg</code> , <code>atm_triplets.cfg</code> , <code>cardtypes.cfg</code> , <code>pcm_triplets.cfg</code> , <code>ps_triplets.cfg</code> , <code>scsi_triplets.cfg</code> , and <code>tms_triplets.cfg</code> files.
cfg	Contains the protocol Configuration Definition Files (*.cdf) as well as the <code>cardtypes.cfg</code> file. CDF files are generally not modified during the life cycle of the system and therefore are not discussed further in this manual.
images	Contains the same files as the Solaris package <code>/cfg</code> directory with the exception of the <code>cardtypes.cfg</code> file and CDF files (see Windows entry immediately above). Otherwise, information in the first column of this table applies.
etc	Parameter and text string data files.
site-cfg	Protocol configuration files and TMS system configuration files copied to <i>The MPSHOME/common/etc/tms Directory</i> (see page 103).

The `cardtypes.cfg` and majority of the `*.triplets` files are configured during manufacture and installation, and do not typically require editing. A few of the files, identified below, contain parameter reset variables that stipulate the interval at which the associated hardware resets itself when required. The defaults are normally adequate for most installations but can be changed if needed. If settings are changed, the system must be restarted for them to take effect.

The `/cfg/atm_triplets.cfg` File

The following is a copy of the `atm_triplets.cfg` file installed by default on the system.

Example: `atm_triplets.cfg`

```

;
;
PARAM Reset_Time = 10
;

```

The /cfg/ps_triplets.cfg File

The following is a copy of the ps_triplets.cfg file installed by default on the system.

Example: ps_triplets.cfg

```
;  
;  
PARAM Reset_Time = 10  
PARAM Latch1_Write = 0xff  
PARAM Latch2_Write = 0  
PARAM Volt1_ConvFactor = 23444  
PARAM Volt2_ConvFactor = 23444  
PARAM Volt3_ConvFactor = 58668  
PARAM Volt4_ConvFactor = 60445  
PARAM Curr1_ConvFactor = 81949  
PARAM Curr2_ConvFactor = 145859  
PARAM Curr3_ConvFactor = 28549  
PARAM Curr4_ConvFactor = 8037  
;  
;
```

The /cfg/tms_triplets.cfg File

The following is a copy of the tms_triplets.cfg file installed by default on the system.

Example: tms_triplets.cfg

```
;  
;  
PARAM Reset_Time = 10  
;  
;
```

%MPSHOME%\PERImps - /opt/vps/PERImps

This directory contains the software used on each MPS component in a system. The package contains configuration files copied to `MPSHOME/mpsN/etc` during system setup (see [The *MPSHOME/mpsN Directory* on page 139](#)). On Solaris systems, initialization and startup files are also included. These files should be used as backups for their deployed (copied) versions. Make production changes to those files only. Should the need arise to revert to a "clean" (original) version, *copy (do not cut)* the file from this package to the applicable directories as outlined in the following table.

PERImps



Subdirectory	Contents
etc	Sample configuration files copied to The <i>MPSHOME/mpsN/etc Directory</i> (see page 142) .
componentetc	Sample configuration files copied to The <i>MPSHOME/mpsN/etc Directory</i> (see page 142) .
misc	<code>S20vps.startup</code> file which is copied to the <code>/etc/rc3.d</code> directory (see S20vps.startup on page 65). Also includes the <code>peri.cshrc</code> and <code>peri.vshrc</code> shell initialization files and <code>peri.*</code> user files, copied to <code>/home/peri</code> (the default <code>\$HOME</code> variable for user <code>peri</code>) minus the <code>peri</code> prefix. See Installing Avaya Software on a Solaris Platform for more information on these files.

The MPShome/tmscommN Directory

The MPShome/tmscommN directory contains files used for bridging within and between MPS components. This component is built by, and files copied from, the PERIbridge package (see [%MPShome%\PERIbridge - /opt/vps/PERIbridge](#) on page 132).

The following guidelines apply when configuring the tmscomm component.

MPS 500

All nodes (including the secondary if one is available) must have one odd numbered tmscomm component (for example: 1, 3, 5, 7).

Check the vpshosts file to ensure that there is an entry for the new component.

MPS 1000

There can be, at most, two tmscomm components within an MPS 1000 cluster and these must be paired and reside on two separate application processor nodes.

For N+1 configurations, it is not necessary that a tmscomm component reside in the secondary node as long as a tmscomm pair exists in the cluster. A tmscomm pair is defined as an odd/even numbered pair of tmscomm components. That is, if there is a tmscomm1 present in the cluster, then tmscomm2 must be present. Similarly (for example), tmscomm3 (odd) pairs with tmscomm4 (even). Also, as an example, tmscomm2 and tmscomm3 is not considered a valid pair of tmscomm components.

Check the vpshosts file on all application processors within this MPS 1000 cluster to ensure that there is an entry for the new component(s).

The /etc subdirectory of the tmscomm component contains the vos.cfg file used to commence the NCD process at system startup (for additional information about this process, see [NCD on page 45](#)). A copy of the vos.cfg file follows: for an explanation of its format, see [The vos.cfg File on page 143](#).

Example: vos.cfg

```
#
# Example vos.cfg file.
#
# NAMEHOSTPORTPRICOMMAND LINE

ncd - - 0 ncd
```

The \$MPSHOME/mpsN Directory

The `$MPSHOME/mpsN` (`%MPSHOME%\mpsN`) directory path contains configuration and operations files specific to a single MPS component. The letter `N` denotes a number that identifies an MPS component associated with the node. The number assigned to an MPS can be obtained by issuing the VSH command `comp`.

One `mpsN` directory path exists for each MPS defined on the node in [The `vpshosts` File](#) (see page 93). For example, if four such components (numbered 1 through 4 in this example) are listed in the file, the following directories exist on the node:

`$MPSHOME/mps1` (`%MPSHOME%\mps1`), `$MPSHOME/mps2` (`%MPSHOME%\mps2`), `$MPSHOME/mps3` (`%MPSHOME%\mps3`), and `$MPSHOME/mps4` (`%MPSHOME%\mps4`).

The files identified in the `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`) and `$MPSHOME/mpsN/apps` (`%MPSHOME%\mpsN\apps`) directories are documented following this table.

`MPSHOME/mpsN`

Directory	Description
<code>apps</code>	<p>Contains call processing and administrative application executable (<code>*.vex</code>) and configuration (<code>*.acfg</code>) files. Executable and configuration files used by VENGINE for all call processing and administration applications are copied to this directory by means of Periview's Application Manager—Assign/(Re)Start Lines—Assign process.</p> <p>Application executable files are identified as <code><appname>.vex</code>. Application configuration files are identified as <code><appname>.acfg</code> and defined in the <code>aseLines.cfg</code> file when an application is assigned to a location.</p> <p>Shared Libraries, identified as <code><libname>.so</code>, are defined for an application by means of Periview's Application Manager—Configure Application—Shared Libraries Option. Shared Libraries are a copied to the <code>apps/lib</code> directory by means of Periview's Application Manager—Assign/(Re)Start Lines—Assign process when the application for which it has been configured is assigned.</p> <p>For additional information, see The <code>MPSHOME/mpsN/apps</code> Directory on page 140.</p>
<code>etc</code>	<p>Configuration and administration Files. Files include:</p> <pre>vos.cfg commgr.cfg vmm.cfg vmm-mmf.cfg ase.cfg aseLines.cfg ccm_phoneline.cfg ccm_admin.cfg tcad-tms.cfg tcad.cfg trip.cfg</pre>

The `MPSHOME/mpsN/apps` Directory

The `$MPSHOME/mpsN/apps` (`%MPSHOME%\mpsN\apps`) directory contains MPS application executable files (`*.vex`) and configuration files (`*.acfg`).

The executable and configuration files used by VENGINE for all call processing and administrative applications are copied to this directory by means of PeriView's Application Manager—Assign/(Re)Start Lines—Assign process. For a complete description of this process, refer to the *PeriView Reference Manual*.

The following are application file types:

`MPSHOME/mpsN/apps`

File	Description
<code><appname>.vex</code>	Call processing or administrative application's executable file. This file is copied to this directory when the application is assigned to a phone line.
<code><appname>.acfg</code>	Application configuration file. This file is copied to this directory when the application is assigned to a phone line.
<code>lib/<libname>.so</code>	Shared library file configured for an application. Shared libraries for these applications are located in the subdirectory <code>apps/lib</code> . These files also are copied during the Assign/(Re)Start Lines process. However, shared libraries must be defined for the application by the Application Manager—Configure Application—Shared Libraries option prior to being assigned along with the application.

Each application has a single configuration file that defines the application's configuration parameters. This allows applications executing on multiple lines to use the same configuration options.

An application's configuration file allows configuration options to be specified outside of the actual application itself. By editing the configuration file, the application can execute with a different set of parameters than those that would have been hard coded otherwise. In this way, an application can remain unaltered regardless of the configuration parameters with which it is executing. For example, by modifying the spoken language parameter in the configuration file, the application stays the same but the spoken language used changes. Changes to application configuration files will only affect applications assigned subsequent to the changes. Applications that have been assigned/started before the modifications will have to be terminated and unassigned with the PeriView Terminate/Un-Assign Lines tool then reassigned and restarted to have the changes take affect.

The file extension (`*.acfg`) is appended to the application name when the file is defined with the Application Manager—Configure Applications tool. If a configuration file is not defined before an application is assigned, a default file is created automatically during the Assign process. When the application is assigned, it gets appended to the list in the `aseLines.cfg` file (see [The `aseLines.cfg` File on page 149](#)). This list may be reordered with the Application Manager—Line Start Order During Reboot tool.

The `aseLines.cfg` file looks for applications in the directory `$MPSHOME/mpsN/apps` (`%MPSHOME%\mpsN\apps`). There are a number of variables that may be defined within an application's configuration (`*.acfg`) file, some of which are illustrated in the following example (typical applications may have more or less).

Example: <appname>.acfg

```
#
# This file is automatically generated by the application
# manager. Do not edit.
#
type=peripro
softTerm=600
args="-r -l 500 -k 600 -M 0 -h -D 128 -t 60 -o \"Please hold on.\"1
-C libpbisSYS.so -B /usr/lib/libc.so -B /usr/lib/ld.so "
interp=vengine
env=""
appmanFlags="-C is1:/home/peri/SHARED_LIBS_TEST/libpbisP.so -C
is1:/home/peri/SHARED_LIBS_TEST/libpbisSYS.so "
```

The file configured by means of PeriView's Application Manager Configure Application Tool (or created on Assign) is automatically generated and should not be edited manually. For additional information about application management, refer to the *PeriView Reference Manual*. For information about application development, see the *PeriProducer User's Guide*.

The `MPSHOME/mpsN/etc` Directory

The `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`) directory contains files for defining SRP, system, and application configuration parameters which may be unique for each MPS component. These files are identified in the following table and further described thereafter.

`MPSHOME/mpsN/etc`

File	Description
<code>vos.cfg</code>	Identifies the processes that run in the MPS VOS process group.
<code>commgr.cfg</code>	Configuration parameters required to manage external host communications.
<code>vmm.cfg</code>	Configuration parameters for the Voice Memory Manager (VMM).
<code>vmm-mmf.cfg</code>	Multi-Media Format files (MMFs) to be activated during system startup and related performance parameters.
<code>ase.cfg</code>	Lists processes that will be running in the ASE process group.
<code>aseLines.cfg</code>	Lists applications running on the specified MPS and the physical phone lines on which they are running.
<code>ccm_admin.cfg</code>	Stipulates phone line and service parameters for administrative applications.
<code>ccm_phoneline.cfg</code>	Stipulates phone line state and service parameter values.
<code>tcad-tms.cfg</code>	Configuration and startup parameters for the TMS.
<code>tcad.cfg</code>	TMS healthcheck and debug options.
<code>trip.cfg</code>	Process alarm and debug parameters.

The vos.cfg File

The `vos.cfg` file identifies the VOS processes that run on the MPS. This file is stored in the `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`) directory, and is used by SRP to start MPS-specific processes during system startup. The following is an example of this file:

Example: vos.cfg

```
#
# Example vos.cfg file.
#
# NAME      HOST    PORT    PRI    COMMAND LINE
trip       -      -      0      trip
tcad       -      -      0      tcad
vmm        -      -      0      vmm
ccma       -      -      0      "ccm -c admin"
ccm        -      -      0      "ccm -c tms -s 1-48"
commgr     -      -      0      commgr
vstat      -      -      0      vstat
#
# Uncomment the appropriate host protocol entries
#
#atte      -      -      0      atte
#vpstn3270 -      -      0      vpstn3270
#appc_cm   -      -      0      appc_cm
#cca_mgr   -      -      0      cca_mgr
#cca_serv  -      <port> 0      cca_serv
#geotel    -      -      0      geotel
#pos_serv  -      <port> 0      pos_serv
```

Variable

Description

NAME	Shorthand notation by which that process is known to SRP, <code>vsh</code> , and any other process that attempts to connect to it by name (essentially the process' well-known system name).
HOST	Allows the process to be started on a remote node. A dash ("-") specifies the local node.
PORT	Specifies the well-known port the process uses for IPC communication with other processes. If a dash is present, it indicates that the system fills in the port value at run time. A static port number only needs to be assigned for those processes that do not register with SRP, and must not conflict with the port numbers configured in the Solaris <code>/etc/services</code> file.
PRI	Real-time (RT) priority. This field is currently not used on Windows. A 0 indicates that the process should be run under the time-sharing priority class.

Variable	Description
COMMAND LINE	Actual command line (binary) to be executed. Command line arguments can be specified if the command and all arguments are enclosed in quotes (see <code>proxy</code> in examples above). The normal shell backslash escape mode (" <code>\</code> ") may be used to embed quotes in the command line. A command with a path component with a leading slash is assumed to be a full path designation and SRP makes no other attempt to locate the program. If the command path doesn't begin with a slash, SRP uses the (system) <code>PATH</code> environment variable to locate the item. Avaya package installations add the various binary location paths to this environment variable during their executions.

The `commgr.cfg` File

The `COMMGR` (Communications Manager) is the VOS software process that provides external host management functions. This process is a generic application interface to communication services independent of protocol. For information about the `COMMGR` process and related `COMMGR` commands, see [COMMGR on page 43](#).

The `commgr.cfg` file defines configuration parameters for the `COMMGR` process. It is stored in the directory `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`). For more information and protocol-specific examples, refer to the *COMMGR Reference Manual*.

VMM Configuration Files

VMM is responsible for many of the speech recording and playback functions in the MPS system. VMM provides run-time services for application-controlled playback and recording of speech elements. There are two VMM configuration files, which are stored in the `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`) directory: `vmm.cfg` and `vmm-mmf.cfg`. For information about the VMM process, see [VMM on page 49](#).

The `vmm.cfg` File

The `vmm.cfg` file defines configuration parameters for the Voice Memory Manager. Any configuration option available to VMM can be entered here and processed for VMM on system startup; however, options to VMM entered at a system console override those provided in this file. For the default file, basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. The options contained in the default file can *only* be issued through the file and *not* from the command line. The following example is the basic default file provided with the system.

Example: vmm.cfg

```
#
# Example vmm.cfg file.
#
# Note: For all available configuration options see documentation
# for VMM.
#
#
# numcachethd <number>
#
# Set the number of cache channel management threads to
# be used by VMM's cache management thread for loading
# audio data into voice data memory (VDM).
#
# <number> is the number of cache channel management
# threads to be started.
#
# default = 1
numcachethd 8
#
# tonetable <MMF filename>
#
# Specifies the MMF containing the tone table to be used
# to generate tones when not using a hardware based tone
# generator.
#
# <MMF filename> must be a full path (.mmd or .mmi
# extensions are not needed but will be accepted).
#
# default = no tone table will be loaded.
#
# tonetable /mmf/peri/dtmf
#
# vmdmaxlock <percentage>
#
# Specifies the maximum amount of VDM to use for locking
# elements. Care must be taken when modifying this parameter
# to insure that there is enough VDM available to page in
# data that is not locked as needed.
#
# <percentage> must be a whole percentage from 0-100.
#
# default = 50
#
# vdmmaxlock 50
```

For a full list of commands and options available to VMM, see the *VMM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

The `vmm-mmf.cfg` File

The `vmm-mmf.cfg` file identifies performance parameters related to MMF files. Any configuration option available to VMM in relation to MMF file processing is entered here and operated on upon system startup. However, options to VMM entered at a system console override those provided in this file. Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. Uncomment a line to activate that option (commented items depict the default value). Starting with MPS 2.1, MMF files are loaded automatically when placed in the appropriate (sub)directory in `$MEDIAFILEHOME`. Loading MMFs in `vmm-mmf.cfg` is still supported but not recommended.

The following example is the basic default file provided with the system.

Example: `vmm-mmf.cfg` Sheet 1 of 2

```
#
# Example vmm-mmf.cfg file.
#
# Note: For all available console options see documentation
#       for VMM.
#
#
#
# loadall <on | off>
#
# When set to on VMM will attempt to lock all elements
# into VDM upon activation on a first come, first serve
# basis. When set to off, only elements flagged for
# locking into VDM will be locked into VDM (again on a
# first come, first serve basis).
#
#       default = off
#
# Note: The loadall setting can be changed before and after
# mmfload commands. This allows some MMFs to have all
# elements locked in VDM while others have no elements
# locked in VDM.
#
#loadall on
#
# preload <number | all>
#
# Specifies the number of seconds of audio data to load
# for each element that is locked in VDM.
#
#       default = all
#
# Note: This option can be used to lock "more" elements in
# VDM by only locking the first n seconds of each element.
# If the remaining data for the element is needed it will
# be paged in as needed by VMM. As with the loadall option,
# preload can be changed before and after mmfload command.
#
# preload 2
```

Example: vmm-mmf.cfg Sheet 2 of 2

```
#
# mmfload <MMF filename,APPName>
#
# Activates <MMF filename> for the application <APPName>.
# If APPName is not specified the MMF will be activated
# system wide (in the system hash table).
#
# <MMF filename> must be a full path (.mmd or .mmi
#     extensions are not needed but will be accepted).
#
# <APPName> may be either "system" or the name of some
# application. If it is not specified "system" will be
# used by default.
#
```

If all the elements do not fit into Voice Data Memory (VDM) when you load (activate) the vocabulary file, there is not enough voice memory. To alleviate this situation, perform the following steps:

- Remove any elements not used by applications. See the *PeriStudio User's Guide* for more information on this procedure.
- Set the **vmm loadall** command to **off**. This allows only elements with a lock flag set to be loaded into VDM (limits total number of elements loaded).
- Set the **vmm preload** option to accommodate the **loadall** command (see previous bullet). If **loadall** is **off**, set **preload** to **all**. If **loadall** is **on**, the number of seconds to preload into VDM should be kept small if you are encountering this condition.

If this file is modified, VMM must be stopped and restarted for the changes to take effect.

For a full list of commands and options available to VMM, see the *VMM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

ASE Configuration Files

The ase.cfg File

The ase.cfg file identifies the names of processes that are associated with the Application Services Environment (see [ASE Processes on page 36](#)). If processes are intended to be run on nodes other than the one containing this file, this is to be indicated for each process in the HOST column. Otherwise, a dash in this column indicates the local node.



VMST is aliased as vms in its SRP startup files, but should not be confused with previous (“non-extended”) versions of VMS.

Example: ase.cfg

```

$1

#
# Example ase.cfg file.
#
# NAME  HOST    PRI    COMMAND LINE
vms     -        0      vms (Solaris entry)
vmst    -        0      vmst (NT entry)
    
```



The string "\$1" must be the only entry on the first line of this file. If the line does not exist, it must be added manually.

Field Name	Description
NAME	Shorthand notation by which that process is known to SRP, vsh, and any other process that attempts to connect to it by name (essentially the process' well-known system name).
HOST	Lists host node used for command and application processing. If processes are to run locally only, this column contains a dash.
PRI	Real time priority. Currently not supported on Windows. A value of 0 in this column both forces the use of the time-sharing class (in case it was set to something else) and sets the numeric priority value to the default base priority for the class. This setting should not be changed on Solaris systems.
COMMAND	Actual command line (binary) to be executed. Command line arguments can be specified if the command and all arguments are enclosed in quotes (see proxy in examples above). The normal shell backslash escape mode (" \ ") may be used to embed quotes in the command line. A command with a path component with a leading slash is assumed to be a full path designation and SRP makes no other attempt to locate the program. If the command path doesn't begin with a slash, SRP uses the (system) PATH environment variable to locate the item. Avaya package installations add the various binary location paths to this environment variable during their executions.

The aseLines.cfg File

The aseLines.cfg file identifies the applications to be run on the specified MPS and the physical phone lines to which they have been assigned. This file is stored in the \$MPSHOME/mpsN/etc (%MPSHOME%\mpsN\etc) directory.

Application and location information is added to this file by means of PeriView's Application Manager—Assign/(Re)Start Lines Tool—Assign process. Each time an application is assigned, it gets appended to the end of this list: conversely, when an application is unassigned with the Terminate/Un-Assign Lines tool, its entry is removed. This list drives the display in these tools. When the tool is launched, the order of the applications reflects the order of this list. The list can be reordered with the Application Manager—Line Start Order During Reboot tool: when it is, the aseLines.cfg file reflects the new order. For information about these procedures, see the *PeriView Reference Manual*.

See that table on the following page for explanations about each entry in the sample file that follows.

Example: aseLines.cfg

```
$1
#
# aseLines.cfg: line --> application database
#
# This file was generated by SRP on:
# Wed Apr 19 11:01:24 2000
#
#
# LineNode  Application      User
#
1      womquatECVinyl      jdg
2      womquatFabPoos    jdg
3      womquatTestRun   peri
4      womquatEskiePix  jftdel
39     womquatOutOfLuck  peri
26     womquatOutOfLuck  peri
77     womquatECVinyl    jdg
18     womquatECVinyl    jdg
```

Field Name	Description
Line	Numeric designation of the MPS phone line to which the application is assigned.
Node	Node name on which the line is configured and the application executes.
Application	Name of application assigned to the line. The application's configuration (*.acfg) and executable (*.vex) files are located in the \$MPSHOME/mpsN/apps (%MPSHOME%\mpsN\apps) directory of the MPS.

Field Name	Description
User	Name of user who assigned the application to the line. This information is used by the Application Manager for security purposes. For detailed information concerning user security, see the <i>PeriView Reference Manual</i> .

The string "\$1" must be the only entry on the first line of this file. If the line does not exist, it must be added manually.

In the example file above, user `jdg` assigned the application named `ECVinyl` to line `1` of node `womquat` first. This user assigned the same application to lines `77` and `18`, in that order, but only after lines `2`, `3`, `4`, `39`, and `26` had applications assigned to them. Though lines `2`, `3`, and `4` had different applications assigned to them by different users, they were assigned to those lines sequentially. Conversely, user `peri` assigned the application `OutOfLuck` to line number `39` before assigning it to the lower numbered line `26`. Thus, this order was established in the `aseLines.cfg` file, and is carried over as the default view in the applicable PeriView Application Manager tools.

CCM Configuration Files

The `ccm_phoneline.cfg` File

The `ccm_phoneline.cfg` file stipulates phone line state and service parameter values. It is stored in the `$MPSHOME/mpsN/etc (%MPSHOME%\mpsN\etc)` directory.

Any phone line configuration option available to CCM is entered here and processed for CCM on system startup. However, options to CCM entered at a system console override those provided in this file. Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. Uncomment a line to activate that option (commented items depict the default value). The following example is the basic default file provided with the system.

Example: `ccm_phoneline.cfg` Sheet 1 of 4

```
#
# $Id: ccm_phoneline.cfg,v 1.9 2002/02/19 20:58:02 russg Exp $
#
# Example ccm_phoneline.cfg file.
#
# Note that options in this file will be overridden by
# console options to ccm
#
#
#
# defLineState <state>
#
# Set the default state for the phone line. The phone line
# Will enter this state at startup, and whenever a call disconnects.
# The available values for <state> are BUSY and NOANSWER.
# Default = BUSY
#
# defLineState NoAnswer

#
# setEditSeq <user seq.>=<seq. str>[,DETECTALWAYS][,ENABLE][,KEEPTERM]
#
# user seq. : any of the following:
# US0 - User edit sequence #0
# US1 - User edit sequence #1
# US2 - User edit sequence #2
# US3 - User edit sequence #3
# USDEL - Delete (empties DTMF buffer) user edit sequence
# Removes all digits from the digit buffer. This edit
# sequence is active only when a request for digits is
# pending.
# USBKSP - Backspace user edit sequence
# Removes the last DTMF digit from the input buffer. This
# edit sequence is active only when a request for digits is
# pending.
# USTERMCHAR - Input Termination user edit sequence
# Causes DTMF input to complete. This edit sequence is
# active only when a request for digits is pending.
# seq. str : A sequence of 0 to 4 characters from the following character set:
# {0,1,2,3,4,5,6,7,8,9,*,#}
#
# NOTE: '<user seq.>=' Results in the configuration being cleared out
#
```

Example: ccm_phoneline.cfg Sheet 2 of 4

```
# DETECTALWAYS : If this argument is provided then the detection of this edit
#                sequence will occur (if enabled) whether or not a mx_ReceiveDTMF()
#                is pending. When this argument is not provided then detection will
#                only occur (if enabled) when an mx_ReceiveDTMF is pending.
#
#                The exception to this are the USDEL, USBKSP and USTERMCHAR user edit
#                sequences which this argument doesn't apply (i.e., USDEL and USBKSP
are
#                always active and US_TERMCHAR is only active when a mx_ReceoveDTMF()
#                request is in progress).
# ENABLE : If this argument is provided then the edit sequence will be enabled.
#          If this argument is NOT provided then the edit sequence will be
disabled.
#
# KEEPTERM : This argument is only valid when programming the sequence USTERMCHAR.
This
#           argument causes the USTERMCHAR edit sequence to be retained and
returned in
#           the event MX_EVENT_RECEIVE_DTMF_COMPLETE. If this argument is not
provided
#           when the USTERMCHAR edit sequence is programmed then the edit sequence
will
#           be removed from the digit buffer and will NOT be returned in the event
#           MX_EVENT_RECEIVE_DTMF_COMPLETE.
#
#           Configures a user edit sequence. User edit sequences are used for detecting
#           special caller keystrokes (touch tone input). This gives the caller and
application
#           additional control over DTMF input.
#
#           NOTE: The special case where NO user sequence is provided. This results in
#           the user edit sequence being cleared
#
#           Default = There are NO default edit sequences they must be configured and enabled
#           if they are to be used..
#
# setEditSeq ....
setEditSeq 'USBKSP=*0*'
setEditSeq 'USDEL=*#'
setEditSeq 'USTERMCHAR=#,enable'
#
# enEditSeq <user seq.>[,<user seq.>...]
#
#           user seq. : any of the following:
#                   US0      - User edit sequence #0
#                   US1      - User edit sequence #1
#                   US2      - User edit sequence #2
#                   US3      - User edit sequence #3
#                   USDEL    - Delete user edit sequence
#                   USBKSP   - Backspace user edit sequence
#                   USTERMCHAR - Input Termination user edit sequence
#
#           Enables all of the edit sequences specified in the argument list.
#           Default = All edit sequences are disabled by default.
#
# enEditSeq ....
```

Example: ccm_phoneline.cfg Sheet 3 of 4

```
#
# maxCacheLoadSize <size in Kilobytes>
#
# Sets the maximum number of pages in a single cache load request.
# Max Pages = (max size in kilobytes) / (size of a single VDM page in kilobytes)
# Range of values is 2, 3, 4, ..., 100.
# Default = 32
#
# maxCacheLoadSize 32

#
# setSvcParam <param>=<value>
#
# Sets a service parameter for CCM/TMS.
#
# Available parameters
# =====
# cpdMinSil                               >= 3000ms and <= 86400000ms
# Sets the minimum amount of silence required for the CPD resource to generate a
# SILENCE event.
#
# defLineState                            Busy or NoAnswer (Default Busy)
# Sets the default state between calls for CCM and the phone line resource.
#
# discguard                               5s - 10m, or 0 to disable (Default 5m)
# The maximum time CCM will wait for all outstanding responses to be received
# before it will force the disconnect sequence to complete
#
# discstrip                               >= 0ms (Default 0ms)
# Sets the amount of data to strip from the end of a recording that is terminated
# by the caller hanging up (disconnect).
#
# dtmfguard                               ON or OFF (Default OFF)
# Used to turn touch tone validation in TMS on or off.
#
# dtmftonedur                             40ms - 2040ms (Default 40ms)
# The minimum duration a touch-tone must exist for before the TMS considers
# it to be valid.
#
# first                                   2s - 86400s (Default 10s)
# Sets the maximum amount of time allowed for a caller to enter the first
# touch-tone in an input sequence (first character timeout).
#
# firstsil                                0ms - 20400ms (Default 0ms)
# Sets the amount of silence required to abort a record on first silence
# detection (before voice starts). This parameter only applies to synchronous
# recordings.
#
# inter                                   2s - 86400s (Default 10s)
# Sets the maximum amount of time allowed for a caller to pause in-between
# touch-tones in a multiple tone input sequence.
#
```

Example: ccm_phoneline.cfg Sheet 4 of 4

```
# intersil                0ms - 20400ms (Default 0ms)
# Sets the amount of silence required to automatically abort a recording at
# the end of voice.  This parameter only applies to synchronous recordings
#
# pickup                  1s - 86400s (Default 30s)
# Sets the guard timer for answering a call originated by the TMS.
#
# rsrcallocguard          0s - 86400s (Default is 1s)
# Specifies the time that TMS should wait for a resource to become available
# during a request by CCM to add a resource to its RSET.
#
# silstrip                0ms - 20400ms (Default 0ms)
# Sets the minimum amount of silence required before the DSP will start
# stripping the silence from the recording. This parameter only applies
# to synchronous recordings
#
# silthresh               0 - 63750 (Default 32)
# Sets the minimum amount of noise needed to distinguish between silence and
# non-silence for a recorder. This parameter only applies to synchronous recordings.
#
# totalcall               1s - 254h (Default 10m)
# Total call guard timer which is started when the connect event is sent to MX.
#
# ttstrip                 >= 0ms (Default 100ms)
# The number of milliseconds of data to strip from the end of a recording
# that is terminated by a touch tone. This parameter only applies to
# synchronous recordings
##
# setsvcp param dtmfguard=on
#
#
# recmode <mode>
#
# Sets the mode of recording that will be used (i.e., Disk based or Network based).
# This parameter affects both synchronous and asynchronous recordings.
# Available values for <mode> are DISK or NETWORK.
# Default = NETWORK
#
# NOTE: This parameter can not be set from vsh console or by the application,
# it can only be set during configuration.
#
# recmode NETWORK
```

For a full list of commands and options available to CCM, see the *CCM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*.

The `ccm_admin.cfg` File

The `ccm_admin.cfg` file stipulates service parameter values for administrative lines to which administrative applications are assigned. This file is stored in the `%MPSHOME%\mpsn\etc` directory.

Any configuration option available to an administrative CCM (`ccma` - see [The `vos.cfg` File on page 143](#)) is entered here and processed for this instance of CCM on system startup. However, options to CCM entered at a system console override those provided in this file. Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply. Uncomment a line to activate that option. For a full list of commands and options available to CCM, see the *CCM Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*. The following example is the basic default file provided with the system.

Example: `ccm_admin.cfg` Sheet 1 of 2

```
#
# $Id: ccm_admin.cfg,v 1.4 2002/02/19 20:58:02 russg Exp $
#
# Example ccm_admin.cfg file.
#
# Note that options in this file will be overridden by
# console options to ccm
#
#
#
# maxCacheLoadSize <size in Kilobytes>
#
#   Sets the maximum number of pages in a single cache load request.
#   Max Pages = (max size in kilobytes) / (size of a single VDM page in kilobytes)
#   Range of values is 2, 3, 4, ..., 100.
#   Default = 32
#
# maxCacheLoadSize 32
#
# setSvcParam <param>=<value>
#
#   Sets a service parameter for CCM/TMS.
#
```

Example: ccm_admin.cfg Sheet 2 of 2

```
# Available parameters
# =====#
# discguard                5s - 10m, or 0 to disable (Default 5m)
# The maximum time CCM will wait for all outstanding responses to be received
# before it will force the reset/disconnect sequence to complete
#
# firstsil                 0ms - 20400ms (Default 0ms)
# Sets the amount of silence required to abort a record on first silence
# detection (before voice starts). This parameter only applies to synchronous
# recordings.
#
# intersil                 0ms - 20400ms (Default 0ms)
# Sets the amount of silence required to automatically abort a recording at
# the end of voice. This parameter only applies to synchronous recordings
#
# rsrcallocguard           0s - 86400s (Default is 1s)
# Specifies the time that TMS should wait for a resource to become available
# during a request by CCM to add a resource to its RSET.
#
# silstrip                 0ms - 20400ms (Default 0ms)
# Sets the minimum amount of silence required before the DSP will start
# stripping the silence from the recording. This parameter only applies
# to synchronous recordings
#
# silthresh                0 - 63750 (Default 32)
# Sets the minimum amount of noise needed to distinguish between silence and
# non-silence for a recorder. This parameter only applies to synchronous recordings.
#
# ttstrip                  >= 0ms (Default 100ms)
# The number of milliseconds of data to strip from the end of a recording
# that is terminated by a touch tone. This parameter only applies to
# synchronous recordings
#
# setsvcp param discguard=5m
#
#
# recmode <mode>
#
#     Sets the mode of recording that will be used (i.e., Disk based or Network based).
#     This parameter affects both synchronous and asynchronous recordings.
#     Available values for <mode> are DISK or NETWORK.
#     Default = NETWORK
#
#     NOTE: This parameter can not be set from vsh console or by the application,
#     it can only be set during configuration.
#
# recmode NETWORK
```

TCAD Configuration Files

The `tcad-tms.cfg` File

The `tcad-tms.cfg` file stipulates configuration and startup parameters for the TMS. Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. Uncomment a line to activate that option (commented items depict the default value). The following example is the basic default file provided with the system.

Example: `tcad-tms.cfg`

```
#
# Example tcad-tms.cfg file
#

#
# tms-cfg-timeout n
#
# Synopsis:
# Set the maximum amount of time(in seconds) to wait
# for a response for a single signal sent to TMS.
# n = seconds, 0 disables timeout.
# Default = 300
#
#tms-cfg-timeout 60

#
#tms-cfg-start
#
# Synopsis:
# Uses tms_AcceptConfig to try to start config. If
# request is rejected by TMS, load aborts, otherwise
# system state will be set to 'config'.
#
tms-cfg-start

#
#syssetparams '<id> <len> <val>'
#
# Synopsis:
# Sets one system parameter.
# id = parameter id
# val = a uint specifying the value
#

#
# Start Loading/Configuring the TMS hardware
#
ldr-start

#
# Notify tcad that load of TMS is complete
#
tms-cfg-done
```

The `tcad.cfg` File

The `tcad.cfg` file stipulates TMS debug options. It is stored in the `$MPSHOME/mpsN/etc (%MPSHOME%\mpsN\etc)` directory.

Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. Uncomment a line to activate that option (commented items depict the default value). Options to TCAD entered at a system console override those provided in this file. The following example is the basic default file provided with the system.

Example: `tcad.cfg`

```
#
# Example tcad.cfg file.
#
# Note that options in this file will be overridden by
# consoled options to tcad
#
#
# dlogDbgOn <destination>,<dbgObject>
#
# Enable tcad debug output and redirect it to stdout
# To redirect it to a file rename stdout to file
#
#dlogDbgOn stdout,general
```



The `STDOUT` and `STDERR` destination objects are for debugging purposes only and should not be used.

TRIP Configuration Files

The trip.cfg File

The trip.cfg file stipulates process alarm, healthcheck, and debug parameters. It is stored in the \$MPSHOME/mpsN/etc (%MPSHOME%\mpsN\etc) directory.

Basic descriptions and formats of file entries are given immediately preceding the actual data to which they apply, and are relatively self-explanatory. Uncomment a line to activate that option (commented items depict the default value). Options to TRIP entered at a system console override those provided in this file. The following example is the basic default file provided with the system.

Example: trip.cfg Sheet 1 of 2

```
#
# Example trip.cfg file.
#
# Note that options in this file will be overridden by
# console options to trip
#
#
#
#hc-interval n
#
# Synopsis:
# TMS system internal health check interval, in seconds.
# n = seconds, range 0..TBD. Value of 0 disables health check.
# Default = 2 (every 2 seconds).
#
hc-interval 15

#hc-miss-cnt-max n
#
# Synopsis:
# TMS Health check miss count maximum
# n = count allowed to be missed, range 0..100
# Default = 5.
#
hc-miss-cnt-max 3

#secondary-vos-ctrl-delay n
#
# Synopsis:
# A secondary TRIP will delay attempting to get control of the
# TMS by the amount of seconds specified by this parameter.
# n = seconds, range 0..TBD.
# Default = 300 (5 min)
secondary-vos-ctrl-delay 300
```

Example: trip.cfg Sheet 2 of 2

```
#
# defaultroute procName
#
# Set the default route for asynchronous TMS messages
# (i.e., alarms) to the specified process.
# procName = name of process to route asynchronous TMS messages to
#   Default = tcad
#
#
# defaultroute tcad

#
# dlogDbgOn <destination>,<dbgObject>
#
# Enable tcad debug output and redirect it to stdout
# To redirect it to a file rename stdout to file
#
#dlogDbgOn stdout,general
```



The **STDOUT** and **STDERR** destination objects are for debugging purposes only and should not be used.

TMS Watchdog Functions

All traffic between an MPS node and its associated TMS systems are sent through TRIP. After the connection between a TMS and TRIP is established, it regularly sends the TMS a *ping* message, which resets the watchdog timer in the Network Interface Card (NIC). If the watchdog timer expires (i.e., is not reset because of a system failure), the TMS can reboot the host node. Similarly, if TRIP fails to receive a reply from the NIC card, it can reset the TMS.

The `hc-interval` entry in this file indicates the time interval, in seconds, for TRIP to ping the TMS. The `hc-miss-cnt-max` entry stipulates the number of missed pings allowed before TRIP reboots the TMS. Both of these settings are used in conjunction with the watchdog timer in the NIC card (see [Network Interface Controller \(NIC\) or Hub-NIC](#) on page 27).

This page has been intentionally left blank.



3

Common Configuration

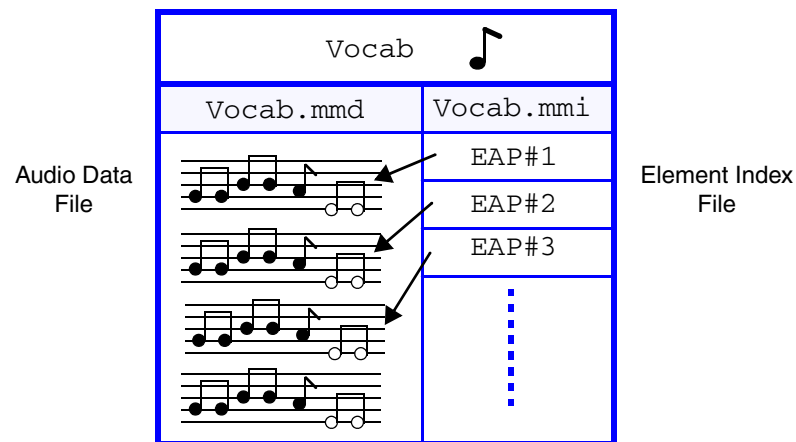
This chapter covers:

- 1. Multi-Media Format Files (MMFs)**
- 2. Call Simulator Facility**
- 3. Alarm Filtering**
- 4. Interapplication/Host Service Daemon Data Exchange**

Multi-Media Format Files (MMFs)

A Multi-Media Format (MMF) file contains audio elements (vocabulary and Caller Message Recording [CMR]) and/or fax data. An individual message in an MMF file is called an *element*. One MMF file will normally contain many elements. A single MMF actually consists of two files:

- The *index file* consists of element names, sizes and other attributes organized by means of *Element Access Pointers* (or EAPs). The index file has a .mmi extension.
- The *data file* contains audio data (audio, fax, Telecommunications Device for the Deaf [TDD] tones, etc.) and has a .mmd extension.



Anatomy of an MMF File

In the diagram above, each entry in `Vocab.mmi` points to audio message data in `Vocab.mmd`. Together, they constitute an *MMF element*, which is the entity comprised by an index/recording pair of data entries. This scheme allows MMF elements to be accessed randomly (in any order).

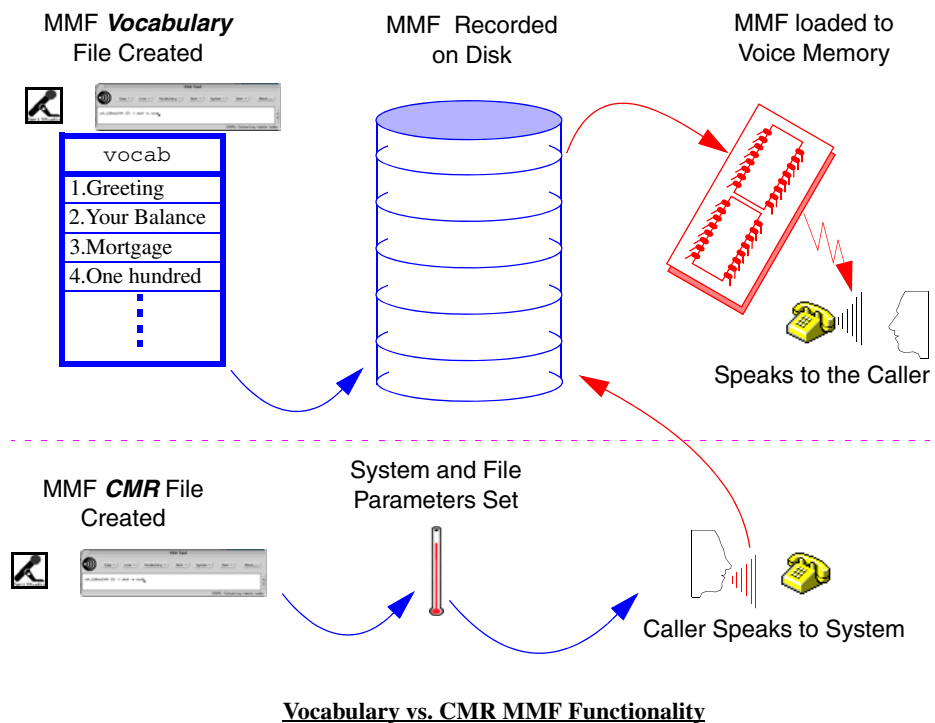
How to Create an MMF File

In order to use MMF vocabulary files, empty MMF files must be created into which vocabulary elements or recorded messages can be stored. This is accomplished with the `mkmmf` command or with PeriStudio. For information on how to create an MMF file with PeriStudio, refer to the *PeriStudio User's Guide*.

Vocabulary MMF Files vs. CMR MMF Files

Applications use MMF files as *vocabularies* to output named elements over the telephone lines. To make recordings from callers, applications use MMF files that are designated for use with the CMR (Caller Message Recording) feature. Although it is possible to both record into and play back from a single MMF file, separate files are generally used for vocabulary and CMR functions.

For more information on CMR, see the *Avaya Media Processing Server Series Caller Message Recording (CMR) Feature Documentation*. The sections that follow concentrate on using MMF files for vocabularies.

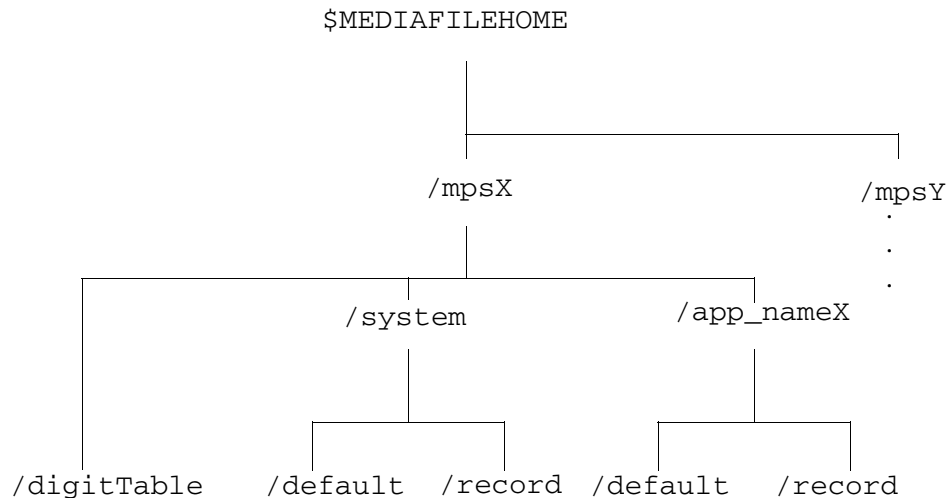


For the purpose of providing voice output over telephone lines, MMF files are played from cache memory.

Activating MMF Files

When an MMF file is activated, its element names are loaded into system memory for fast lookup. The recorded data of the elements are also loaded into Voice Data Memory (VDM).

VMM automatically loads all MMFs placed in the \$MEDIAFILEHOME directory structure. The directory structure is configured according to component and the function of the MMF:



- \$MEDIAFILEHOME - The root media file directory (typically /mmf/peri)
- /mpsN - The component subdirectory. There is one subdirectory for each component installed on the system.
- /system - Contains files for MMFs used system wide. There should be only one system subdirectory per component. Any MMFs placed in this subdirectory are available to all applications.
- /appname - Contains files for MMFs used by a specific application. There should be an application-specific subdirectory for each application that uses an application-specific MMF. Any MMFs placed in this application are available only to the specific application.
- /record - Contains the MMF used as the default record (CMR) MMF for the system/application. This should contain **only one** MMF.
- /default - Contains the MMF used as the default play MMF for the system/application. This should contain **only one** MMF.

For example, the component MPS1 uses `library.mmf` as the default system play MMF and `messages.mmf` as the default system record MMF. An application `banking1` uses an application specific MMF, `banking.mmf`, as a default play MMF. The `numset.mmf` application is available to all applications. The following are the directory locations for the MMFs:

```
$MEDIAFILEHOME/mps1/system/play/library.mmf
$MEDIAFILEHOME/mps1/system/record/record.mmf
$MEDIAFILEHOME/mps1/system/numset.mmf
$MEDIAFILEHOME/mps1/banking/play/banking.mmf
```



Loading MMFs using the `vmm mmfload` command in the `vmm-mmf.cfg` file is still supported but not recommended. Using `mmfload` in `vmm-mmf.cfg` instead of using the `$MEDIAFILEHOME` directory structure/VMM automatic load function can create problems in N+1 redundancy systems. `mmfload` commands in `vmm-mmf.cfg` are processed **after** VMM finishes loading all other MMFs in the `$MEDIAFILEHOME` directory structure.

Both `mmfload` and `mmfunload` (unload MMFs) commands may be issued from the VSH command line while the system is running. For a step-by-step procedure of how to activate and deactivate MMF files, see the *Avaya Media Processing Server Series System Operator's Guide*.

You can also set applications up with their own dedicated MMF files for recording and speech playback. (See [Application-Specific MMF Files](#) on page 174.)

Delimited and Partial Loading

By default, the system loads each element’s full name into system memory (as opposed to voice memory). Complete element name loading may cause complications if system memory is limited. There are two methods of conserving system memory when loading elements, which are set using `vmm nload` command in the `vmm.cfg` configuration file (see *The vmm.cfg File* on page 144):

- *Delimited loading* loads element names up to a special delimiter character (the semicolon ";"). When creating elements in PeriStudio, assign names to the elements such that they contain the delimiter character. Then place the command `vmm nload del` into the `vmm.cfg` configuration file.
- *Partial loading* saves memory by only loading a certain number of characters of each element name, but this increases the possibility of a name conflict. To specify partial loading, set `nload` to the number of characters to load from the element names. For example, to load only the first 10 characters, put the following line in the `vmm.cfg` configuration file: `vmm nload 10`

The table below compares the two methods of loading element names into memory:

- Partial loading has been set to the first three characters.
- Delimited loading always uses the semicolon (" ; ") as the delimiting character.

Delimited vs. Partial Loading of Vocabulary Labels

Element name	Partial loading	Delimited loading
Greeting	Gre	Greeting
Yes	Yes	Yes
No	No	No
P123;Mortgage	P12	P123
P456;Check	P45	P456
Breakaway	Bre	Breakaway
Thank;you;for	Tha	Thank
BlahBlahBlah	Bla	BlahBlahBlah



Once the system is initialized, the value for `nload` cannot be modified.



VMM allows identical element names after the names are truncated. However, only the element that was loaded first will be accessed when referenced. To avoid this problem, make sure that all element names will be unique after the partial or delimited loading you selected. (See the above table.)

Audio Playback

By default, VMM does not attempt to load all vocabulary elements into VDM. If "loadall on" is specified in the `vmm-mmf.cfg` file, VMM attempts to load all vocabulary items into VDM. Elements not loaded into audio memory will be cached in/out of memory as necessary.

Proper setting of `vdmmxlock` is important to insure there is enough VDM reserved for caching. If the size of the activated MMF elements exceeds available voice memory (VDM becomes depleted), an alarm is generated. If some of the VDM is freed by deactivating one or more MMF files, the MMF files to be loaded must be deactivated and then reactivated in order to use newly available memory. The following parameters directly affect VDM performance. `pagesize`, `vdmmxlock`, and `preload` are set in the `vmm.cfg` file (see [The `vmm.cfg` File](#) on page 144). `loadall` is set in the `vmm-mmf.cfg` file (see [The `vmm-mmf.cfg` File](#) on page 146). If changes are made to these entries, VMM must be stopped and then restarted for the changes to take affect. For information on stopping and starting VMM, see the *Avaya Media Processing Server Series System Operator's Guide*.

Configuration Parameters for Voice Data Memory Management

VMM Parameter	Description
<code>pagesize</code>	The size, in kilobytes, to use for a single segment of VDM. A value that is too large means that more memory is taken from VDM. Decreasing the value makes more efficient use of VDM (less wasted space) but uses more system memory. The default value is 8 Kb. Typically, the defaults should be used. Changes to this parameter should be considered in the context of the value of <code>vdmmxlock</code> (see below).
<code>vdmmxlock</code>	Specifies the maximum amount of VDM, as a percentage, to use for locking elements. This option is used to ensure there is sufficient VDM available for the VMM caching mechanism to function efficiently. Unless good reason exists otherwise, the default value of 50% should be used. Increasing this value makes audio element access quicker, but reduces VDM available for caching audio data not locked in VDM; decreasing this value has the opposite affect. Changes to this parameter should be considered in the context of the value of <code>pagesize</code> (see above).
<code>preload</code>	Specifies the number of seconds of audio to load into VDM prior to an element's initial usage. This option is used in conjunction with <code>loadall</code> (see below). If <code>loadall</code> is turned on, VMM attempts to preload audio data for each element: if <code>off</code> , VMM makes the attempt only for locked elements (see "Custom Loading" on page 171 for more information). If set in the <code>vmm-mmf.cfg</code> file, should precede any <code>mmfload</code> commands. Default value is <code>a11</code> (audio data loaded into VDM).

Configuration Parameters for Voice Data Memory Management

VMM Parameter	Description
---------------	-------------

<code>loadall</code>	Determines whether VMM should load and lock all elements into VDM when activating MMF files. This option is used in conjunction with <code>preload</code> (see above). When <code>loadall</code> is <code>off</code> (the default), only elements with the lock flag set are loaded into memory. When <code>loadall</code> is <code>on</code> , VMM attempts to load all elements into VDM, regardless of their lock flag status (see “Custom Loading” on page 171 for more information). If set in the <code>vmm-mmf.cfg</code> file, should precede any <code>preload</code> and <code>mmfload</code> commands.
----------------------	---

The following formula should be used to determine the maximum safe setting for `vdmmmaxlock`. Note that this calculates the maximum safe setting; not the optimal setting, which depends upon what vocabulary items are spoken and their frequency. Exceeding the value determined using this calculation may result in the system failing to play an item.

$$maxvdmmmaxlock = 100 - \left[\frac{\left(\frac{2 \times maxCacheLoadSize \times numberLines}{pageSize} \right)}{numberCachePages} \right]$$

where:

`maxCacheLoadSize` = the value of `ccm maxcacheloadsize`

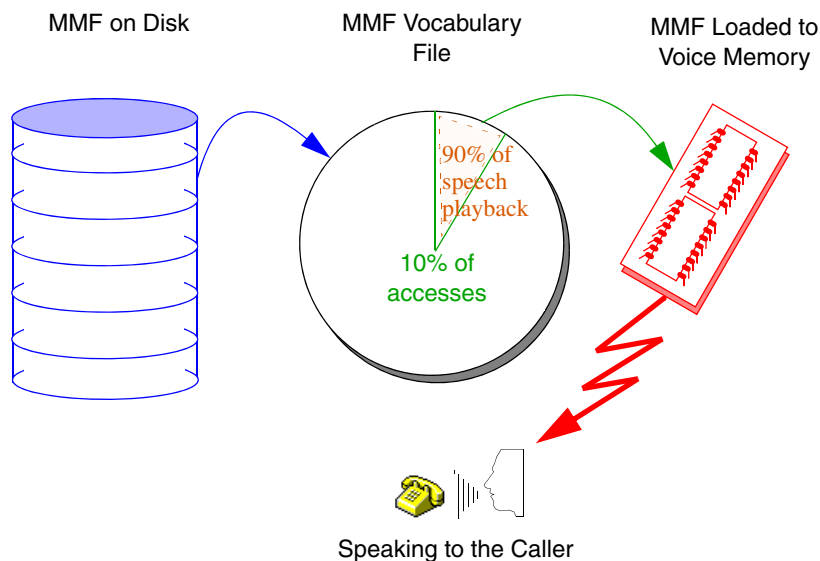
`numberLines` = the number of lines in the system

`pageSize` = the value of `vmm pagesize`

`numberCache Pages` = Number of pages in cache value returned by `vmm cachestatus`

Custom Loading

If all of the data doesn't fit into voice memory, it will be necessary to select which elements to load into memory and which ones to cache. By default, elements are loaded on a first-come-first-serve basis.



MMF Lock and Load Example

For example, if 90% of the speech playback comes from 10% of the elements, to save memory, set the *lock* flags of the frequently used elements and allow the rest to be played as needed. (The setting of lock flag is done in PeriStudio. See the *PeriStudio User's Guide* for more information.) To enable selective loading, issue the `vmm loadall off` command from the `vmm.cfg` file prior to the command for loading the MMF file.

On the other hand, if there is an MMF file for which all elements should be loaded into voice memory, use the `vmm loadall on` command. Once `loadall` is enabled, this setting stays in effect until explicitly changed.

When the size of voice memory is less than the total combined size of all audio data, it is best to lock the most frequently used elements and adjust the `vmm preload` value in conjunction with the `vmm loadall` option. To determine which elements are spoken frequently, use the `vmm refstatus <mmfname>` option.

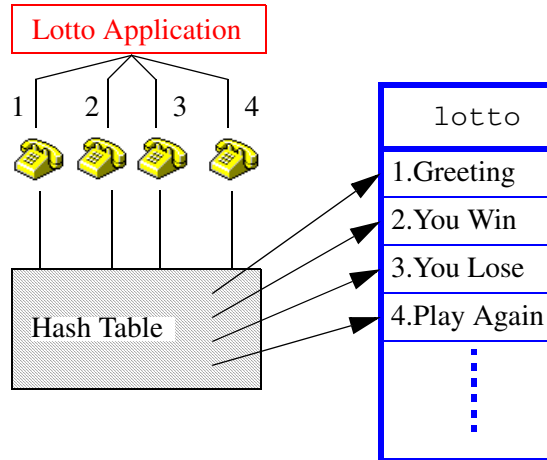


The `vmm loadall` parameter may be set from the command line, and changed on an as-needed basis.

Using Hash Tables

To improve access time to vocabulary MMF files, the VMM process creates a hash table. The following example illustrates this concept:

A Lotto application that runs on four phone lines (1-4) uses the `lotto` MMF file. The elements within the file may be located on disk or in voice memory. The application accesses the file through a hash table.



Basic Hash Table Schematic

Application-specific hash tables are created using the following command, which must be issued before those applications are started. If this command is not used, the VMM process automatically generates the hash tables and sets the `hashfirst` sequence to first search the system-wide hash tables.

```
vmm appinit <application name>
```

To change the hash table lookup sequence for an application, enter the following command. `local` indicates the application looks to its own hash table first: `system` instructs it to use initially the system wide hash table. If an element is not found in one, the other is then searched.

```
vmm hashfirst <application name>,{system | local}
```

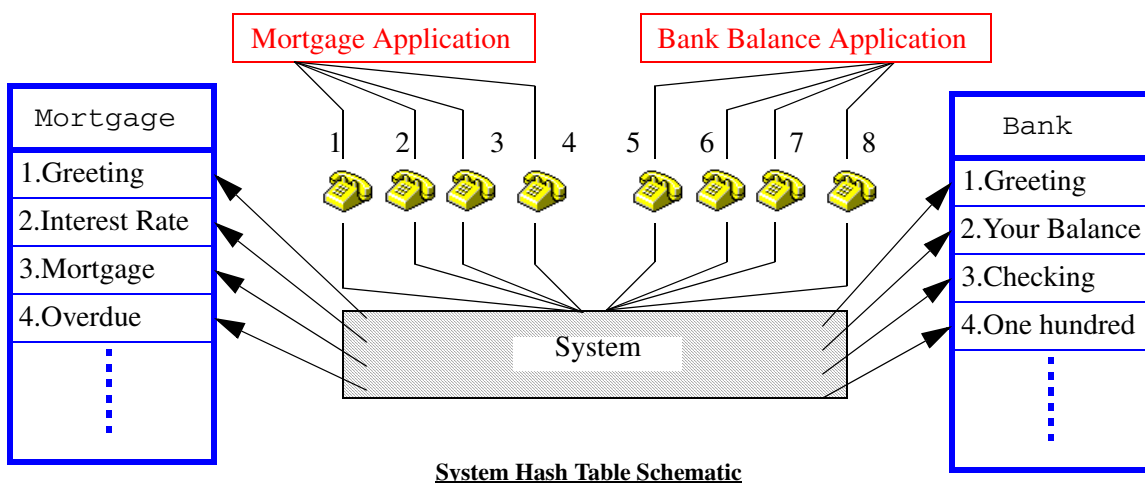


The following is important information about hash tables:

- One hash table can index multiple vocabulary files. However, it cannot distinguish *duplicate element names* across index files.
- A hash table can service the entire system or just one application.
- To automate the entire MMF process, the `appinit` and `hashfirst` commands can be added to the `vmm-mmf.cfg` file, in that order.

System MMF Files

This is the simplest way to organize vocabulary MMF files. System MMF files are public. They can be accessed by any application on the system.



In the illustration above, the two applications use two different vocabulary MMF files that are hashed together into a single system hash table. Typically, system MMF files contain such common and frequently accessed elements, such as Dual Tone Multi-Frequency (DTMF) tones (`dtmf`) and numeric elements (`numset`).

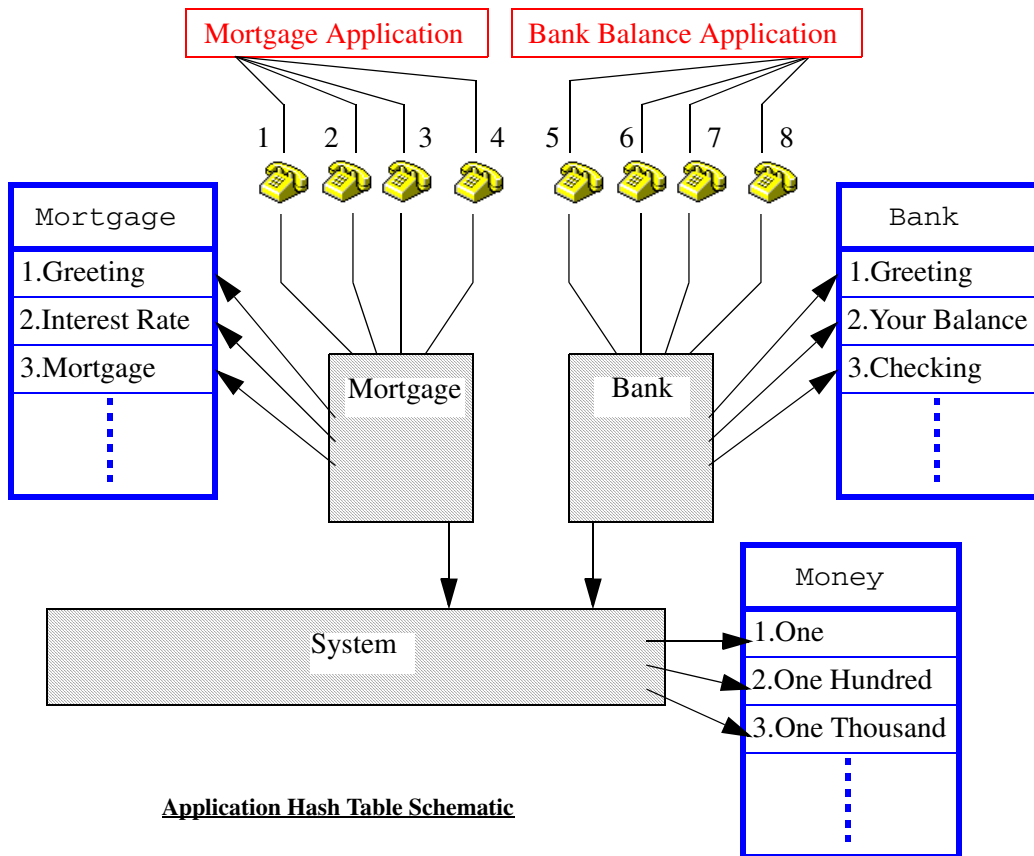
All vocabulary elements could be hashed into a single system table, like in the illustration above, but this is not recommended (if there is more than one application). Large hash tables can impact system performance because of the longer look-up time. Also, a single hash table does not allow duplicate element names (i.e., every element name in every hashed MMF file must be unique).

In the preceding illustration, both MMF files have an element called *Greeting*. When these elements are hashed together, the first element hashed (Mortgage's *Greeting*) is the one that is spoken. So, if the Bank application requests to speak its *Greeting*, it will get the Mortgage *Greeting* instead. To overcome this problem, use application-specific MMF files.

It is recommended that the system hash table be used only for common MMF files that will be accessed by several applications. Use application-specific hash tables for all other MMF files. If there is only one online call processing application, all MMF files should be activated using the system hash table.

Application-Specific MMF Files

The following illustrates how multiple applications can use application-specific MMF files to avoid element name conflicts:



In this configuration, there are still two vocabulary elements with the name *Greeting*. However, each application has been given its own MMF file, and all common elements (such as dollar amounts) have been grouped into one system MMF file. The `hashfirst` parameter is also set to `local`, which causes speak requests to attempt element lookups first in the application-specific MMF file.

This setup works as follows:

- If the Mortgage application attempts to speak its *Greeting*, the Avaya Media Processing Server (MPS) first looks at the Mortgage hash table, finds the correct element, and speaks it.
- If the Bank application attempts to speak its *Greeting*, the MPS first looks at the Bank hash table, finds the correct element, and then speaks it.
- If the Mortgage application attempts to speak *One Thousand dollars*, the MPS first searches the Mortgage hash table, then proceeds to the system hash table and find the element(s) *One Thousand dollars*.
- If the Bank application attempts to speak *One Thousand dollars*, the MPS first searches the Bank hash table, then proceeds to the system hash table and finds the element *One Thousand dollars*.
- The Bank application, for example, cannot access *Interest Rate*, which is an element specific to the Mortgage application. That is, application-specific elements can only be accessed by the applications for which they have been activated.

Default Vocabulary and Record MMF Files

MMF files may be set as default MMF play or record files. This is used to emulate previous generation Avaya systems that use the 24-Byte Header mode. The default vocabulary is the only vocabulary MMF file that is searched when an application makes a speak request that specifies an element number (instead of an element name). To set a default vocabulary file for a specific application or system-wide, add the MMF to the specific subdirectory in the \$MEDIAFILEHOME directory structure (see [Activating MMF Files on page 166](#)).

System-wide Record:

\$MEDIAFILEHOME/*component/system/record/mmfname*

System-wide Vocabulary:

\$MEDIAFILEHOME/*component/system/default/mmfname*

Application-specific Record:

\$MEDIAFILEHOME/*component/appname/record/mmfname*

Application-specific Vocabulary:

\$MEDIAFILEHOME/*component/appname/default/mmfname*

Diagnostics and Reports

The following table explains some useful MMF file diagnostics commands:

Commands for MMF Diagnostics

Command	Description
<code>vmm mmfstatus</code>	Shows the MMF status report, including MMF files that are currently activated, and the number of elements loaded from each MMF file. Also includes space allocations for each file.
<code>vmm refstatus <mmf file></code>	Displays the elements in <code><mmf file></code> including their EAP numbers, how many times they have been referenced, and whether or not an item is locked in VDM. You can use this command to verify that all elements were loaded.
<code>vmm hashreport <system app_name all></code>	Displays a hash table report, indicating which elements have been loaded to the hash tables, along with the lengths of the elements and the MMF files they were loaded from. Use <code>all</code> to display a report for each active application hash table as well as the system hash table.
<code>vmm appstatus <app_name></code>	Displays an application status report, including which MMF files have been activated for application-specific use.
<code>vmm repconfig</code>	Displays the VMM configuration report, including the parameters used during MMF file activations.

All status reports can be issued with the shorthand "`st`" if desired (i.e. `vmm mmfstatus` or `vmm mmfst`).

The MPS allows MMF files to contain both digital and analog versions of an element. (They may both have the same name.) If a name conflict exists, the first one loaded will be spoken.

Synchronizing MMF Files Across Nodes



Presently available on Solaris platforms only.

In instances where many nodes utilize the same MMF files, and changes to these files would mean putting an undo burden on network facilities, management, and customer use, the **Z**ero **A**dministration for **P**rompts (ZAP) utility is used to automate the process.

This automated MMF file synchronization facility provides a means of administering updates to and maintaining consistency between all activated instances of an MMF file which reside on different nodes on a network. It determines if a set of MMFs contain identical elements and provides the capability to rectify any differences between files. In addition, reports illustrating the differences between the source and target MMF files and the results of modifications made to the target MMF files are generated.

By definition, the ZAP facility requires a master MMF file be designated as the reference file. This file can exist on any node in the network. All additions, deletions, and modifications *must* be made to this designated file *only*, preferably through the use of PeriStudio (see the *PeriStudio User's Guide* and the *Avaya Media Processing Server Series System Operator's Guide* for further information).



ZAP requires the presence of the `/etc/vpsrc.sh` file on every node that is synchronized. This file is usually present as part of the standard MPS installation.

ZAP and MMF files on the MPS

In an MPS system, when ZAP updates any MMF file, it is required that there exists a copy of that MMF file for each component in the system. It is recommended that a directory be created for each of the MPS components on the MMF partition and all the files, that ZAP operates on, be duplicated under these directories. Make sure that the `/opt/vps/mpsN/etc/vmm-mmf.cfg` files on the system are updated to reflect the change in the file locations.

For example:

On an MPS 500 (with components `mps1` and `mps2`), the MMF “myPrompts” needs to be updated periodically by ZAP. Hence, the following directories must be created:

```
/mmf/mps1
/mmf/mps2
```

The MMF “myPrompts” must be copied into each of these directories. The files `$MPSHOME/mpsN/etc/vmm-mmf.cfg` must have the following line added:
`mmfload /mmf/mpsN/myPrompts`



Ensure that any previous references to the MMF in vmm-mmf.cfg file are removed.

MMF Abbreviated Content (MAC) File

For any form of synchronization, an MMF Abbreviated Content (MAC) file is created from the designated master file and placed into the `$MPSHOME/common/zap/distribution` directory of the reference node and `$MPSHOME/common/zap` directory of the target node(s). By default this file uses the base name of the reference MMF file that is specified. The `-m <mac_name>` option allows a pre-existing MAC file to be specified during an update (where **mac_name** indicates the path and name of the file).

The MAC file is compressed to reduce the time and load the transfer places on the network. It uses attributes and a 32-bit Cyclic Redundancy Checking (CRC) value for each element in the reference MMF file to compare it to the target MMF file. This 32-bit CRC value represents the elemental data without having to actually store the data in the MAC file. Thus, the MAC file is much smaller in size than its MMF file counterpart.

The MAC file is decompressed when verification commences. The verification process compares each element in the target MMF file against its counterpart in the MAC file, and consists of a comparison of each element's attributes followed by its 32-bit CRC value. If either of the comparisons is found to be inconsistent, the element is flagged as requiring an update: after all comparisons are completed, these elements are downloaded from the source and updated on the target. Conversely, the target MMF file is also checked for elements that were not found in the MAC. In this case, the extraneous elements are deleted from the target file.

If multiple element names exist with the same encoding, ZAP only uses the first element with the duplicated name and encoding from the source MMF file to update the target MMF file. This is due to the fact that VMM only uses the first item in the source MMF file with a particular name and encoding as a reference; therefore, only this first element needs to be updated and maintained. The element which appears first in the target MMF file (i.e. the element with the lowest EAP number) is updated; however, none of the remaining duplicate elements is updated. A warning is placed into the update results log file indicating that multiple elements with the same encoding are present in the MMF file (see [Log Files on page 188](#)).

If duplicate element names with different encodings exist in an MMF file, only one copy of an element is added to the target MMF, and this element is the one in the source that has the highest EAP number. The condition caused by duplicate element names can be eliminated by assigning unique names to all elements within an MMF file.

The following paragraphs offer suggestions for running ZAP, though the modes are not mutually exclusive (that is, either form can be used in either instance).

Basic Implementation (Low Volume/Traffic)

In environments where network traffic saturation is not a concern or there are few

MPS' or only one node in the system, ZAP can be run directly from a command line without any other intervention. To initiate the facility for all activated instances of an MMF file, use the command line syntax **zap <mmf_name>**, where **mmf_name** indicates the path and name of the reference MMF file. The facility must be initiated from a command line of the node on which the reference file resides.

By default, all nodes and MPS' listed in the `$MPSHOME/common/etc/vpshosts` file on the reference node are addressed. This is called *distributed synchronization*, where the synchronization of the target nodes is scheduled in groups of up to ten, with each group having its synchronization starting one minute apart. This staggered scheduling helps to limit use of network bandwidth during the data transfers.

Command Line Options

To specify the nodes and MPS' that are actually zapped, as opposed to all those located in `$MPSHOME/common/etc/vpshosts`, a user-defined file is created in the same format as the `vpshosts` file and used in place of it. This file can be located anywhere on the reference node. To use this option, specify the `-f` switch followed by the alternate file name (if located in the current directory) or the path and alternate file name.



The alternate file used with the `-f` option *must* be in the same format as the `vpshosts` file. As a suggestion, make a copy of the `vpshosts` file, edit it to include the desired entries, then save that file with the alternate name. *Do not overwrite the existing `vpshosts` file!*

In addition, because ZAP references the local node's `vpshosts` file to determine which MPS' are available to update, it is imperative that all MPS' in the entire network appear in that file (as well as the corresponding files on all remote nodes). This file equivalency guarantees that all MPS' in an alternate file also appear in the local (reference) node's `vpshosts` file.

Selective synchronization causes a specified node or MPS to be synchronized immediately. This is accomplished by using the `-n` option to specify a specific node as the target, where all active instances of the MMF on all MPS' on that node are addressed. Use the `-v` option to specify a specific MPS when only that copy of the MMF needs updating.

In instances where mixed systems have not had all target nodes updated to use the latest ZAP release or which have security in place that does not allow remote ZAP sessions to complete correctly, the `-L` option must be used to ensure compatibility. This command line option forces all applicable components on all nodes to be updated directly from the local (reference) node.



The `-L` option prevents any remote ZAP processes from occurring, thereby overriding any `zap.network.cfg` files that have been defined (see [Advanced Implementation \(High Volume/Traffic\)](#) below).

Additional command line options are included at "Synchronization (ZAP) Command Summary" on page 191.

Advanced Implementation (High Volume/Traffic)

By default, ZAP connects from a local (reference) node to all remote (target) nodes (see [Basic Implementation \(Low Volume/Traffic\) on page 178](#)). Where multiple LANs exist, which in turn contain multiple nodes that need to be updated by ZAP, network traffic is further reduced and performance improved by having ZAP function on a proxy basis. In this case ZAP updates *one* MPS for a particular node in a group (LAN): each of the other MPS' on this node, and one MPS on each of the other nodes in the group, are updated remotely from this “locally updated” (proxy) server. This functionality requires the presence of a user-defined `zap.networks.cfg` file.



The order of nodes in the `zap.networks.cfg` file determines the order in which each node acts as a proxy for its group. Analogously, the order of MPS' in each node's `vpshosts` file determines the order in which each acts as a proxy for that node. If a node or MPS is unavailable for any reason, ZAP moves to the next one in the sequence.

The `zap.networks.cfg` File

The `zap.networks.cfg` file *must* contain every node in the network since this file is used to determine the topography of the network. If a specific series of MPS' needs to be updated, the update can be instituted through use of the `-f` option (see [Individual Group Update Option on page 183](#)).

The most commonly suggested format of the `zap.networks.cfg` file is to have each LAN defined as a group; however, other arrangements are also possible, depending on site requirements. In all cases, the following syntax rules must be followed:

- Groups are defined by using the term `[GROUP]` on its own line. All nodes that follow are construed as belonging to that group until ZAP encounters another `[GROUP]` tag or the end of the file.
- Only one node is listed per line, and each node must belong to only one group.
- No empty groups are allowed, and no node can appear ahead of the first group.
- An pound symbol (`#`) precedes commented data. This symbol must appear at the beginning of a line (comments entire line) or have at least one space before it.
- Blank lines are ignored.

With these rules in mind, a sample `zap.networks.cfg` file might appear as follows:

```
# Start of zap.network.cfg file

[GROUP] #Group 1
nodeA
nodeB
nodeC

[GROUP] #Group 2
nodeD
nodeE #this node is in the middle
nodeF

[GROUP] #Group 3
nodeG
nodeH
nodeJ

#EOF
```

The `zap.networks.cfg` file must be placed into the `$MPSHOME/common/etc` directory. If the file is built so that every LAN is its own group, only one MPS on one node in each group is updated directly, with the remainder in that group being updated by this node remotely. Using the sample file [shown above](#), and given that ZAP was started on `nodeA`, one MPS on one node in group 2 and one MPS on one node in group 3 is updated via network traffic; each of the other nodes in the groups are updated on a localized basis by this initial MPS. Group 1 contains the local node (`nodeA`), and so does not require any network-wide update; instead, all MPS' in this group are updated by `nodeA`. Only the MPS' listed in the `vpshosts` file on `nodeA` are addressed. If any node in any group contains MPS' that are not in this file, those servers are not updated.



Though the `vpshosts` files on remote nodes can in theory have more MPS' listed than that on the reference node (these others do not get updated), in practice they should *never* have *fewer* than those of the `vpshosts` file on the reference node.

Individual Group Update Option

To update all MPS' on all nodes in a group, use the **zap -G <group_number>** option. This causes ZAP to update the MPS' it finds in the reference node's `vpshosts` file for nodes defined for the group. For instance, if the `zap.network.cfg` file contained the following:

```
# Start of zap.network.cfg file

[GROUP] #Group 1
nodeA
nodeB
nodeC

[GROUP] #Group 2
nodeD
#EOF
```

and the command **zap -G 1** is issued on `nodeA`, all MPS' listed in the `vpshosts` file on node `nodeA`, for `nodeA`, `nodeB`, and `nodeC`, are synchronized in accordance with the guidelines discussed earlier.

To limit the MPS' within a group that get synchronized, issue the **-G** option in combination with the alternate (`vpshosts`) file option (see [Command Line Options on page 180](#)):

```
zap -G <group_number> -f <alternate_file>
```

In this instance, refer to the previous `zap.network.cfg` file example for illustrative purposes and assume that each node contains four MPS'. By using an alternate `vpshosts` file that contains the following:

#COMP	NODE	TYPE
1	nodeA	VPS
2	nodeA	VPS
5	nodeB	VPS
16	nodeD	VPS

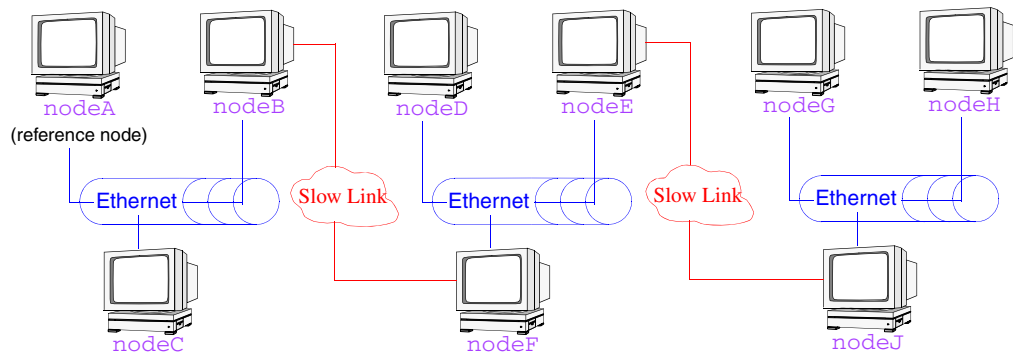
the command **zap -G 1 -f alternate** only synchronizes MPS numbers 1 and 2 on `nodeA` and MPS 5 on `nodeB`. Notice that MPS 16 on `nodeD` does *not* get synchronized because that node does not belong to group 1.

Using Multiple zap.network.cfg Files

In general the zap.network.cfg file exists only on the reference node. This requires that the initial update for each group travel over network pathways. If slow or ineffective links exist within these paths, overall system performance can be adversely affected. To circumvent these deficient links, additional zap.network.cfg files are defined on the remote nodes.

The additional zap.network.cfg files must be defined differently from those on the reference node. If the remote nodes contain the exact same file as that of the reference node, ZAP behaves the same way as if the additional configuration files did not exist.

This functionality is illustrated in the following example. The network topography contains three LANs with two slow links between them.



In this example there are two zap.network.cfg files: one is located on nodeA, the reference node, and the other on nodeD, nodeE, and nodeF. There is no file on the remaining nodes. (This format should not be construed as a requirement; rather, further customization can be made by using various file location configurations.) The files are defined as follows:

nodeA	nodeD, nodeE, and nodeF
[GROUP] #reference node	[GROUP] #secondary zap file
nodeA	nodeD
nodeB	nodeE
nodeC	nodeF
[GROUP]	[GROUP] #other group
nodeD	nodeG
nodeE	nodeH
nodeF	nodeJ
nodeG	
nodeH	
nodeJ	

If the `zap.network.cfg` file existed only on `nodeA`, and each LAN were its own group, the reference node would have to update one MPS in each LAN, requiring it to travel over a total of three slow links (one to `nodeD`, `nodeE`, and `nodeF` and two to `nodeG`, `nodeH`, and `nodeJ`). With the example scenario in place, the reference node updates one MPS on `nodeB` and `nodeC`, then tries one MPS on each of the other nodes *in the order they appear in its `zap.network.cfg` file*. ZAP detects that there is another `zap.network.cfg` file on `nodeD` (or `nodeE` or `nodeF` if one of the other nodes fails): instead of `nodeD` updating one MPS for every node in its group as defined on `nodeA`, it updates one MPS for every node in its group and one from the other group as defined on `nodeD` (see [nodeD](#), [nodeE](#), and [nodeF](#) on page 184). Initial processing time may be slower because the nodes in the latter group are not updated until one MPS in that of `nodeD` completes (as opposed to parallel processing ZAP normally uses): however, overall processing time and network congestion are reduced substantially since the number of times ZAP would have had to travel over the slow links is also reduced. Though this example uses a very basic model, the savings becomes substantial on systems of greater complexity.

Updating a Specific Element

By default ZAP compares each target MMF with the designated MMF on the reference node and transmits to each one those elements which are different. In instances where the element that has changed is known, ZAP can be directed to update *only* that element and ignore any other comparison of the file. This increases significantly to the speed at which ZAP functions.



In this case, instead of updating one MPS per node and *then* executing other remote instances, ZAP copies the file created from selected element(s) to the remote node and executes a remote ZAP on all MPS' on that node.

To update a specific element, use the `-e` option in the following manner from the node that contains the updated element:

```
zap -e {@ <EAP_number> | <"Element Name">} <mmf_name>
```

If specifying an element name that contains spaces, it *must* be enclosed in quotes. This ensures that the variable is passed as one argument to ZAP. If there are no spaces in the element name, the quotes may be omitted. Multiple element names and/or EAP numbers are stipulated through multiple `-e` arguments.

As an example, an MMF file named `Talk2Me` contains the following elements:

EAP#	Element Name
1	Welcome Message
2	Salutations
3	Goodbye Message

To update the second and third elements from this reference file to all other nodes in the `vpshosts` file of the local machine, issue any of the following commands:

```
zap -e Salutations -e "Goodbye Message" Talk2Me
zap -e "Goodbye Message" -e @2 Talk2Me
zap -e @3 -e Salutations Talk2Me
zap -e @3 -e @2 Talk2Me
```



Though the *location* of the `-e` option in the command is imperative, the *order* that multiple elements appear is not.

Additional ZAP command line options may be used as well. The following examples show, in order and on a limited basis, how to update these elements on all the MPS' on only the node named `womquat`; on only MPS 11; on all the nodes in the alternate `vpshosts` file named `usethisone`; and on all the nodes in group 3 of the `zap.network.cfg` file defined on the local node. Other options can also be used and combined depending on the complexity of the situation.

```
zap -e Salutations -e @3 -n womquat Talk2Me
zap -e @2 -e "Goodbye Message" -v 11 Talk2Me
zap -e @2 -e @3 -f usethisone Talk2Me
zap -e Salutations -e "Goodbye Message" -G 3 Talk2Me
```

Consolidating Multiple Element Updates

When multiple elements in an MMF file need to be updated, and use of the option documented above becomes unwieldy, use the `-E <filename>` option instead. (If this file is not located in the current working directory, the path to it must also be included.) The plain text file must adhere to the following conditions:

- elements may be listed by EAP number, name, or a combination of both
- elements numbers must be preceded by the @ sign
- elements containing spaces in the name must *not* use quotes
- each entry must be listed on a separate line

For instance, `zap -E thisfile Talk2Me` updates only those elements found in the file named `thisfile`, for the MMF file named `Talk2Me`, on the nodes listed in the `vpshosts` file of the reference node. Other options can also be used and combined with this one depending on the complexity of the situation



Do not use the upper case `-E` option with the lower case `-e` option: these two must not be combined.

Exception Processing

If a remote node fails to respond or a MAC file cannot be transferred, an attempt is made at a later time to retransmit the file. The number of retries is preconfigured at three, but may be specified otherwise using the **-r** option. The time interval between retries is likewise preconfigured and can be changed through use of the **-d** option: the default is 30 minutes, but at a minimum this interval must be set to ten minutes. It is also possible to schedule a date and/or time for the synchronization to take place. This is accomplished by using the **-t** option. These options can be used individually or in combination.



When using ZAP with groups (see [Advanced Implementation \(High Volume/Traffic\) on page 181](#)), the retry count specifies the maximum number of retry attempts made directly by the reference node.

Inconsistencies detected during synchronization are resolved by either deleting extraneous elements from the target MMF file and/or downloading all unreferenced elements in the MAC file from the reference node and adding them to the target MMF. Any errors recorded during this process are added to the update results log file. Every individual update and delete request is processed regardless of whether or not an unsuccessful operation has occurred during the procedure. The procedure is performed on all target nodes independently, and can proceed simultaneously or individually, depending on how the process was initiated (see [Basic Implementation \(Low Volume/Traffic\) on page 178](#)).

The **-A** command line option enables ZAP to generate an alarm upon completion of synchronization on each MPS, regardless of whether an attempt was successful or not. This alarm contains one of the statuses found in the following table.

Completed	Synchronization was successful
Failed	Synchronization and all retry attempts were unsuccessful.
NotActive	The MMF file on the particular MPS was not active and therefore could not be synchronized.
Terminated By User	The process was killed by the user, either by pressing CTRL-C, issuing a kill <PID> command, or by some other means.

Each time *any* MPS finishes ZAP processing with a status of Completed or NotActive, it generates an alarm message. If processing on *every* MPS on a node fails or succeeds, only *one* alarm message is generated for the entire node. If *every* node in a *group* experiences failed processing, one alarm message is displayed for *each* node in the group.

The synchronization alarms instituted by the **-A** option appear in the following format:

```
ZAP: Sync of [<mmf_name>] on [<target_node.mps#>]
has completed with status [<status>].
```

where `<mmf_name>` is the base name of the MMF file that was to be updated, `<target_node.mps#>` identifies the node (as listed in the `vpshosts` file of the node on which the MMF file update is attempted) and MPS number on which the synchronization attempt took place, and `<status>` is the end result in accordance with [those listed previously](#).



If the `.mps#` portion of the alarm text is omitted, the status message refers to all selected MPS' on the target node.

For example, a successful synchronization of the MMF file named `test_mmf` on MPS number `238` located on the node identified as `is29538` appears in the Alarm Viewer as follows:

```
Fri Oct 16 15:14:13 <console> 02040 Severity 1
ZAP: Sync of [test_mmf] on [is29538.238] has
completed with status
[Completed].
```

Log Files

Several log files are generated during ZAP execution. These log files are stored in the `$MPSHOME/common/log` directory of the reference node, and can be viewed using any ASCII text editor. Administration of all ZAP log files (i.e. when the files should be removed) is left to the discretion of the user. The files are generated on an individual basis or can be combined (see [Consolidation of Log Files on page 190](#)).

After synchronization retries are exhausted, an error message is displayed on the console and entered into the synchronization distribution log file. This log file is generated by the node originating the synchronization request with the name `zap.distribute.refnode.mmf_name.selected_elements.MMDDCCYY`, where `refnode` is the name of the node originating the synchronization request, `mmf_name` indicates the base name of the reference MMF file, `selected_elements` is the name or EAP number of the element(s) that have been selected for updates, and `MMDDCCYY` indicates the date the file was generated. In addition to errors encountered during the synchronization process, this file contains information regarding the distribution and completion status for all MMF file synchronization requests. It also contains information on which nodes were *not* notified of the updates and the reason thereof. If a `zap.network.cfg` file were present but incorrectly formatted and thus not usable, this error is entered into the log file as well. This file is appended to and never overwritten. A new file is created on a daily basis for each unique MMF file and selected elements that are synchronized during the day, and all log files for previous days are left intact.



If all elements within an MMF have been selected for updating, the `selected_elements` portion of the log file name appears as `ALL_ELEMENTS`.

If an elemental comparison finds inconsistencies between the MAC and target MMF files, the MMF file is considered inconsistent and the errors are logged to the update results log file. This file is named in the format `zap.results.target_node.mps#.mmf_name.selected_elements`, where `target_node` is the name of the remote node where the synchronization has occurred, `mps#` is the number of the MPS on which the target MMF file is located, `mmf_name` indicates the base name of the reference MMF file, and `selected_elements` is the name or EAP number of the element(s) that have been selected for updates. The file also contains information on modifications made to the MMF file. This log file is generated by the remote (target) node: each target node and MPS with an active MMF file has its own corresponding log file. The file is appended to and never overwritten, but is automatically renamed as `*.bak` (backup) when it reaches its predetermined size of 100K, and a new file created.

Upon completion of the **zap** process, all synchronized MMF files contain identical elements and data, even though the elements may be stored at different positions within the files. This result is known as *logical equivalence*. The synchronization status log file contains the state of the synchronization process for each target node. The file naming convention exists as `zap.status.refnode.mmf_name.selected_elements.MMDDCCYY`, where `refnode` is the name of the reference node, `mmf_name` indicates the base name of the reference MMF file, `selected_elements` is the name or EAP number of the element(s) that have been selected for updates, and `MMDDCCYY` the date the file was generated. A new file is created daily for each unique MMF file and selected elements synchronized during the day, leaving the status of all prior days intact.

The `zap.debug.log` file contains a history of each instance of ZAP processing initiated from the node. This file is most often used by Avaya to troubleshoot unexpected results that may occur, and can be used for informational purposes by customers as well. The file is appended to and never overwritten, but is automatically renamed as `*.bak` (backup) when it reaches its predetermined size of 1 MB, and a new file created.

If a ZAP process is terminated during execution by pressing CTRL-C or issuing a **kill <PID>** command, ZAP attempts to update the applicable log files and delete any temporary files that may have been created during processing. If ZAP is terminated with the **kill -9 <PID>** command (*highly discouraged*), these temporary files are not removed. If, after terminating an instance of ZAP, future attempts at using the utility fail, all files named `/tmp/zap.*` must be removed from the local and all remote nodes (this most often occurs when using the **kill -9** command, and is one of the reasons it is *highly discouraged*).

Consolidation of Log Files

By default log files are created whenever ZAP is used, and are never overwritten. While administration of these files is left to the discretion of the user, this can eventually lead to disk saturation if files are not off-loaded or deleted. To reduce this need for manual intervention, use the **-C** option to consolidate the files.

Use of the **-C** option must be consistent: all instances of ZAP must either use it or leave it out. When instituted, ZAP initially creates the individual log files as it would without the option: however, when the ZAP process completes, each individual file is merged into the corresponding consolidated log file. A maximum of seven files are created if using ZAP on a proxy basis; four files are created if using ZAP without proxies (these numbers do not include backup files). The `zap.debug.log` file is created as usual. The other files are consolidated into the following:

For instances of ZAP started on the local node:

- `zap.status.log`
- `zap.results.log`
- `zap.distribute.log`

For instances of ZAP that use the local node as a proxy:

- `zap.status.proxy.log`
- `zap.results.proxy.log`
- `zap.distribute.proxy.log`

Each consolidated file can reach a maximum size of 1 MB. When this limit is reached, the file is appended with a `.bak` extension and a new file created. If this new file then reaches the maximum size, it too is renamed and the previous backup file is replaced by it.



If ZAP is used without the **-E** option, then run later the same day with the option for the same command, the log file generated earlier in the day is merged along with the latest one into the consolidated log file.

Synchronization (ZAP) Command Summary

The following table contains a list of the options available to ZAP. To initiate the synchronization process, enter the **zap** command in a VSH window on the reference node (i.e., the node containing the MMF file that is used to update other instances of the file across the network).

zap	[-A -C -d <delay> -e <@ EAP# "Element Name"> -E <filename> -f <alt_file> -G <group#> -L -m <mac> -n <node> -r <retries> -t <time> -v <mps#>] <mmf_name>
-A	Specifies that alarm messages are to be generated upon completion of the synchronization request. (See "Exception Processing" on page 187.)
-C	Consolidates individual log files into merged log files. This saves disk space as well as maintenance time. (See "Consolidation of Log Files" on page 190.)
-d <delay>	Specifies the number of minutes between retry attempts. The minimum allowable value is 10 minutes. The default is 30 minutes.
-e <@ EAP# "Element Name">	Updates a specific element. If specifying an element name that contains spaces, it <i>must</i> be enclosed in quotes. If there are no spaces in the element name, the quotes may be omitted. Multiple element names and/or EAP numbers can be stipulated. (See "Updating a Specific Element" on page 185..)
-E <filename>	Uses a text file list of elements to perform updates. Eliminates the need to use multiple individual element update options (-e above). (See "Consolidating Multiple Element Updates" on page 186..)
-f <alt_file>	Optionally specifies the name of a file to be used instead of the <code>vpshosts</code> file, as the source for information about the MPS' that receive the transmitted MMF data. This option is useful if the <code>vpshosts</code> file contains entries for MPS' that should not be involved in the element transfer. The specified <code>alt_file</code> must be in the same format as the standard <code>vpshosts</code> file included with the system (see The <code>vpshosts</code> File on page 93.) If it is desired to use the system's <code>vpshosts</code> file, this option is omitted.
-G <group#>	Update all MPS' on all nodes in a group, as defined in a <code>zap.networks.cfg</code> file. This causes ZAP to update the MPS' it finds in the reference node's <code>vpshosts</code> file for nodes defined for that group. (See "Advanced Implementation (High Volume/Traffic)" on page 181..)
-L	Forces ZAP to update all MPS' on all nodes directly from the reference node. This must be used in mixed release systems have not had all target nodes updated to use the latest ZAP release or which have security in place that does not allow remote ZAP sessions to complete correctly.

```
zap          [ -A -C
             -d <delay> -e <@ EAP# | "Element Name">
             -E <filename> -f <alt_file> -G <group#> -L
             -m <mac> -n <node> -r <retries>
             -t <time> -v <mps#> ] <mmf_name>
```

-m <mac> Specifies the name of pre-existing MAC file to be used in the synchronization process. If this option is omitted, a new MAC file is created based on the specified MMF reference file.

-n <node> Specifies *selected* synchronization mode, where the **<node>** argument indicates the name of the particular node to be synchronized. If this option is omitted, by default, *distributed* synchronization is enabled, where every node in the `vpshosts` file is included in the synchronization process. If the specified node contains multiple MPS', then an attempt is made to synchronize all MPS' on the node, unless the **-v <mps#>** option is also specified.

-r <retries> Specifies the number of retries that should be attempted for each node before indicating that the synchronization attempt for that node has failed. The default is 3.

-t <time> Optionally specifies the date and time when the synchronization operation should take place. The time is specified using the following syntax:

H[H[:] [MM]] [{am|pm}] [{month day|dow}]

The time may be specified using either one, two, or four-digit numbers. A one or two-digit number is interpreted as an hour specification. A four-digit number is interpreted as an hour-and-minute specification. Hours and minutes may be separated by a colon. An **am/pm** specifier can be included. If neither **am** nor **pm** is specified, 24-hour time is assumed. If the time is omitted, the synchronization operation commences immediately.

If the date is omitted, then the current date is assumed. The date can be specified as either the month name and day number, or day of the week (**dow**) (in which case, the current week is assumed).

(See [Examples: on page 193](#) for more information.)

-v <mps#> Specifies the numeric designation of the MPS that is to be synchronized. If this option is omitted, by default, all MPS' with identification numbers in the `vpshosts` file are included in the synchronization process. If used in conjunction with the **-n <node>** option, only the specified MPS number on the specified node is synchronized.

<mmf_name> Specifies the path and name of the MMF reference file. This argument *must* be specified.

zap

```
[ -A -C
-d <delay> -e <@ EAP# | "Element Name">
-E <filename> -f <alt_file> -G <group#> -L
-m <mac> -n <node> -r <retries>
-t <time> -v <mps#> ] <mmf_name>
```



- The following examples are not meant to construe limitations on or required use of ZAP or any of its options either individually or in combination.
- Each example assumes that synchronization occurs for all active instances of the files `dtmf.mmi` and `dtmf.mmd` residing on all MPS' configured in the `vpshosts` file of the reference node, and that no `zap.network.cfg` files exist (see [Advanced Implementation \(High Volume/Traffic\)](#) on page 181).

Examples:

```
zap /mmf/peri/dtmf
```

Synchronization operation begins immediately.

```
zap -A -C /mmf/peri/dtmf
```

Generates an alarm upon completion of synchronization on each MPS and consolidates all related individual log files.

```
zap -d 11 -r 7 /mmf/peri/dtmf
```

If a synchronization fails, ZAP waits 11 minutes before retrying, up to a maximum of 7 times.

```
zap -t 11pm /mmf/peri/dtmf
```

The synchronization operation takes place either today or tomorrow, whenever 11 p.m. occurs next.

```
zap -t 5am march 11 /mmf/peri/dtmf
```

The synchronization operation takes place on March 11 at 5 a.m.

```
zap -L -m /home/run/ball /mmf/peri/dtmf
```

Synchronization occurs from the local node across the network by using the MAC file named `ball` located in the directory `/home/run`.

Troubleshooting

The information below describes how the ZAP and VMM (Voice Memory Manager) software manage synchronization events when various types of errors and exceptions occur.

Problem	Description
<i>Cannot connect to VMM.</i>	If the VMM process is not running on the remote node, the synchronization attempt is abandoned and rescheduled for a later time.
<i>Multiple threads of VMM have an activated version of this MMF file.</i>	If multiple threads of VMM have their own copies of the MMF file open with write permission, both instances perform online updates.
<i>Cannot delete an element.</i>	The failure is indicated in the update-results log file, and the next element is processed.
<i>Cannot update an element.</i>	The failure is indicated in the update-results log file, and the next element is processed. All elements that cannot be verified are updated from the reference MMF file.
<i>Remote node is down.</i>	The MAC file is not copied to the remote node. An error message is generated to the console, and the failure is recorded in the synchronization-distribution log file. The update is also rescheduled for a later time, and fails altogether after the configured number of retry attempts. (See <code>-r <retries></code> on page 192.)
<i>Network connection interruption.</i>	If the network connection goes down between the reference and remote nodes during the transfer of any file (MAC file, log file, etc.), the file is retransmitted once the connection is re-established. If the connection is down for an extended time (such that the transfer times out), the synchronization process is rescheduled for a later time.
<i>Remote system goes down while processing updates.</i>	The synchronization process is rescheduled for a later time. The elements that have already been processed for updating are not updated again at that time.
<i>Reference node goes down while creating MAC file.</i>	The synchronization process is aborted. The process must be manually restarted at a later time.
<i>Reference node goes down while distributing MAC file.</i>	The distribution attempt is restarted automatically at a later time, after the system comes back up.
<i>Insufficient disk space on the remote system to store update package.</i>	The MAC file is not copied to the remote node and appropriate error messages are generated to the console and log files. The synchronization is rescheduled for a later time, and fails altogether after the configured number of retry attempts. (See <code>-r <retries></code> on page 192.)
<i>Insufficient space on the reference system to create update package.</i>	The MAC file is not created, and appropriate error messages are generated to the console and log files. The synchronization process is aborted and must be manually restarted at a later time.

For a procedural narrative concerning `zap`, see the *Avaya Media Processing Server Series System Operator's Guide*.

Call Simulator Facility

The call simulator facilities of the Avaya Media Processing Server (MPS) Series system allow testing of various system and application functions. Primarily, this is done to test load capacity and performance under various conditions before having the system process actual phone calls. Script commands mimic actions that a caller typically performs from the viewpoint of a caller dialing into the system.

Simulation scripts are loaded and parsed into TMS memory. Once loaded, they are assigned to one or more resource sets (`rsets`), after which they can be executed. Under normal conditions, scripts execute as written: however, should a resource set the script is bound to be destroyed, or a resource required for simulation not be available in the set, the flow of execution is disrupted. In these cases, an asynchronous event containing the event that occurred, the resource set involved, and the script that generated the cause is returned to the command shell from which the simulation was initiated. For additional information on actions discussed in this paragraph, see “Command Line Interface” on page 200. For more information on resource sets and their mappings, see *The `tms.cfg` File* on page 106.

Scripts are written in ASCII format using the scripting language and constructs as described in “VEMUL Script Format” on page 195.



For best performance results, the system should be in an offline state when the call simulator commands are issued. To avoid any potential consequences, it is recommended that the system’s phone lines be physically disconnected before activating any simulator functions.

VEMUL Script Format

The VEMUL script language provides basic programming constructs that are used to generate complex script scenarios such as looping and “if-then-else” statements (see “Script Commands” on page 196). These ASCII script files can contain multiple START-END Statement blocks. This block defines the actions that a script is to execute, and is formatted in the following construct:

```
start
  <script commands>
end
```



Note that * is not a valid character when using call simulator.

If more than one START-END Statement block exists in a script, they are executed consecutively up to and including the last one. After the final START-END Statement block has been completed the script stops running. To execute the script again it must be restarted by using the `scriptcntrl` command (see “Command Line Interface” on page 200). Alternatively, particular *commands* within the script can be reiterated by using the REPEAT Statement (see “Script Commands” on page 196).

Any format-related errors in the script files are reported only when the script is loaded into the TMS. If there are syntax errors, error messages are reported to the console during the execution of the script file load command, and the file is not loaded. Scripts must be located in the `/tftpboot` directory, but do not require any file extension.

Script Control

When a script is loaded, a handle (associated alphanumeric value) is returned in response. This handle must be used for any subsequent commands involving that script. The conversion from script name as used in the load command to the assigned handle requires no user intervention. After being loaded and before being started, the script must be bound to one or more resource sets. Once bound and started, the script commences execution (see “Command Line Interface” on page 200 for these commands). Unless otherwise designed, a script runs once and then stops (see “VEMUL Script Format” on page 195). Scripts can also be arranged to loop, either infinitely or finitely, by using the LABEL and GOTO Statements (see “Script Commands” on page 196).

Configuration Parameters

Simulation configuration parameters are preset for each system during system configuration. These parameters include the maximum size of an ASCII script, in bytes, that can be downloaded, the maximum time in seconds allocated for downloading a script file, and the events which are reported to the console running the simulation. The default values for these parameters are normally sufficient and need not be changed: however, they can be queried by using the `simgetparams` command. For additional information see “Command Line Interface” on page 200.

Script Commands

The following tables list the language that is valid for use in a call simulation script. All commands must be preceded by the `start` statement and followed by the `end` statement (see “VEMUL Script Format” on page 195). Some commands (statements) can take additional arguments (see “Primitives” on page 198). These two sets of data can be combined to author scripts ranging from simple to complex (see “Example Call Simulation Script Files” on page 202).

Statements

Statement	Description
SEND	Execute a command primitive. Processing of the script stops until the statement is completed. The maximum length of the <code><command primitive></code> string as shown in the example below is 1024 bytes. Execution can occur immediately or after a period of time specified in seconds. <pre>send "<command primitive>" in <seconds> send "<command primitive>"</pre>
ASYNC	Execute a command primitive asynchronously after waiting the specified delay period. Processing of the script continues even after this statement is encountered, with the <code>async</code> command executing after the aforementioned delay. <pre>async "<command primitive>" in <seconds></pre>
IF	Standard if-then-else programming statement. <pre>if (expression) { <script command> } else { <script command> }</pre> <p>Where expression is one of either <code>\$VAR <op> <val></code> or <code>\$PASS <op> <val></code> (see Global Variables on page 198) and where <code><op></code> is any of <code>==, !=, <, <=, >, or >=</code> and <code><val></code> is any applicable value.</p>
LABEL	Specify a label which is referenced by the <code>goto</code> statement (see next). The LABEL must be a one or two-digit number. <pre>label <label>:</pre> <p>where <code><label></code> is by necessity a short integer (less than three places).</p>
GOTO	Reference a section of the script that is defined with the corresponding <code>label</code> statement. <pre>goto <label></pre> <p>where <code><label></code> is a short integer that must be defined elsewhere within the same script block (see previous).</p>
REPEAT	Repeat one or more script commands and count the number of times repeated. <pre>repeat <n_times> { <script commands> }</pre> <p>where <code><n_times></code> is the number of times to repeat the commands. <i>NOTE:</i> If no value is assigned for the count, the statement repeats (loops) indefinitely.</p>
DELAY	Delay the processing of the script for a defined number of seconds. <pre>delay <seconds></pre>

Statement	Description
Global Variables	Variables that can be defined for use within an IF statement (see IF on page 197).
\$VAR	Script global variable.
\$PASS	Tracks the current iteration through the script.

Primitives

Command primitives can be used as arguments to the **SEND**, **ASYNCH**, and "on <event> {<command>}" (**IF**) statements (see [Statements on page 197](#)). A primitive is a low-level object or operation from which higher-level, more complex objects and operations can be constructed.

Primitive	Description
MkCall	<p>Simulate a call being made into the system. The Dialed Number Identification Service (DNIS), Automatic Number Identification (ANI), or Calling Line Identification (CLI) digits can be specified. A numerical value is substituted for <rings> and must specify the number of rings that the call will actually be answered on by the system.</p> <p>MkCall <CALLING=<digits> CALLED=<digits> > <rings></p>
Wait	<p>Delay script execution for a specified number of milliseconds. Does not provide synchronization of script execution with completion of a prompt. Typically used to provide short delays between internal messages.</p> <p>Wait <milliseconds></p>
TT	<p>Simulate touchtones from the keypad of a phone. Valid values for digits to enter are:</p> <ul style="list-style-type: none"> 0-9 — Numerical digits zero through nine [xx-yy] — Range of digits from zero through ninety-nine A-F — DTMF digits A through F # — Pound keypad equivalent ~ — Any single random numerical digit from zero through nine (multiples allowed) _ — An underscore represents one random numerical digit from zero through nine or the pound character (multiples allowed) <p>TT <digits></p>
HangUp	<p>Simulates a caller hanging up, but does not cause the script scenario to end. Should be used in conjunction with the restart option either by itself or within an ASYNC statement (i.e. async "HangUp" in <seconds> restart) (see ASync on page 197).</p> <p>HangUp</p>
RcvCall	<p>Simulate the far (receiving) end of a call. The status can be returned as busy, no answer, or the call can be "answered" on the specified number of rings. Ring range can also be random in accordance with TT above.</p> <p>RcvCall <Busy NoAnswer <rings> ></p>

Phone Line Behavior During Simulation

The call simulation functions are designed to exercise all normal higher-level software paths in the system (above the most basic hardware interface functions). No physical phone line connections or actual DTMF tones need be present. While a call simulation script file is executing, the phone line software does not alter the line's on-hook or off-hook state (e.g., to answer a simulated call or to execute a **busy** command), and performs only the associated software state changes and messaging functions (e.g., the voice path to the line is active). All lines undergoing call simulation should be placed in a *busy* state prior to starting the call simulation.

Call Simulator Conditions and Usage

The system software and the applications are unable to distinguish call simulator traffic from real telephone traffic. However, all call simulator driven traffic has some regular pattern as configured in the script file(s). For the most part, internal conditions occurring while the call simulator is running are indistinguishable from real conditions, except for the intensity and constancy of the workload.

It is possible to drive the system too hard by using unrealistic call lengths or by executing application paths that normally account for only a small percentage of the total calls at any time. Be aware of the call scenarios that have been created and used while observing and/or extrapolating call simulator results. Use only realistic call patterns and leave a few phone lines for actual calls (to be made from within the testing environment) to observe the overall system response and behavior from a real caller's viewpoint.

Command Line Interface

Call simulation functions are performed from the command line of the Diagnostics, Logging, and Tracing (DLT) process (see [dlt on page 53](#)). To connect to DLT, open a command window on the node you wish to monitor and enter the `dlt` command. Connections to TRIP and TCAD are attempted: if these connections are successful, the `dlt` prompt appears in the command line. The following commands can be entered from that point (see the *DLT Commands* section in the *Avaya Media Processing Server Series Command Reference Manual*).

Command	Description
<code>scriptload</code>	<p>Downloads a script into TMS memory. Before a script can be loaded it is parsed to check command formats. If errors are discovered the script is not loaded. Likewise, if the script passes the parsing but there is not enough memory available, the script is also not loaded.</p> <p>A script load response event is generated immediately upon a successful load or if an error has occurred during the load process. In addition, a script handle is returned which must be used in all subsequent commands referencing the script (see Script Control on page 196).</p> <p><code>scriptload <script_name></code></p>
<code>scriptbind</code>	<p>Binds a script to a resource set (line). When a script is bound to a resource set, a mapping is created that is used to deliver the requested commands to the resources in that set when script commands are run. A script must be bound to a resource set, or multiple resource sets (one at a time), before it is run.</p> <p>A script bind response event is generated immediately upon completion of the bind or if an error has occurred in the process.</p> <p><code>scriptbind <rset_handle> <script_handle></code></p> <p>where <code><rset_handle></code> is a line number and <code><script_handle></code> is that generated by the <code>scriptload</code> command.</p>
<code>scriptcntrl</code>	<p>Controls execution of scripts in the system. In order to execute a script must first be loaded and then bound to a resource set. Once this has transpired it can be started and stopped using this command. Starting a script that is already running or stopping one that is already stopped has no effect on it (in either case the status is returned as okay).</p> <p>A script control response event is generated immediately upon applying this command.</p> <p><code>scriptcntrl <rset_handle> <script_control></code></p> <p>where <code><rset_handle></code> is a line number and <code><script_control></code> is a numerical value of <code>1</code> to start a script or <code>0</code> to stop it.</p>
<code>scriptlist</code>	<p>Provides information about the scripts that have been loaded. This information includes the name of the scripts, whether they have been bound and how many instances are so, the lines they are bound to, and their statuses.</p> <p>A script list response event is generated immediately upon collection of all the required data.</p>

Command	Description
---------	-------------

scriptlist

```

cmdtool - /bin/csh
dlt#vps.31,vos/tms3000 {40} -> scriptlist
dlt#vps.31,vos/tms3000 {41} ->
Status: 0
Loaded Scripts          Bound?  How many?
jdztestsim             Yes     3

Line#      Script Name      State
1          jdztestsim      Idle
11         jdztestsim      Running
23         jdztestsim      Idle
dlt#vps.31,vos/tms3000 {41} ->

```

scriptunbind

Removes the binding of a script to a resource set (line). After a script has been unbound from a resource set, it can no longer be executed.

A script unbind response event is generated immediately upon completion of the unbind or if an error has occurred in the process.

scriptunbind <rset_handle>

where **<rset_handle>** is a line number the script was bound to.

scriptunload

Unloads simulation scripts from TMS memory. In order for a script to be unloaded, it must first be stopped (see **scriptcntrl** on page 200); otherwise, an error code is returned. Scripts should be unbound prior to being unloaded (see **scriptunbind** above); however, if a script is presently bound to a resource set when this command is executed, it will first be unbound and then removed from TMS memory.

A script unload response event is generated immediately upon completion of the command or if an error has occurred in the process.

scriptunload <script_handle>

where **<script_handle>** is that generated by the **scriptload** command (see page 200) or **all** to unload all scripts.

simgetparams

Returns the simulation parameter settings for the TMS. Parameter settings are returned in the same format as is used to set them.

A simulation get parameters response event is generated immediately upon collection of all the required data.

simgetparams

Example Call Simulation Script Files

Two sample call simulation files are shown below. The first listing illustrates a simple looping script. The second is more complex and includes two different scenarios that run randomly, as well as a randomly timed asynchronous disconnect and script restart. For additional information regarding commands used in these scripts, see “Script Commands” on page 196.

```
start
  label 123:
    Delay 1~                # Wait 10-19 seconds before starting
    send "MkCall 1"         # Simulate a call (ring once)
    Delay 3                 # Wait 3 seconds
    send "TT 2"             # Enter (touchtone) 2
    Delay 5                 # Wait 5 seconds
    send "TT 9876#"        # Enter (touchtones) 9, 8, 7, 6, and

    Delay 7                 # Wait 7 seconds
    send "HangUp"          # Simulate a disconnect
    goto 123                # Go to beginning (label) of script
end
```

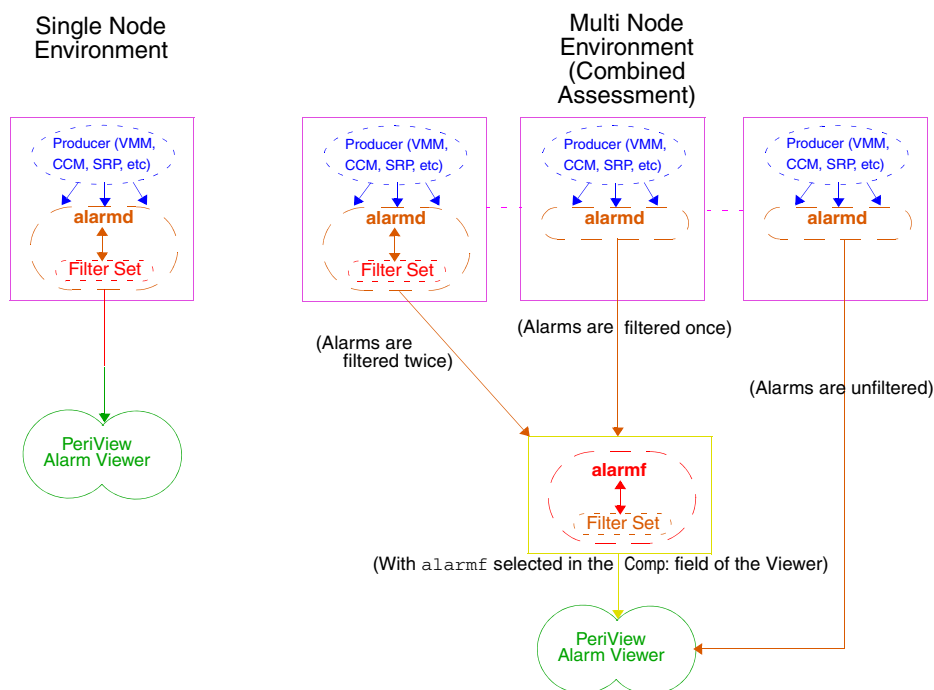
```
start
  asynch "HangUp" in ~~    # Hang up in 00-99 seconds

  label 123:
    Delay 10                # Wait 10 seconds before starting
    send "MkCall 1"         # Simulate a call (ring once)
    Delay 2                 # Wait 2 seconds
    $VAR = [0-1]           # Global variable set to 0 or 1
    if $VAR = 0 {          # Initial "if" statement, executed
                          # only if variable is 0
      send "TT 2"          # Enter (touchtone) 2
      Delay 5              # Wait 5 seconds
      send "TT 9876#"     # Enter (touchtones) 9, 8, 7, 6, and
                          # the octothorp
      Delay 7              # Wait 7 seconds
    }
    else {                  # Secondary "if" statement, executed
                          # only if variable is NOT 0
      repeat 3 {          # Repeat the following commands 3 times
        send "TT ~_"      # Enter random 2 digit touchtone,
                          # which may include an asterisk
                          # or octothorp in
                          # the second position
        Delay ~           # Wait from 0 to 9 seconds
      }
    }
    send "HangUp"          # Simulate a disconnect
    goto 123                # Go to "label" in script and generate
                          # another call
end
```

Alarm Filtering

Typically alarm filtering is performed through details stipulated at the PeriView Alarm Viewer; however, this still requires that all alarms generated be forwarded to the viewer. In cases where alarm production is excessive, traffic overhead can place an undue burden on the system, thereby degrading overall performance. In order to avoid a situation like this, alarm filtering has been instituted at a lower (daemon) level, allowing for a boost in system performance and/or distinct customizing to meet site and customer needs.

Filtering can be accomplished through either the **alarmd** or **alarmf** daemon, or both (**alarmf** can also be run as a utility - see the latter portion of “Notation Functionality” on page 207). Generally in a single or limited node environment filtering through **alarmd** is sufficient. However in situations where there are many nodes in a network, each running an instance of **alarmd**, or where alarm traffic has become substantial, further consolidation of filtered alarms can be performed by **alarmf**. Depending on system needs and configuration, there can be present either one or multiple instances of **alarmf**. This can reduce the overall number of alarms ultimately delivered to the viewers, and depending on the actions stipulated by the user, can reduce the frequency with which system operators or administrators are required to interact with the system. Any number of filtering criteria can be applied in situations where multiple instances of the daemons exist, and filtered and unfiltered daemons can be combined as well. Additional details on these scenarios are provided in the sections that follow, and at “Filtering Examples” on page 213.



Alarm Filtering Conceptual Diagram

For information on the PeriView Alarm Viewer, please see the *PeriView Reference Manual*. For additional information concerning the alarm daemons, see [VOS](#)

[Processes on page 39.](#)

Filtering Precepts

In general, alarm collection is under control of the process **alarmd**. By default, **alarmd** passes all arriving alarms to any connected alarm viewers. However, it can also be configured to use a filter set to initiate certain actions or to discard alarms which satisfy defined filter criteria. Though this filtering limits the number of alarms forwarded to the viewers by **alarmd**, each instance of the daemon still takes this action; thus, in instances where there are many nodes in a network, repetitious routing is still likely. In situations like these **alarmf** can be configured to further perform filtering action for one or more of the **alarmd** daemons; full potential for this is exacted by having **alarmf** execute as close to the **alarmd** daemons as possible. In the case of multiple geographical locations this includes having at least one instance of **alarmf** present per site. In addition **alarmf** can be run as a utility to verify and test alarm filters, view log files, or check associated statuses (see [alarmf Command Line Options on page 206](#)). In instances such as these **alarmf** executes the command and exits. When run as a daemon, it connects to the specified **alarmd** daemons and continues active processing. To be run as a daemon **alarmf** must be started in the `$MPSHOME/common/etc/gen.cfg` file (see [The gen.cfg File on page 96](#)). This allows it to be recognized by SRP and displayed as a selection in the Comp: field of the PeriView Alarm Viewer Alarms Filter window. You must select the **alarmf** daemon (or daemons) whose alarms (which pass its filtering criteria) you want to view.


A filter set is specified either by its full path name or by its file name if it resides in the `$MPSHOME/common/etc` subdirectory. Filter sets, though standard ASCII files, should be appended with the `.flt` extension. Filter sets are added or cleared from the daemon through VSH interface commands (see [Command Line Interface on page 200](#)), or loaded at boot time by adding the appropriate line to the `$MPSHOME/common/etc/alarmd.cfg` or `alarmf.cfg` file (see [The alarmd.cfg and alarmf.cfg Files on page 99](#)). Only one filter set can be loaded at a time.

When a new alarm arrives, **alarmd** invokes the filter set, which includes one or more filters. Each filter can trigger certain actions if the alarm passes the filter. If **alarmf** is also configured on the system in relation to **alarmd**, it receives the alarm and performs its own filtering action. The alarm is then pushed to all attached alarm viewers if at least one filter returns a value of `true`. (If **alarmf** is not configured on the system, this process occurs directly from **alarmd**.) This value is returned for any filtering criteria that is met by the alarm; if no match is made, the value is returned as `false` and the alarm is not forwarded to the viewer. Alternatively, if the function **discard()** has been invoked as an action, the alarm is discarded and not sent to any viewer. For additional information concerning conditions and functions, see [Notation Functionality on page 207](#).

Command Line Interface

As discussed earlier, alarm filtering can be invoked upon system startup by specifying the necessary arguments in the `alarmd.cfg` or `alarmf.cfg` file (see [The `alarmd.cfg` and `alarmf.cfg` Files](#) on page 99). Filtering can also be invoked from the VSH command line of the common component for that node. This includes configuration, maintenance, and monitoring of the filtering aspects of the daemons. In addition, since only one filter set at a time can be active per daemon, and there may be instances where a different filter set needs to be invoked while the system is running, filters can be cleared and changes made from this interface.

The following table presents the command line options available for **alarmd** and **alarmf** filtering capabilities (and where **alarm*** indicates either daemon):

Command	Description	Available to
addflt	Loads or reloads a new or modified filter set and activates it. If this command is used while another filter set is already active, the new command loads that filter file in place of the existing one.	Both
alarm* addflt <filter_set>		
clearflt	Disables the previously loaded filter set. Use this command to halt specified alarm filtering or before loading a different filter set	Both
alarm* clearflt <filter_set>		
status	Shows the current status of the daemon, including whether alarm filter sets and logging are enabled, and which viewers are attached.	Both
alarm* status		
lsize	Specifies the size of the log file in kilobytes. The default size is 100K; the minimum value this can be set to is 1K.	alarmd
	The maximum size the log file can be set to is exceedingly large. Take care not to use up system resources and space by setting this value too high.	
alarmd lsize <nnn>		
where <nnn> is the number, in kilobytes, of the size of the log file.		
nolog	Turns alarm logging for a specific component on or off.	alarmd
alarmd < <no>log > <comp_type.number>		
where <nolog> turns logging off for the component type (<comp_type>) and <number> specified in the command, and where <log> turns it back on.		

alarmf Command Line Options

The following options apply only to **alarmf** and are used from a VSH command line of the common component for the node:

alarmf	[-b] [-d start<-end>] [-f <name>] [-H] [-h <hostname all>] [-I <filter_expression>] [-l <logfile>] [-t] [-T <poll_rate>]
-b	Debug mode.
-d start_date <-end_date>	Time interlude filter for viewing log file. All alarms outside the date intervals are not displayed. Format of date interval is <code>yyymmddhhmmss</code> ; hours, minutes, or seconds can be omitted, in which case zeroes are assumed, but leading zeroes in year, month, and day cannot be left out. If a -end_date interval is not provided, output will be up to end of log file. Used in conjunction with the -f or -I and -l options.
-f <name>	Name of filter file to invoke.
-H	Help (options usage) file.
-h <hostname all>	Host name(s) to connect to alarmd with when run as a daemon. By default, alarmf connects to all nodes listed in the <code>vpshosts</code> file of the node from which it is invoked. Use all to reconnect to these nodes if individual nodes had been specified in the interim via the hostname option.
-I <filter_expression>	Filter text to be processed from the command line rather than a <code>*.flt</code> file. See Notation Functionality on page 207 for proper syntax/format.
-l <logfile>	Name of log file to be viewed. The location of these files is assumed to be <code>\$MPSHOME/common/log</code> unless a full path to another location is provided. Used with -d and -f or -I options.
-t	Displays a list of alarm producers (processes sending alarms to either daemon).
-T <poll_rate>	Rate, in seconds, to poll nodes for missed connections. Default is 1 second.

Notation Functionality

Alarms are generated by the system in the following format:

```
Mon Jan 3 08:51:17 <srp> 12008 Severity 5 Comp
#mps.55/raven Line 23 Host 2 sysmon: disk space for
/var below LWM: capacity at 67%
```

Included in the contents of the alarm are the following fields (with their corresponding example entries in parentheses):

- Date (Mon Jan 3)
- Time (08:51:17)
- Producer name (srp)
- Alarm number (12008)
- Severity (5)
- Component name, number, and node (mps, 55, raven)
- Phone Line number (23), if any
- Host number (2), if any
- Alarm Text (sysmon: disk space for /var below LWM: capacity at 67%)

Each filter in a filter set can refer to the alarm fields by their fixed names: `producer`, `number`, `severity`, `compname` (component name), `compnum` (component number), `node`, `line`, and `host`. Filtering expressions use the following basic C-like notation:

```
( logical condition ) ? actions_if_true :
actions_if_false
```

An expression is any syntactically correct combination of symbols that represents a value. Every expression consists of at least one operand and can have one or more operators. Operands are values, whereas operators are symbols that represent particular actions (see [Logical Conditions](#) on page 209).

If no action is required, the filter should use the `true` or `false` function instead. For example, the filter

```
((producer == "srp" && severity > 5) || producer ==
"vmm") ? print() : false
```

passes all alarms generated by `srp` with `severity` larger than 5, or generated by `vmm`. The filter prints matching alarms because the `print()` action is invoked, and ignores the rest. The `print()` action returns `true`, so the filter returns `true` if this action is invoked, and also sends the alarm to all connected viewers.

Several filters, separated by a semi-colon (;), can be combined into one filter set. Each alarm is compared to all filters in the set since each filter may cause a different action. If any filter in the set executes the action `discard()`, the alarm is discarded and no further matching of the alarm is done. For example, the set

```
(producer == "ccm") ? print() : discard();
(severity > 6)? email("Something wrong", "sysadmin")
: false
```

prints all alarms from `ccm` and sends an e-mail for `ccm` alarms of `severity` greater than 6 to the email address `sysadmin` with the message `Something wrong`. If the alarm is not produced by `ccm`, it is discarded.

If a field is not specified in a filter, the match is assumed. Filter sets may also contain comments. A comment is introduced with either a `#` or a `//`, and continues until the end of the line.

To verify the syntax of a filter set, use **alarmf** as a utility by entering the **alarmf -f filter_name** command. If the filter set passes the check, a message is returned indicating this: if not, the message returned gives an approximate location of the error in the file. After corrections are made the command should be repeated to validate the efficacy of the changes.

To authenticate the performance of a valid filter file, apply it to an existing log file that contains applicable data. Use the **alarmf -l logfile -f filter_name** command to do so. Filter files can also be tested against live alarms by using the **alarmf -h hostname -f filter_name** command.



Use the `print()` action function to send the results of these checks to the console. However, once the files have been validated the `print()` function should be halted in accordance with the note at “`print(<format>`)” on page 211. Use the `clearflt` command to disable the filter (see [Command Line Interface](#) on page 205).

Logical Conditions

Filters can use the following functions to specify logical conditions:

Function	Description
intext ("str")	Returns a value of <code>true</code> if " str " is a substring of the text of the alarm.
inText ("str")	Same basic rules as above, but the substring search is case sensitive.
frequency(n, secs)	Returns a value of <code>true</code> only if at least the number of alarms indicated by n arrive in the time span (in seconds) indicated by secs . This function should always be used as the second part of a conditional expression: only the alarms that satisfy the first part of the expression are counted against the frequency. For example: <pre>(producer == "vmm" && frequency(5, 10)) ? print() : discard ()</pre> <p>prints the current alarm after receiving the fifth <code>vmm</code> alarm within 10 seconds. Since the <code>print()</code> action also returns true, the alarm is sent to connected viewers as well.</p>
previous()	Returns the number of seconds since the previous alarm passed the filter. This function can be used to prevent repeating actions caused by a flow of alarms. For instance, the filter: <pre>(severity > 8 && frequency(10,3) && previous() > 300) ? email("Flood of alarms", "sysadmin"): 0</pre> <p>sends an e-mail to address <code>sysadmin</code> only when 10 alarms of severity greater than eight have arrived within three seconds <i>and</i> the time since the previous e-mail is greater than five minutes (300 seconds). Without the use of this function the filter would have generated one e-mail for each new alarm as long as they met the other criteria.</p>
count()	Returns the number of alarms which passed the filter (normally used as part of an expression). For example: <pre>(producer == "vmm" && (count() % 10) == 1) ? print() : discard ()</pre> <p>prints and sends to connected viewers each 10th alarm produced by <code>vmm</code>.</p>

The examples shown in the preceding table use operators in their expressions. An operator is a symbol that represents a specific action. Many programs and programming languages use operators to manipulate numbers and text in sophisticated ways. The following table specifies the operators available for alarm filters and their precedence, from highest to lowest, in a logical expression:



Operator	Operation
-	Unary minus. Unary describes an operator which takes one argument. In this aspect a unary minus is used to create a negative number.
!	Logical NOT: the function which is true only if its input is false.
*	Multiply.
/	Divide.
%	Remainder.
+	Add.
-	Subtract.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
==	Equal.
!=	Not equal.
&&	Logical AND: the function which is true only if all its arguments are true.
	Logical OR: the function which is true if any of its arguments are true.
? :	Conditional evaluation/comparison.

Action Functions

A filter can utilize action functions to specify activities to perform in response to the success or failure of a logical condition. All action functions, with the exception of **discard()**, return a value of `true`. Several action functions can be combined into one filter by using the **&&** operator (see [the table above](#)), for example:

```
<alarm_criteria> ? (<action1> && <action2>) : false
```

The following action functions are available for use in alarm filters:

Action Function	Description
print (<format>)	Prints the current alarm to standard output (current console). If format is not specified, the alarms are printed as they arrive. The only acceptable formats use either one or two specifiers in the form of %s to refer to the first alarm line (one) or the alarm text (two). For example: <pre>print("You got it: %s\n")</pre> prints the phrase "You got it:" followed by the first line of each alarm meeting the filter criteria, with each "printing" separated by blank lines (since there is always an implicit new line at the end of an alarm).  This function is used primarily by Certified Avaya personnel for debugging purposes. If used to filter live alarms, an unending loop results through the interaction of the alarmd and consoled daemons.
log (<filepath <, format>)	Save the current alarm in a file. The format option has the same functionality as that in print (see above). Only one log file per filter is allowed. If only the file name is provided, it is stored in <code>\$MPSHOME/common/log</code> . For example: <pre>log("/home/marxbros/sawa.log" , "%s%s\n")</pre> saves the text of the current alarms (separated by blank lines) to the file <code>sawa.log</code> located in the <code>home</code> directory of user <code>marxbros</code> .  To log different alarms into different files, use several filters with one log per filter, i.e.: <pre><alarm_criteria1> ? log<file_one> : false; <alarm_criteria2> ? log<file_two> : false</pre>
email (<subject, address<, address>)	Sends an e-mail with the specified subject line to the address(es). The body of the message includes the alarm. Address aliases may be used if the mail is being sent internally with respect to system location. For example: <pre>email("High load alert", "sysadmin", "helpdesk@nni.com")</pre> sends an e-mail with the subject line "High load alert" and containing the alarm to the address <code>sysadmin</code> (an alias used within the originating network) and to the external address <code>helpdesk@nni.com</code> .
action (<command <, arg1><, arg2>)	Executes the specified command. Arguments may be represented by literals or field names (see Notation Functionality on page 207). For example:

Action Function	Description
-----------------	-------------

```
((producer == "mxvmt") && frequency(5, 10)) ?  
  (action("alert.sh", number, "manager") &&  
  action("alert.sh", producer, "boss")) : false
```

executes the script `alert.sh` twice if `mxvmt` sends five alarms within 10 seconds: the first instance of the script includes the alarm number and the word `manager` as arguments, while the second instance includes the producer (in this case `mxvmt`) and the word `boss` as arguments.

consolidate(alarm# <, newtext>)	Consolidates alarms meeting the filter criteria into a secondary alarm with corresponding alarm number and new text (text can be a maximum of 1024 bytes). If an alarm renders this function, it is discarded and the new consolidated alarm is used afterward (and sent to the viewers). For example:
--	--

```
((producer == "conout" && severity > 5 || producer == "trip") ?  
  consolidate(90025, "Something wrong with conout or trip")  
  : false
```

generates a secondary alarm numbered `90025` with the indicated text if the `conout` process produces an alarm of its own with a severity greater than five *or* any alarm is produced by the `trip` process (the original alarms are discarded).

discard()	Discards the alarm, and therefore does not apply any remaining filters to it.
------------------	---

To prevent **alarmd** or **alarmf** from applying additional filters, use the syntax

```
<alarm_criteria> ? <action_functions> && discard() :  
discard()
```

at the point in the set where you wish filtering to stop.

Filtering Examples

Several examples illustrating the concepts discussed earlier are included below. While providing a sense of the capabilities available with alarm filtering, they are by no means comprehensive. The flexibility inherent in the programming model allows you to develop filtering as simple or complex as required to suit your needs.

- In this example, two separate *filters* are combined into one *filter set* to log CCM and VMM alarms into separate files and send only those alarms to the viewers:

```
(producer == "ccm") ? log("ccm.log") : false;
(producer == "vmm") ? log("vmm.log") : false
```

When an alarm arrives, it is compared to the filters in the set, in order. If the process producing the alarm is CCM, the first part of the first filter is satisfied, the alarm written to its respective file, and the value returned as `true` (resulting in the alarm being sent on to the viewers). If the alarm is not from the CCM process, it fails the first part of the filter and is assigned a value of `false` by the second part. However, because a function of `discard()` has *not* been assigned, checking continues through the second filter, for the process VMM, in the same fashion as the first. If the alarm fails this check it is again assigned a value of `false` and is not sent to the viewers, and no further checking is performed. The two filters are separated by a semicolon (;).

- The following filter e-mails (internal) user `peri` and `helpdesk@nni.com` with `nriod` alarms, but does not send them on to the viewers because neither filter returns a value of `true` (even though an alarm may satisfy the producer portion of the filter, the filter itself assigns a value of `false` through the `&& false` expression). Any alarm not produced by either process is also kept from being sent to the viewers due to the secondary `false` statement. The flow from filter-to-filter occurs in the same manner as that in the previous example.

```
(producer == "mxvmt") ?
(email("MXVMT alarm", "peri")&& false):false;
(producer == "nriod") ?
(email("nriod alarm", "helpdesk@nni.com")&&
false):false
```

- In this instance, all `commgr` alarms are logged to a file located in `/home/user`, but not sent to the viewers or checked further due to the `discard()` function. All other alarms, however, are sent to the viewers because the second portion of the filter always returns a value of `true`.

```
(producer == "commgr") ?
(log("/home/user/commgr.log") && discard()) : true
```

- To have any alarms with a severity less than or equal to five *or* from SRP kept from the viewers, but all other alarms sent to them, you could use:

```
(severity <= 5 || producer == "srp") ? false : true
```

The operator `||` indicates that if *either* criteria in the first part of the filter is met, a value of `false` is assigned and the alarm is not forwarded; otherwise, a value of `true` is assigned and on it goes to the viewers.

- For this scenario, the assumption is made that an `alarmd.cfg` configuration file exists, and contains a filter set named `startfile.flt`. When the system boots up, this file is processed and the filter invoked. However, at some point in time we wish to instead activate and use a filter file named `gifile.flt`. To do so, we would issue the following commands, in order, from the VSH prompt of the common component:

```
alarmd clearflt
alarmd addflt gifile.flt
```

To go back to the original configuration, we would issue the commands again, but with `startfile.flt` in the second one. To use a filter set in a location other than `$MPSHOME/common/etc`, we would provide the path to it, and its name, in the `addflt` command. These commands can also be used to perform the same actions for `alarmf`.

Interapplication/Host Service Daemon Data Exchange

VMST (VMS)

In a multi-MPS network environment, applications running on different nodes can exchange data with each other via the local VMST (VENGINE Message Server - Extended) service daemons. VMST is an ASE software process that performs message server functions for VENGINE. It funnels VOS messages that have been translated by VAMP to VENGINE processes and service daemons. VMST interprets and supports all pre-existing VMS options, allowing scripts incorporating them to continue functioning under the present release without any modifications. For additional information, see [VMST on page 36](#).



VMST is aliased as `vms` in its SRP startup and `services` files, but should not be confused with previous (“non-extended”) versions of VMS.

In order for this functionality to be enabled, the connections between VMST daemons on the various MPS systems must be established at the time of startup. This is done by placing `vms` commands with appropriate options in either the `ase.cfg` or `gen.cfg` files, as applicable (see [Starting Under SRP on page 215](#)). In addition, appropriate port numbers for the VMST servers must be specified in the `services` file (see [The /etc/services File on page 129](#)).

The following is important information about configuring inter-VMST communication:

- In the configuration file(s), VMST-to-VMST connections are established from one VMST to another. Although the specification is made one way, it is a two-way communications channel, and the applications associated with any VMST may send messages to any applications associated with the other.
- Connections are established between the VMST processes at startup. However, when sending messages, applications must specify the destination VMST number and application line number. As soon as a connection is established, applications can send data to each other in both directions.
- To guarantee a correct connection, both VMSTs (connecting and accepting) must be started with the `-s` option and be specified with different service port numbers. If the accepting VMST is not available, the connecting VMST will retry periodically, every 3 seconds, until the connection is established.

Starting Under SRP

Previously, VMS' were only defined in the `$MPSHOME/mpsN/etc/ase.cfg` file. Under ASE 4.7.1, VMST (running in place of VMS) may also be specified in the file `$MPSHOME/common/etc/gen.cfg`. When used with a single MPS, VMST is started by SRP through the `ase.cfg` file. When used with multiple MPS' (whether real or virtual), it is started through the `gen.cfg` file.

PeriPro Interaction

To be able to pass messages to other components of the IVR system, a PeriPro application (VENGINE) or a service daemon must connect to a VMST process. This process to connect to and its host are defined by a client's command line option **-v mps_num** or **-v host:mps_num** (VENGINE and some older service daemons can also define a host name by the deprecated option **-N**). If a host name is not specified, the corresponding VMST must be running on the same node as a connecting client.

For example:

```
venGINE -v 1 ... app1
```

connects to a local VMST numbered 1, while

```
venGINE -v ablaze:1 ... app4
```

connects to a VMST numbered 1 on the host `ablaze`. Some service daemons (but not VENGINE) can take several **-v** options to connect to several VMSTs simultaneously, e.g.

```
periq -v 3 -v ablaze:2 ...
```

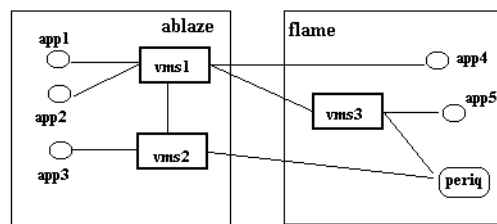
In turn, VMSTs can interconnect in order to route messages to each other. For example:

```
vmst -v 1 -v 2 -v flame:3 ...
```

specifies that VMST number 1 (**-v** option) connects to a local VMST numbered 2 and to a VMST numbered 3 running on the host `flame`.



Applications and service daemons can communicate only if they are attached to the same or to directly interconnected VMSTs.



For example, in the [preceding diagram](#), `app1` and `app2` can communicate with any other application, whereas `app3` can reach `app1`, `app2`, and `app4` but cannot reach `app5`. This is because a one-to-one relationship must exist between VMSTs, and in the case of `app3` and `app5`, a total of three VMSTs would have to be involved in the communication link. The `periq` service daemon, meanwhile, is connected to local VMST 3 and remote VMST 2, and can communicate with all the applications through these connections.

The new VMST daemon logically represents several VMS' combined in a single process. All these "internal" VMS' are considered interconnected. For example:

```
vmst -v1 -v2 -v flame:3:3 ...
```

is equivalent to two local VMS' numbered 1 and 2 connected to each other, with each also being connected to VMS number 3 on host flame.

Arguments

<code>vmst [options] -v <mps_num></code>	
<code>-b <port></code>	The base TCP/IP port on which to listen for application requests (default is none).
<code>-h <hostname></code>	This parameter specifies the host name of an alternative network interface (1e0, 1e1, etc.).
<code>-P <secs></code>	Specifies the ping timeout for monitoring remote connections (default is none). If a daemon process (e.g., SBRM) running on a remote machine generates consistent traffic, then the lack of said traffic for some time is interpreted as a sign that this daemon is down. If this option is specified, and there is no message traffic from the daemon for a time longer than specified here, VMST disconnects the remotely connected daemon if the process is identified with the keyword PING in the <code>services</code> file. (See The /etc/services File on page 129.)
<code>-p <max_num></code>	Maximum line/service port number (default is 508). VMST now allows use of MPS numbers equal to or larger than 255. All existing software works with numbers less than 255, as is (by default, VMST continues to allow phone lines/service port numbers less than 255 only). This option is not needed for large MPS numbers alone (for example, <code>vmst -v 555</code>): however, even with the <code>-p</code> option, the phone line numbers equal to or larger than 255 can not be used with real MPS' because of software limitations: they should be used with simulated MPS' only (see the <code>-v s</code> option below).
<code>-r <file></code>	Redirects debugging output to the specified file.
<code>-s <port></code>	Specifies the <code>services</code> port number to be used by VMST when it starts. (The default is the next available port taken from the range as defined in the <code>services</code> file.) Each service port number must be a valid service port for the VMST process as defined in the <code>services</code> file on each host (see The /etc/services File on page 129). This provides a specific port address that can be used by other VMSTs for the purpose of direct connection. This number should be specified in the range 1-10. If it is necessary to go beyond this limit, the <code>services</code> file must be modified accordingly.

vmst [options] -v <mps_num>



VMST may actually use a range [**port** + #_of_mps' -1] of ports all of which have to be available. For example, if the number (quantity) of MPS' (specified by the **-v** option) is 3 and **port** is set to 1, ports 1 through 3 are taken, and the next available port is 4.

-T <sec>	The reconnect timeout parameter, specifying the interval at which VMST tries to re-establish lost connections to an MPS or another VMST. Default value is 3 seconds.
-v [host:]mps_num:s<port_num>	Used to connect to another VMST or VMS running on the same or different node; s<port_num> is a <i>target</i> service port number (from \$ASEHOME/etc/services).
-v [host:]mps_num	Specifies the host and MPS number of a VMST process with which to establish a connection (i.e., this refers to the VMST <i>receiving</i> the connection). If the receiving VMST is not running on the local machine, the name of the particular host must be specified.
-v s<mps_num>	Specifies running in simulated MPS mode. In this mode, no connection to a real MPS is made.
-X {v### a}	Use -a to display internal VVPmessage message traffic, or -x v### to debug a VMST specified by number at ###.

Examples:

```
vmst -s 1 -v 1
vmst -s 2 -v 2
```

Specifies connections to MPS 1 using port 1, and to MPS 2 using port 2. The connection is made from VMST number 1.

```
vmst -s 1 -v 2 -v 1
vmst -s 2 -v 2
```

Specifies that VMST number 1 connects from port number 1 to VMST number 2 via port number 2, thus allowing bidirectional traffic between the two VMST processes.

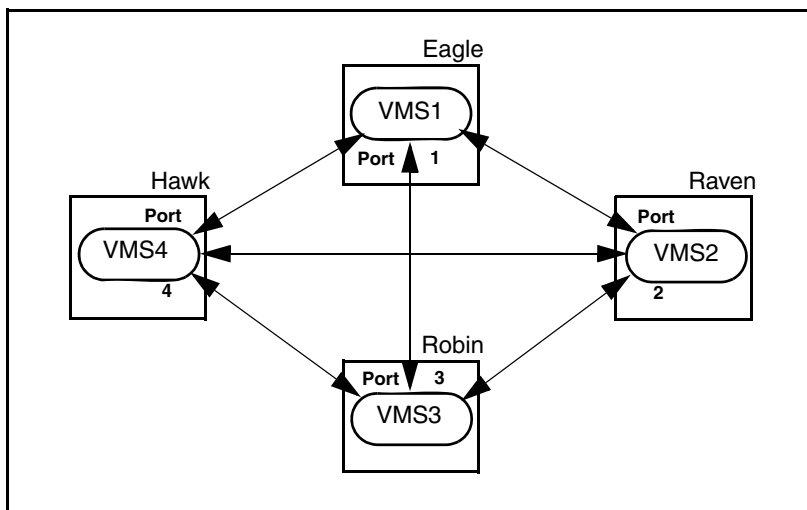
```
vmst -s 1 -v raven:2 -v robin:3 -v hawk:4 -V 1
(running on host eagle)
```

```
vmst -s 2 -v robin:3 -v hawk:4 -V 2
(running on host raven)
```

```
vmst -s 3 -v hawk:4 -V 3
(running on host robin)
```

```
vmst -s 4 -V 4
(running on host hawk)
```

Supports traffic between any of four VMSTs on different hosts. Note, each service port number is unique, numbered in the range 1-10, and each VMST is directly connected to each of the other three. The service ports have no relation to the MPS number, although both must be unique. Each service port number must be a valid service port for the VMST process as defined in `services` file on each host.



Multiport VMST Interaction

```
vmst -s 1 -v womquat:1 -v womquat:2 -v 3 -v s101 -v s102
```

The preceding example command line starts a VMST that uses port number 1 and is configured for 3 real MPS' and 2 simulated MPS'. Two of the real MPS', numbers 1 and 2, are remotely located on a node named `womquat`, while the other real MPS, number 3, and the simulated MPS', numbers 101 and 102, are local to the VMST process being run.

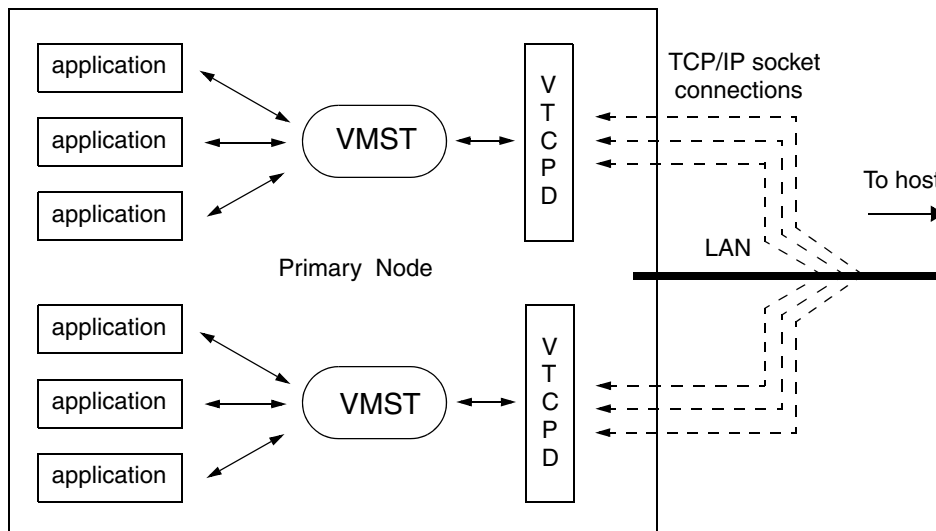
VTCPD

The *VTCPD* (*VPS Transmission Control Protocol Daemon*) process is usually executed from the `ase.cfg` file (see [The `ase.cfg` File](#) on page 148) during system startup (as opposed to being started from a command line). It provides a general way for applications to communicate with one or more external hosts through TCP/IP connections. To voice applications, the daemon functions as a resource. To perform host interaction, applications use the standard functions for acquiring, freeing, sending, and receiving resources.

The *VTCPD* resource is very flexible and can accommodate a wide variety of application requirements and host configurations. Included in these configurations are:

- connection to multiple VMST processes,
- one or more connections to a single host,
- multiple connections to multiple hosts,
- connections to yet-to-be-specified hosts and port numbers,
- one connection per line and running multiple *VTCPD* daemons with one or more hosts.

If *VTCPD* is set up for multi-host interaction, in the event of host communications failure, it can be configured to automatically switch to another host.



VTCPD Conceptual Diagram



This section documents only the configuration of the *VTCPD* daemon process. For information about application communication with TCP/IP hosts, refer to the *VTCPD Feature Description Manual*.

Single Connection to Host

The daemon can connect to the specified VMST or all VMST instances running on a node.

In its most basic configuration, at startup, the VTCPD daemon connects to a single host using the host name and TCP/IP port number that are specified as arguments on the command line. (If a host name is not specified, VTCPD is configured as a server, in which case it accepts connections on a specified port. See [Server mode on page 224](#) for more information.)

To send a message to the host, an application uses the SEND function of the Resource block in PeriProducer. When the application is expecting a response, it issues the RECEIVE function of the resource block.

VTCPD must be able to associate received messages with application requests. To do this, it uses a *Message ID*. Any request sent to the host from an application, and any response from the host to the application, must have the same message ID. This way, the daemon is able to route a received message to the correct application. VTCPD provides several alternatives for the message ID. It will manage the ID on behalf of the application or let the application assume complete control of the ID.

VTCPD can accommodate fixed or variable-length messages. The length of a variable-length message can be specified as a 1, 2, or 4-byte binary value preceding the message, or VTCPD can search for a delimiter character placed at the end of the message.

Multiple Connections to Multiple Hosts

The VTCPD daemon may be connected simultaneously to several hosts. To do this, each host must be specified on the command line when the daemon is started. No priority is associated with any particular host. VTCPD sends messages on a round-robin, load-balancing basis, and will assume that each host provides equivalent services. The application can override VTCPD and choose a specific host, if necessary.

VTCPD may make several connections to any host without giving priority to any particular connection. Messages are sent on a round-robin and load-balancing basis. (It is assumed that each connection provides equivalent services.) The number of outstanding requests on a given connection can be limited, if required by the host. If VTCPD cannot send a message to a host because this limit would be exceeded, it returns a condition to the application.

One Connection Per Line

VTCPD can also create as many connections to a host as there are lines available. In this type of configuration, each application has its own connection to the host. Therefore, there is no need to use Message IDs in this type of configuration.

Multiple VTCPD Daemons

It is also possible to run multiple VTCPD daemons connected to one or more hosts, each handling different message formats and services. The application can distinguish between services by examining the resource names.

Host Connections

Host connections are specified by the VTCPD daemon's command line options. Each connection is defined by the specified host machine name (or IP address) and the TCP/IP port number.

The VTCPD daemon can be used as a client or server. This is specified using the **-1** command option. (See [Client Mode on page 223](#) and [Server mode on page 224](#).) In a multiple-host environment, for different hosts, the daemon can support both client and server modes simultaneously, if appropriate (and different) port numbers are specified.

Client Mode

In client mode, VTCPD connects to a host as a client process. The option **-1** may be used to specify the number of connections (links) to the host using a specific TCP/IP port. The option may be used multiple times. The following syntax is used to specify that VTCPD should be run in client mode:

vtcpd -1 [#:] [host]: [port]

Args: **#** Indicates the number of connections to the specified host. (The default is 1.)

host Indicates the name or IP address of each host.

port Indicates the TCP/IP port number. (Port numbers 7000-7256 are reserved for the VMST and must not be used.)

Examples: **vtcpd -1 2:eagle:10000 -1 eagle:10001
-1 hawk:11000**

Specifies three connections to the host name `eagle` (TCP/IP ports 10000 and 10001) and one connection to the host named `hawk` (port 11000).

vtcpd -1 1::5000

Specifies a link to a yet-to-be-defined host via port 5000.

vtcpd -1 2:eagle:

Specifies two links to the host `eagle` via a yet-to-be-defined port.

vtcpd -1 3::

Specifies three yet-to-be-defined links.



In client mode, if the host or port arguments are omitted, the links are unavailable to applications until this information is supplied by an administrative application. This allows the links to be allocated dynamically.

Ports 7000-7256 are reserved for VMST and must not be used.

Correct sequencing of the connection specifications is important because applications can refer to connections by numeric designation when requesting a specific connection. The connections are numbered in the order specified on the command line. (For example, the numeric designation of the `hawk` connection as shown above is 4.)

If the number of connections is not specified, it defaults to one.

Server mode

In this mode, the VTCPD daemon accepts connections on the specified TCP/IP port. The following syntax variation of the command line option **-1** configures all connections between the daemon and the specified TCP/IP port by identifying the port number on the command line.

vtcpd -1 [#:] [port]

Args:	#	Indicates how many connections can be accepted on a given port. The daemon can support client and server modes simultaneously for different hosts (in which case, the specified port numbers must be different).
	port	Indicates the TCP/IP port number. (Port numbers 7000-7256 are reserved for the VMST and must not be used.)

Example: **vtcpd -1 3:**

Specifies three links in server mode on yet-to-be defined ports



If all connections are in server mode, the switch to the backup LAN (see [Backup LAN on page 239](#)) will not be reversed even when the LAN is down.

The VTCPD daemon always uses the server mode if a host name is not specified using the **-1** option.

If the port argument is omitted, the links are unavailable to applications until this information is supplied by an administrative application. This allows the links to be allocated dynamically.

User Datagram Mode

In both the client and server modes, VTCPD supports User Datagram Protocol (UDP) host connections. This is specified by the `-m U` command line option. In UDP mode, the host must extract the address and port number of VTCPD from the UDP message to be able to reply to applications.

vtcpd **[-m U]**

Arg:	<code>-m U</code>	Specifies UDP mode. In this mode, the host must extract the VTCPD address and port number from the UDP message to be able to reply to an application.
------	-------------------	---



In the UDP mode, the daemon usually cannot determine when the host goes down. Therefore, the application does not receive a `hostdown` condition if the host becomes unavailable. (A `hostdown` condition can only occur if VTCPD receives a corrupted message from the host.)

In UDP mode, an application can force the daemon to broadcast on the (local) network by specifying a proper IP address in the SET command. For example,

```
ISSUE SET("vtcpd")
LABEL ("host:192.84.160.255 port:5000")
```

broadcasts on subnet 192.84.160.* and port 5000.

Normally, VTCPD in a client mode uses an *ephemeral* port number, obtained from the kernel. The option

```
vtcpd ... -u port ...
```

allows the user to specify the UDP port number on which to listen for replies from the server. The option has no effect in a server mode or in non-UDP mode.

Attaching to VMST

VTCPD can attach to any specified instance of VMST running on a LAN-based host node. If the target VMST resides on another machine, the name of target node must be specified on the command line. To attach to VMST, VTCPD has to use one of the service ports specified in the *The /etc/services File* on page 129.

If there is a need to process messages with different formats, several VTCPD daemons may be attached to the same VMST. In this case, each VMST must have a unique name (defined in the `services` file) and be specified on the command line. Applications address a particular instance of VMST by its defined name.

vtcpd [-v {<name>|<#>}] -s <port_num>
 -n <name> [-m u] -Y <sec>

Args:	-v {<name> <#>}	Identifies a particular VMST to which VTCPD will establish a connection. If the VMST resides on another host node (specified by <name>), the number of the MPS must also be specified. The list of VMSTs to connect to must be specified <i>explicitly</i> by multiple command line arguments. The default is the VMST on the local node.
	-s <port_num>	Specifies the port number assigned to the target VMST. This must be a valid VMST port as defined in the <code>services</code> file.
	-n <name>	Defines a name that is used by applications to identify this instance of VTCPD. This is useful when there are multiple instances of the VTCPD process, and applications need to distinguish between them. Note that these names have to be specified in the <code>services</code> file. Primary and redundant daemons must have the same name, but have unique port numbers. VMST has to be restarted to reread this file. The default is <code>vtcpd</code> .
	-m u	Sends the <code>hostup</code> condition to the application when the host process restarts. By default, applications ignore <code>hostup</code> , but they can be coded to wait for the condition rather than loop when a host is down.
	-Y <sec>	VTCPD timeout in seconds. If a reply is not received during the specified period, the daemon is deemed to be down. Default is 5 seconds.

Example: `vtcpd -v mainnode:25 -s 160 -n hserver1`

Starts the VTCPD daemon attaching to the VMST on MPS number 25 (which is associated with the node called `mainnode`) via port 160. This VTCPD daemon is assigned the name `hserver1`.



If the `-v` option is not specified the daemon attaches to all VMST instances running on the same host node. This configuration can create a bottleneck situation and should be used with caution if the number of available host links is limited.

Message Format

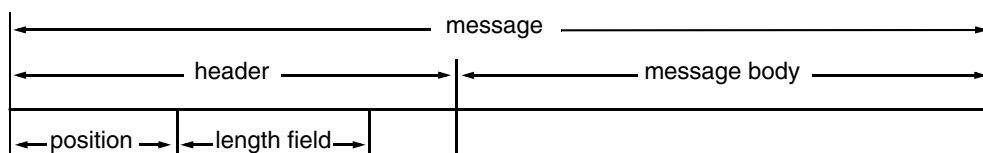
Previously, only one message format could be specified for all connections to one instance of VTCPD. However, VTCPD now supports different message formats for outgoing and incoming messages. By default, the header format is the same for both message formats as defined by the VTCPD options `-f` or `-i`. In the case of differing formats, these options are applied to the outgoing messages only, while the `-F` and `-j` options are applied to incoming messages. (See [vtcpd on page 227](#) and “Message Identification (ID)” on page 231.) The following formats are available.

Format	Description
fixed length	The message contains a fixed number of bytes.
variable length (with delimiter)	The message is delimited by a specified character.
variable length (with byte specification)	The number of bytes precedes each message. The message starts after the header. The message length does not include the size of the header.



The following is important information regarding the use of these options:

- In UDP mode, the entire message comes from a host in a single packet: therefore, the `-f` option should not be used.
- VTCPD adds a message header or a delimiter to a string received from an application. The application should not include a message header or a delimiter with the message sent to the daemon.



```
vtcpd  -f | -F {len | d:D | l:#{#|lf[:hp[:h1]]} |
        L: {w|lf[:hp[:h1]]} |
        a: {n|lf[:hp[:h1]]} |
        A: {n|lf[:hp[:h1]]}
        [{-mB | -m h | -m t}]
```



The `-F` option applies to incoming messages only, and then only when the format differs from outgoing messages. Otherwise, the `-f` option prevails.

```

vtcpd    -f | -F {len | d:D | l:#{l:f[:hp[:h1]]} |
          L:{w|l:f[:hp[:h1]]} |
          a:{n|l:f[:hp[:h1]]} |
          A:{n|l:f[:hp[:h1]]}
          [{-mB | -m h | -m t}]
    
```

Args:	len	Specifies a fixed-length format with the indicated number of bytes in length.
	d:D	Specifies the delimiter D , which can be a printable character or 0xxx format to specify the character's hexadecimal number.
	l:#	Specifies the length (in binary format) defined in bytes that precede the message (1, 2, or 4). Note that for inter-platform compatibility, the length must be specified in the order recognized by the network (big endian or most significant byte). If the length is in non-network order (little endian) then the option -mB must be specified. If the header includes data length and the header length, then use -f L:Lf .
	l:l f[:hp[:h1]]	Specifies that the (binary) length field is embedded within the message header. The length field (l f) contains the length of the body portion of the message. The header position (hp) is counted starting at 1 (which is the default). The header length (h1) defaults to l f+hp-1 . By default, VTCPD strips the header before passing the message to the application, and prepends it before sending to the host.
	L:L f[:hp[:h1]]	Specifies that the (binary) length field is embedded within the message header. The length field (L f) contains the total message length (which is the header plus body portion). The header position (hp) is counted starting at 1 (which is the default). By default, VTCPD strips the header before passing the message to the application, and prepends it before sending to the host.

```

vtcpd -f | -F {len | d:D | l:{#|lf[:hp[:hl]]} |
          L:{w|lf[:hp[:hl]]} |
          a:{n|lf[:hp[:hl]]} |
          A:{n|lf[:hp[:hl]]}
      [ {-mB | -m h | -m t} ]

```

a:lf[:hp[:hl]]	Specifies that the (ASCII) length field is embedded within the message header. The length field (lf) contains the length of the body portion of the message. The header position (hp) is counted starting at 1 (which is the default). The header length (hl) defaults to lf+hp-1 . By default, VTCPD strips the header before passing the message to the application, and prepends it before sending to the host.
A:Lf[:hp[:hl]]	Specifies that the (ASCII) length field is embedded within the message header. The length field (Lf) contains the total message length (which is the header plus body portion). The header position (hp) is counted starting at 1 (which is the default). By default, VTCPD strips the header before passing the message to the application, and prepends it before sending to the host.
A:n	Specifies the length (in ASCII format) of the message header, which contains the data length in ASCII format.
a:n	Specifies that the length in the header field which includes the length of the data in addition to the length of the header.
-mB	Stipulates binary headers in non-network byte order.
-m h	Specifies that VTCPD is to retain the header in the message.
-m t	Specifies that VTCPD is to truncate all trailing spaces from the string prior to sending it to the host. This option should not be used with fixed-length messages.

```

vtcpd    -f | -F {len | d:D | l:{#|lf[:hp[:hl]]} |
              L:{w|lf[:hp[:hl]]} |
              a:{n|lf[:hp[:hl]]} |
              A:{n|lf[:hp[:hl]]}
            [ {-mB | -m h | -m t} ]
  
```

Examples: **vtcpd -f 120**

Indicates that the message has a fixed length of 120.

vtcpd -f d:0x0A

Indicates that the messages are delimited by newline characters (0x0A).

vtcpd -f l:2 -m t

Indicates that the message length is contained in a two-byte header. Any trailing spaces in the message are removed before it is sent to the host.

vtcpd -f A:3

Indicates that the message length is contained in a three-character header. For example, if the message sent by the host is "ABCDE", the message as received by an application would be "008ABCDE".

vtcpd -f a:3

Indicates that the combined message and data length is contained in a three-character header. For example, if the message sent by the host is "ABCDE", the message as received by an application would be "005ABCDE".

vtcpd -f a:4:1:4 -or- vtcpd -f l:4

Indicates that the message header contains only the length field (4 bytes in ASCII format) and the header is stripped when passed to the application.

vtcpd -f l:2:1:8 -m h

Indicates that the message header contains 8 bytes, the binary length field of 2 bytes is located at the beginning of the header, and the header is passed to the application.

-f Option	Action	strip header	retain header (-m h)
{l a}:lf	send to host	18 bytes	18 bytes
	send to application	10 bytes	18 bytes
	value of lf	10	10
{L A}:lf	send to host	18 bytes	18 bytes
	send to application	10 bytes	18 bytes
	value of lf	18	18

Message Identification (ID)

Messages are identified by a unique combination of consecutive bytes called the *message ID*. Unless a host connection is exclusively dedicated for a single phone line (see below), all messages must include an ID field. A request from an application and the corresponding reply from the host will have the matching message ID fields.

Identification fields for requests can either be assigned by applications (specified by the `-i` or `-j` option) or by VTCPD (specified by the `-I` option), but not both. If the daemon assigns the ID, it uses one of the following methods:

Bytes	Message ID Contents	Digits
one	Phone line of the application sending the request.	binary
two	Phone line and MPS number of the requesting application.	binary
three	Phone line number of the requesting application in the format 999.	ASCII
six	Phone line and MPS number of the requesting application.	ASCII



If it is specified that the ID is to be assigned by applications, they are responsible for ensuring that no two outstanding requests have the same ID. The daemon doesn't check the validity of the ID field. It only matches it byte for byte with the ID in the host responses. If two requests with the same ID are pending, applications may be given unrelated responses to their requests.

If host data contains an ID that cannot be associated with any phone line, it is normally discarded. However, it is possible to pre-assign an administrative phone line that receives all unidentified messages. The messages are sent to the line as the data portion of the condition `unexdata`.

vtcpd **{-i|I} {-j} {<len>:<pos>|<len>@<str>}
 [{-a <#>|-A <#> [-m i]]}**

Args:	-i	The message ID is assigned by applications. This option is used by default when the -j option is omitted, or for outgoing messages when the formats differ.
	-I	The message ID is assigned by VTCPD.
	-j	Same as -i , but for incoming messages only (when formats differ).
	<len>	Specifies the length in bytes for the message ID field.
	<pos>	Specifies the starting position of the ID field (i.e., the offset).
	<str>	Specifies a string within the message that identifies the ID field by directly preceding it. The string can be any set of ASCII characters except 0x00. Unprintable characters may be represented in hexadecimal as 0xXX.
	-a <#>	Specifies the line number of an administrative application that receives any unidentifiable messages.
	-A <#>	Specifies the line number of an administrative application that receives all messages from the host. In this case, the ID fields from application GET and SEND requests are ignored.
	-m i	Specifies that VTCPD is to run in <i>inverted</i> mode. This is intended for messages that cannot be correlated by means of the message ID field. Use this option if the message IDs are set by the host.

Examples: **vtcpd -i 48:1**

The message ID is supplied by the application. The ID field is 48 bytes in length and starts from the beginning of the message.

vtcpd -I 2:1

The message ID is supplied by VTCPD. The ID consists of binary representations of the phone line and MPS numbers.

vtcpd -I 6:24

The message ID is supplied by VTCPD. The ID consists of ASCII representations of the phone line and MPS numbers.

vtcpd -i 4@abc0x0A0x0B

The message ID is supplied by the application. The ID field is four bytes in length, and is located directly after the string "abcXY", where X and Y are binary values representing 1 and 2.

Connection Capacity

The term *connection capacity* refers to the number of outstanding host requests that can be associated with a given connection. The following choices are supported:

Capacity	Description
Unlimited	There is no limit to the number of outstanding host requests that can be associated with each link.
Limited	Each connection has a pool of slots, which are allocated to phone lines by VTCPD.
Common Pool	All host links have a common pool of slots, which means that the total number of outstanding host requests is limited.

The capacity parameter is specified as follows. Only one of these options is allowed. If neither of them is specified or if the value is greater than or equal to 254, the connections have unlimited capacity (the default). If the VTCPD daemon is to serve phone lines with numbers greater than 254, its `-p / -P` option must be increased to exceed the maximum line number.

```
vtcpd [ {-p <#> | -P <- | #>} ] [-m R]
```

Args:	<code>-p</code>	Lowercase "p" specifies limited capacity, where each connection has a pool of slots.
	<code>-P</code>	Uppercase "P" specifies a common pool of slots for all connections. The default is unlimited, indicated by a dash (-).
	<code>-m R</code>	Specifies that VTCPD chooses the connection and the corresponding slot in the pool using round robin load balancing (instead of the default method of round robin balancing among slots with the minimum number of outstanding requests.) If all slots are used the request is rejected.

```
Examples: vtcpd -l 20:mpshost:1000 -P 20
```

Specifies a common pool of 20. It is possible that all 20 requests will be sent through one connection, leaving all other links idle.

```
vtcpd -l 20:mpshost:1000 -p 1
```

Specifies one request per connection. Messages may be sent without IDs.

Application-Host Interaction Configuration Options

The following discussion is an overview of the essential aspects of programming an application for VTCPD communications from a configuration perspective. Typically, PeriProducer applications are programmed to perform the following steps in the indicated order using the functions of the Resource block:

- acquire the VTCPD resource (GET),
- send a message to the host (SEND),
- receive a reply from the host (RECEIVE),
- free the VTCPD resource (FREE).

This entails having VTCPD reserve an available host connection (the slot), send data to the host (associating the ID field with the phone line), receive data from the host based on the specified ID, and then release the slot.

After a successful attempt to acquire a VTCPD slot, the application receives the condition **gotres**. Data associated with the condition contains the connection number, slot number, and VTCPD port number in the format **XXX:YYY:ZZZ**. The application does not need to process this data.

If a reply from the host arrives before the application issues the RECEIVE command, the reply is held by the VTCPD daemon. If the application terminates, the slot is released automatically.

Regardless of the message format, the SEND request should include only the data portion of the message, because the delimiter (**-f | -F d**) or length header (**-f | -F 1**) is provided by VTCPD.

It is possible after the initial SEND request to mix SEND and RECEIVE requests in any order. The message ID of the last SEND is always in effect (until the slot is freed) and can be used to identify consecutive RECEIVES.

For example, in the following scenario, the application receives two messages with the specified ID-field:

- acquire the VTCPD resource,
- send a message to the host,
- receive a reply from the host,
- receive a second reply from the host,
- free the VTCPD resource.

During the interval between acquiring the resource (GET) and when the resource is released (FREE or STOP), the slot is reserved exclusively for that application line.

If the size of the slot pool is limited, it may be necessary to reserve the slot for a shorter period of time. If an application does not use GET (i.e., only executing SEND and/or RECEIVE requests), the slot is released as soon as the host data arrives.

If it is desired to not allocate a permanent slot with the GET request, the application can issue the FREE request (thus guaranteeing timely slot release) and still have dynamic slot allocation for each SEND and/or RECEIVE. To do this, use the configuration option **-m I** discussed below.

vtcpd [-m [d]] [{a|n}] [{1|L}] [{I|R}]

Args:	-m d	Specifies that a reply from the host may come from any connection. (Normally, a reply arrives from the host through the connection used to send the corresponding request.) The use of this option can increase the time used for internal message processing by VTCPD.
	-m a	Specifies asynchronous data delivery. If this option is included, and VTCPD receives data from the host when a RECEIVE request is not pending, the application receives the condition unexdata along with the host data.
	-m n	Specifies asynchronous data delivery. If this option is included, and VTCPD receives data from the host when a RECEIVE request is not pending, the application receives the condition unexdata , but does not automatically receive the host data. VTCPD will hold the data for the application and deliver it upon the next RECEIVE request.
	-m 1	Specifies that application replies to host-issued commands are to be sent via the same link from which the commands were received. This option is intended for environments where there is one host, and host commands drive the application. This cannot be used in environments where other host requests may arrive before the application sends its reply to a host command. (See -m L below.)
	-m L	Specifies that applications determine the particular links used for each host reply. This is intended for environments where multiple hosts might send requests while an application is still processing a prior request. This option causes VTCPD to prepend each message routed to an application with 12 ASCII bytes in the format xxx:yyy:zzz , which respectively denote the connection number, slot number, and VTCPD port number. The application should also prepend this header to all messages sent to VTCPD to guarantee that their delivery is via link xxx .
	-m I	Specifies that when an application issues a GET request, the slot is not reserved. Slots are allocated dynamically in this mode.
	-m R	Specifies that when an application issues a GET request, the slot is reserved until it is freed. Slots are allocated on a round robin basis.

Queuing Requests

When VTCPD receives a request from an application, it can either transmit it immediately to the host (the default mode), or keep it locally until any previous requests are answered.

In the former case, all requests are physically stored either on the host side or in the kernel buffer (from which the host process reads only one message at a time from the socket). If an application restarts before receiving the reply, this has no effect on the host, which eventually retrieves and processes the request (which may possibly be outdated at that time). The processing of outdated requests can further reduce limited host resources. The reply for such a request is discarded by VTCPD, if the application changes its ID each time it restarts.

In the latter scenario, if VTCPD holds requests locally, it can remove outdated requests from the queue when applications timeout and restart. Note, that this is useful only if the host is unable to process several messages simultaneously. The command line option `-q <#>` specifies how many unanswered requests should be sent to the host before queuing them locally. If the option is not specified (default), all requests are immediately written to the host. When `-q` is specified, VTCPD determines a proper link to the host and either sends the request or stores it in a queue, specific to the link. When a reply comes from the host, VTCPD transmits the first message from the queue to the host.

Another method is to maintain one global queue for all links. This is specified with the `-Q <#>` option. This option causes all requests to go into a queue where they are retrieved when the number of unanswered requests for any link becomes smaller than the specified `-Q <#>` argument. The `-m I` option is implicitly added to the list of command line arguments due to the fact that in this mode any request can be sent through any link. (The option `-m R`, if specified, is disabled.)

vtcpd [{`-q <#>` | `-Q <#>`}]

Args:	<code>-q <#></code>	Specifies how many unanswered requests go to the host before requests are queued locally. Unless this option is specified, all requests are immediately written to the host.
	<code>-Q <#></code>	Specifies that all requests from all links are to be placed in a global queue. When the number of unanswered requests for any link becomes smaller than the specified value, the requests are retrieved.



If an application times out and restarts, executing a FREE request causes the removal of all queued requests. Also, all line-related messages are removed from all queues when the application issues a GET request.

By default, VTCPD assumes that every request is answered by the host. It queues the requests and does not send a given request until the previous one has been answered. However, this linear relationship of send and receive requests might become unsynchronized for a variety of reasons (including improperly formatted requests, program malfunctions, etc.). Once the requests are out of synchronization, the associated connection effectively goes out of service (i.e., it is in an up state, but is unusable).

vtcpd **{-d <#> | -D <#>}**

Args:	-d <#>	Connection is recycled and VTCPD can use it again after the specified number of seconds.
	-D <#>	Connection is closed after the specified number of seconds. It may be re-opened by the system at a later point.



These two options are intended to work in conjunction with `-Q` or `-q`.

Monitoring Host Connections

If the event of a host process failure, the kernel on the host machine notifies VTCPD, which in turn sends a **hostdown** condition to applications. However, VTCPD is not notified if the entire host machine crashes or if the LAN connection is broken. To detect this situation, VTCPD should query the host periodically with a **ping** command. If a reply is not received in a specified amount of time, VTCPD then determines that the host is down.

vtcpd **-T <timeout>[:<ping-len>] -h len:<str>**

Args:	-T <timeout>[:<ping-sec>]	Specifies that a ping message is to be sent to the host every ping-seconds . (The default is 1.) If a reply is not received within timeout number of seconds, the host is deemed to be down.
-------	--	--

-h len:<str>	Specifies the format of the ping message. For message formats using delimiters or length headers (see <i>Message Format on page 227</i>) the length of the ping message may be equal to zero (in which case, only the header or delimiter is sent to the host. The str option is ignored). For fixed-length format, the length of the ping message cannot be zero, but may be smaller than the length of the message specified by the -f option. If str is not specified, the ping message is filled with ASCII '0' (e.g., -h4 , -h4:0 and -h4:0000 are equivalent). To specify an unprintable character, the hexadecimal representation (0xXX) should be used.
---------------------------	---

Examples: **vtcpd -T 30:10 -h 4:0x00**

Generates a ping message every 10 seconds with a total timeout of 30 seconds. The message consists of four binary zeroes.

vtcpd -T 20:5 -h 4:A0x410x42B

Generates a ping message every 5 seconds with a total timeout of 20 seconds. The message is the string "**AABB**".

Backup LAN

To help to ensure continued host connectivity, a backup LAN connection may be established between the node(s) and host machines. VTCPD tries to switch to the backup LAN if all host connections through the primary LAN are lost.

To indicate the existence of a backup LAN, use the following option. To specify multiple hosts and port numbers, the option should be used once for each host.

vtcpd **-L [#:]host:port**

#	Indicates the number of connections to the specified backup host. (The default is 1.)
host	Indicates the name of each host as it is known on the secondary network.
port	Indicates the TCP/IP port number to use with the backup host. (Port numbers 7000-7256 are reserved for the VMST and must not be used.)

In the event of primary host failure, if no connection is established through the backup LAN, VTCPD switches to the main LAN or proceeds with redundancy procedures.

VFTPD



The *VFTPD* (*VPS File Transfer Protocol Daemon*) resource is a background process that provides file transfer capability to application programs. This program is intended for high-volume file transfers, and runs as a memory-resident process. VFTPD provides for application programs the same functionality as does the standard FTP Solaris utility. Applications can access this external resource by using the SEND RESOURCE construct.

The number of running FTP processes determines the number of transfers that can be performed concurrently. These processes can be used for any host.

The command syntax and arguments are as follows:

```
vftpd  -v <#> [-N <name>] [-n <#>] [-q <#>] [-k <#>]
        [-R <#>] [-a]
```

Args:	Option	Description
	-v <#>	MPS number associated with an instance of vftpd.
	-N <name>	VMST host machine name. The default is the local node.
	-n <#>	Number of FTP processes for on-demand connections. The default is 1.
	-q <#>	Request queue size. Set to 0 for unlimited size. The default is 200.
	-k <#>	Keep-alive timer for permanent connections. The units for this timer are minutes. The default is 1.
	-R <#>	Retry timer for opening permanent connections. The units for this timer are seconds. The default is 30, the minimum is 0, and the maximum is 500.
	-a	Enables automatic retry of failed request due to connection loss. The maximum number of attempts is 3.

Specifying a Port

To specify the port number for VFTPD, make an entry in the \$ASEHOME/etc/services file leaving the Protocol field blank. For example:

#Service	Port	Protocol
vftpd	151	

See [The /etc/services File](#) on page 129 for more information.

Automatic Startup

To have VFTPD automatically start up with other VOS processes, make an entry for the daemon in the `$MPSHOME/common/etc/gen.cfg` file using appropriate command line arguments (see “vftpd” on page 240). For example:

#NAME	PRI	COMMAND LINE
vftpd	0	"vftpd -v1 -n2"

This entry starts VFTPD on MPS number 1 and allocates 2 FTP processes for on-demand connections.

Automatic FTP Logins

The `.netrc` file contains the data necessary for logging into remote hosts across the network for performing FTP file transfers. In order to automate FTP logins, the `.netrc` file must be modified to contain an entry for each configured host that supports FTP connections.

Each entry requires the following fields. An example follows.

Field	Description
machine	Identifies a remote node. The auto-login process searches the <code>.netrc</code> file for a node name that matches that of the remote machine specified on the FTP command line or as a command line argument. Once a match is made, the subsequent <code>.netrc</code> tokens are processed. This process stops when the end of the file is reached or another machine name is encountered.
login	Identifies a user on the specified node. The auto-login process initiates a login using the specified name.
password	Supplies the password for the specified user name. The auto-login process supplies the password as part of the login process.

Sample `.netrc` file

```
machine raven login peri password peri;
machine robin login peri password peri;
machine hawk login root password root;
machine eagle login root password root;
```

The `.netrc` file resides in the user's home directory on the node originating the file transfer requests. For example, if VFTPD is run under the user name `peri`, then the `.netrc` file should exist in the `/home/peri` directory. If the file does not exist, it must be created.

For additional information about this file, refer to the Solaris **netrc** man page.

Identifying the Configured Host Computers

To save time, permanent FTP connections should be made to all configured hosts used for high-volume file transfers. This is done in the `vftpd.cfg` file, which should be located in the `$MPSHOME/mpsN/etc` (`%MPSHOME%\mpsN\etc`) directory. If it is not there, the VFTPD daemon looks for it in the current directory. If the file does not exist, one should be created in accordance with the format below.

In the `vftpd.cfg` file, create entries for the host names, one per line, using the format `hostname, number_of_connections`. (Host names must not be entered more than once.) The following is an example:

Sample `vftpd.cfg` file

```
# vftpd.cfg
# Entries are of the form:
# <host>, <number_of_connections>
#
raven, 4
robin, 3
hawk, 1
eagle, 4
```

The **vftpd** daemon starts the specified child processes for every host included in the `vftpd.cfg` file as well as those FTP processes included with the **vftpd -n** command line option (see “-n <#>” on page 240).

No more than 25 FTP child processes can be started and these processes are first allocated for the configured connections. Since this is the maximum number of available processes, if 25 permanent connections are established in the configuration file, the system is unable to provide on-demand connections (despite the **-n** option). The fewer permanent connections created in the configuration file, the greater the number of available child processes for on-demand connections.



4

Configuration Procedures and Considerations

This chapter covers:

1. Making Changes to an Existing System
2. Verifying/Modifying Boot ROM Settings
3. N+1 Redundancy
4. Pool Manager (PMGR)
5. Database Conversion
6. Call Monitoring

Making Changes to an Existing System

The following sections offer basic examples of how to make changes to an existing Avaya Media Processing Server (MPS) Series system. A thorough understanding of the configuration issues discussed in earlier sections of this manual is necessary before attempting any of these procedures.

To upgrade your MPS network by adding a new node, you must configure existing PeriView Workstations to enable control from PeriView (see [Introducing a New Node](#) on page 248).

To enable statistics collection for the new Avaya MPS, see [Enabling Statistics Collection](#) on page 249.

Adding Spans

In this example, a new TMS (eight new spans) has been added to an existing chassis. Make the following modifications to the `tms.cfg` file:

1. Under [Line Card Protocol Package Definitions](#) on page 114, add one new LOAD statement for each new span.
2. In the [MPS Line Definition Section](#) on page 116, add one new LINE statement for each new span.
3. In the [Synclist Configuration Section](#) on page 119, Add new REF_SRC statements for the new spans, or incorporate the new spans into the existing statements. The new spans should be specified under entries for both REFCLK_A and REFCLK_B.
4. Stop and restart SRP (see [Manually Starting and Stopping SRP](#) on page 70).

Modifying the Span Resource Set

This example is indicative of recent statistics and/or traffic conditions indicating a need for additional outbound lines (e.g., one span). To accommodate this, make the following modifications to the `tms.cfg` file:

1. Under [Line Card Protocol Package Definitions](#) on page 114, place an asterisk (*) in the `Outline` field of the LOAD statement for the span being changed.
2. In the [Resource Set Table Section](#) on page 118, add an RSET statement using the `RSET_PROFILE` name defined in the [System Description Section](#) on page 106.
3. Stop and restart SRP (see [Manually Starting and Stopping SRP](#) on page 70).

Changing Pool/Class Names

To change a resource pool/class name (e.g., make it more descriptive or specific), effect the following modifications to the `tms.cfg` file:

1. In the *TMS Resource Definition Section* on page 108, change the `CLASS_NAME` to the new value.
2. In the *System Description Section* on page 106, change the class name where it appears in all `RSET_PROFILE` statements.
3. Stop and restart SRP (see *Manually Starting and Stopping SRP* on page 70).

Renumbering a Component



The following procedure does not apply to nodes designated strictly as PeriView workstations, as these nodes do not contain any components.

Avaya typically ships systems using the default component number 1. If you are on a network with multiple nodes, you must perform the following steps on that node where you desire to renumber the component (the node that component is associated with):

1. Take the **MPS** system down or, at a minimum, kill all SRP processes. To take the system down, see *Solaris Startup/Shutdown* on page 67 or *Windows Startup/Shutdown* on page 69. To kill SRP on the MPS, see *Manually Starting and Stopping SRP* on page 70. See the *Avaya Media Processing Server Series System Operator's Guide* for additional information on performing this step.
2. Use the syntax `mvcmp 1 <destination#> <component_type>` where `<destination#>` is the new number of the component and `<component_type>` is the noun name of the component (i.e. `mps`, `oscar` etc.). The noun name must be entered in lower case letters. Enter this information in a command tool or at the command prompt of the node where the component in question resides.
3. Restart the SRP processes (or restart the MPS system). To restart the MPS system, see *System Startup* on page 65. To restart SRP on the MPS, see *Manually Starting and Stopping SRP* on page 70. See the *Avaya Media Processing Server Series System Operator's Guide* for additional information on performing this step.

Renaming a Solaris MPS Node

Avaya typically ships systems with default node names already established. To rename a Solaris node:

1. Log in as super user (`root`).
2. Modify the node name in the files `/etc/hosts`, `/etc/nodename`, `/etc/ethers`, and `/etc/hostname.le1`. Contact your System Administrator or Certified Avaya support personnel for details.



The file `/etc/hostname` shown in the [example](#) may contain an extension other than `le1`, depending on the type of network card installed on the system. To identify your file, replace `le1` with the name required by the network card driver.

3. Update the name of the node in the `vpshosts` files. This file identifies the MPS network components, and resides on each node in the network. The same information should be contained across all nodes. For information on performing this action, please see [step 2](#). below.

Renaming a Windows MPS Node

Avaya typically ships systems with default node names already established. To rename a Windows node:

1. Modify the node name in the Workgroup entry (you must be logged in with administrative privileges to perform this action). To check on the nodes listed in the Workgroup, follow the menu path **Start—Settings—Control Panel**. Double <LEFT> click on the **Network** icon, and select the **Identification** tab on the pop up window (this tab appears initially by default). <LEFT> click on the **Change** button, enter the new name in the **Computer Name** window, and stipulate whether the node is a member of a Workgroup or domain (MPS systems are shipped as a Workgroup) and the name of such. <LEFT> click on the **OK** button to have the change take effect. The old name is replaced by the new name in the Workgroup.



There are considerations to be made depending on how a node appears on a network. If the node is maintained as part of a domain, the new name must be registered in the domain. If the hostname is in any `hosts` or `lmhosts` or `WINS` databases, they must also be updated. Contact your System Administrator or Certified Avaya support personnel for details if your hostname is or was once defined in a domain maintained by them.

2. Update the name of the node in the `vpshosts` files as well. This file identifies the MPS network components, and resides on each node in the network. The same information should be contained across all nodes. For information on performing this action, please see [step 2. on page 248](#).

Introducing a New Node

1. Perform either of the following, as applicable:
 - On the existing Solaris node(s):
 - (1) Log in as super user (`root`).
 - (2) To identify the new node to existing nodes in your network, modify the `/etc/hosts` file on each node. Each node uses its own `/etc/hosts` file to recognize nodes in the network. Add the new node information.
 - On the existing Windows node(s), add the new node to the Avaya Workgroup or physically add it to the domain (see [step 1. on page 247](#)) for the Windows domain server to recognize (you must be logged in with administrative privileges to perform these actions). Contact your System Administrator or Certified Avaya support personnel for details.
2. Use **vhman** (text-based editor) or **xvhman** (GUI-based editor) to modify the `MPSHOME/common/etc/vpshosts` file. This file identifies the MPS network components, and resides on each node. The same information should be contained across all nodes.
3. Issue the **vhman** command from each existing node in the network. Use the syntax **vhman -c<#> -h<hostname> -t<component> -a**, where **<#>** is the new component number, **<hostname>** is the new node name, and **<component>** is the new component type (i.e. MPS, oscar, etc.). Component type should be entered in lower case letters, and all entries do *NOT* have a space between the option and the variable (for example, **vhman -c113 -hnewnode -tmps -a**).
4. Issue the **vhman** command from the new node. Use the syntax [shown above](#), where **<#>** is an existing component number, **<hostname>** is an existing node name, and **<component>** is an existing component type. You must use a separate **vhman** command to add each existing component/node to the `vpshosts` file on the new node.



If **xvhman** is used, multiple steps can be accomplished from the single tool.

5. Comment out any references to VSUPD in the `MPSHOME/common/etc/gen.cfg` file *only* if you do not wish to collect any application statistics from the MPS (or any other component on the node); otherwise, uncomment this line. (See [The gen.cfg File on page 96.](#))

Enabling Statistics Collection

To enable system or application statistics collection for an MPS node:

1. For *new* nodes, be sure that the MPS has been added to the `vpshosts` files in accordance with the preceding instructions (see [Introducing a New Node on page 248](#)). Existing nodes should already appear in the `vpshosts` files in the network.
2. For application statistics collection purposes, you *must uncomment* any references to `VSUPD` in the `$MPSHOME/common/etc/gen.cfg` file. (See [The gen.cfg File on page 96](#).)
3. For system statistics collection purposes, you *must uncomment* any references to `VSTAT` in the `$MPSHOME/mpsN/etc/vos.cfg` file. (See [The vos.cfg File on page 143](#).)

The new node or nodes must also be specified as a collection point in the report(s) defined in PeriReporter Tools (if desired). For details on how to configure report parameters, see the *PeriReporter User's Guide*. For procedural steps on generating and printing reports, see the *Avaya Media Processing Server Series System Operator's Guide*.

Debug Terminal Connection

For verifying the TMS boot ROM and certain diagnostic operations, it is necessary to establish a connection to the serial port on the front of the TMS or DCC. (See illustration of *TMS Front Panel* on page 252.) This is an asynchronous RS232 interface via an 8-pin DIN connector. The interface is configured with the following parameters:

- baud rate = 9600 bps
- character size = 8
- stop bits = 1
- parity = 1

Connection can be made with a dumb terminal or PC/laptop using the proper cables. It may also be possible to establish a session from the system console.

Connection Using a Dumb Terminal or PC

A dumb terminal is connected to the TMS or DCC port using an 8-pin DIN to serial port adapter cable (Avaya part no. 9901347). You can also use a DB25 to 8-pin DIN cable (Avaya part no. 9900746) and a null modem adapter (Avaya part no. A0874210).

A PC/laptop can be connected using the same cables, by connecting the DB25 connector to one of the PC's serial port connectors. Some PCs may have a DB9 (9-pin) serial connector. If so, a DB9 to DB25 adapter/cable (commercially available) must also be used. The COM assignment of the port being used must be determined and some configuration may be necessary. The connection is established using the HyperTerminal utility included with all versions of Microsoft Windows. If HyperTerminal is not installed, it will need to be installed from the Windows CD, using the Windows setup tab in the Add/Remove Programs dialog box in the Control Panel. HyperTerminal is included under the Communications utilities.

Connection from the System Console

If a terminal or PC/laptop is not available, it may be possible to use the system console or workstation to establish a telnet connection to the desired port using the Solaris **tip** command.



1. Login to the system as `root`.
2. Determine that the system is configured as required. If the system is not configured as follows, the system console cannot be used to open a **tip** session.
 - a. Change directory (`cd`) to `/dev/term` and check (`ls -al`) that there are symbolic links for device ports a and b:

```
peri@scn1 {2} ls -al
total 14
 2 drwxrwxr-x  2 root   root       512 Jun 11 23:52 ./
 8 drwxrwxr-x 17 root   sys        3584 Sep 17 11:53 ../
 2 lrwxrwxrwx  1 root   root        47 Jun 11 23:52 a ->
../..../devices/pci@1f,4000/ebus@1/se@14,400000:a
 2 lrwxrwxrwx  1 root   root        47 Jun 11 23:52 b ->
../..../devices/pci@1f,4000/ebus@1/se@14,400000:b
```

- b.** Change directory (**cd**) to `/etc` and view (**more** or **cat**) the file `remote`. Check for the following statements in the file:

```
hardwire:\
:dv=/dev/term/b:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
hardwirea:\
:dv=/dev/term/a:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
```

- 3.** Connect the A/B connector of a “Y” adapter (Avaya part no. A0734660) to the serial port on the application processor.
- 4.** Connect the DB25 to 8-pin DIN cable (Avaya part no. 9900746) from one of the ports (A or B) on the “Y” adapter to the system port to be checked.
- 5.** Open permissions on the symbolic links:
chmod 777 /dev/term/*
- 6.** Enter the command **tip hardwire** (default, for port B) or **tip hardwirea** (for port A).



When the `tip` session is closed, permissions on the symbolic links are returned to their original state.

Verifying/Modifying Boot ROM Settings

The TMS, DCC, and NICs contain field programmable gate arrays (FPGA) that must be loaded with an *image* that provides the functionality for the board. These images are loaded whenever SRP starts. The boot ROM defines parameters such as which image file to load, the clock frequency of the board, IP addresses, and other local parameters. When the system is installed, the default factory settings must be verified and/or modified to ensure the correct image files are loaded and that the local IP addresses for the TMS and server (where the image definition files (*.idf) reside) are correct.

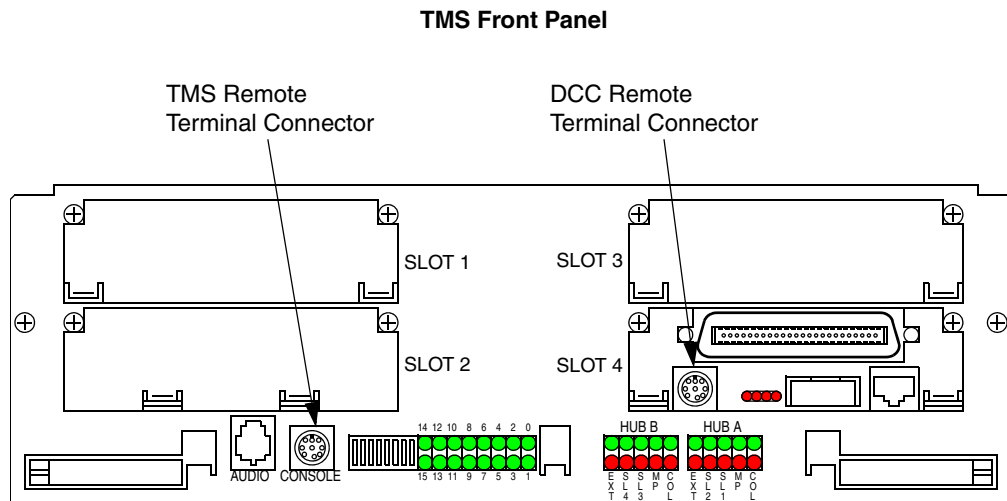
Whenever a new system is installed or has assemblies added or replaced, the boot ROM should be verified and, if necessary, modified. When replacing or adding assemblies, only the boot ROM on the associated TMS(s) or NIC(s) will need to be checked.

For a TMS, first verify or modify the boot ROM settings for the DCC. (See [DCC Boot ROM on page 252](#).) Then, verify/modify the boot ROM for the TMS. (See [TMS Boot ROM on page 256](#).) Follow this order for each TMS in the system. For the NICs, no particular order is needed.

Checking the boot ROM is an interactive process. If you elect to modify any of the current boot ROM settings, you will be prompted on each individual item to change or accept the current value (by hitting <RETURN>).

DCC Boot ROM

Perform the following procedure to verify/modify the boot ROM settings for each DCC in the TMS. Repeat for all DCCs in a TMS, then proceed to [TMS Boot ROM on page 256](#).



1. Connect a serial console to the remote terminal connector of the DCC. (See illustration of [TMS Front Panel](#) above and [Debug Terminal Connection on page 250](#).)

2. Cycle the power on the TMS. The ROM banner and the current settings will be displayed on the serial console. Press any key within 5 seconds to suspend the system from booting.

```
DCC-2000 Rom Version V2.0, Checksum = 0x0187f210
dcc_rom.elf, Release 1.0.0.14 [05/17/99 06:35:25 PM]
Copyright (C) 1999, Periphonics Corporation
-----
STARTUP MODE:
  Run Application
NETWORK INTERFACE PARAMETERS:
  TMS is present
  Use Message Exchange to download image
  IP address on LAN is 192.168.101.5
HARDWARE PARAMETERS:
  Serial channels will use a baud rate of 9600
  This board's Ethernet hardware address is 00:80:01:A0:00:96
  This board's clock frequency is 40 MHz
  This board has 16 Megabytes of DRAM
SOFTWARE WATCH DOG TIMER STATUS:
  The watch dog timer is DISABLED and will NOT timeout.
MXD BOOT LOADER PARAMETERS :
  The download file will come from the TMS
STARTUP DELAY:
  Will wait 5 seconds before start up to allow parameter modifications
-----
To change any of this, press any key within 5 seconds
```

3. Verify that the Watchdog timer is DISABLED.
4. If displayed values are all correct, select (C)ontinue option by typing
 - c. To modify any current settings:
 - a. Select (M)odify option by typing 'm':

```
(M)odify any of this or (C)ontinue? [M] m
```

- b. Press <RETURN> to accept current values of items to remain unchanged.

- c. To change an item's value, type the new value after the prompt. Continue until all values are done.

```
DCC-2000 Rom Version V2.0, Checksum = 0x0187f210
dcc_rom.elf, Release 1.0.0.14 [05/17/99 06:35:25 PM]
Copyright (C) 1999, Periphonics Corporation
-----
STARTUP MODE:
  Run Application
NETWORK INTERFACE PARAMETERS:
  TMS is present
  Use Message Exchange to download image
  IP address on LAN is 192.168.101.5
HARDWARE PARAMETERS:
  Serial channels will use a baud rate of 9600
  This board's Ethernet hardware address is 00:80:01:A0:00:96
  This board's clock frequency is 40 MHz
  This board has 16 Megabytes of DRAM
SOFTWARE WATCH DOG TIMER STATUS:
  The watch dog timer is DISABLED and will NOT timeout.
MXD BOOT LOADER PARAMETERS :
  The download file will come from the TMS
STARTUP DELAY:
  Will wait 5 seconds before start up to allow parameter modifications
-----
To change any of this, press any key within 5 seconds
```

- d. When all prompts are finished, the original verify screen is displayed again with the new values. To make additional changes, select the (M)odify option again. To save the changes, select (C)ontinue.

```
(M)odify any of this or (C)ontinue? [M] c
Writing the modified parameters to the removable EEPROM
Waiting for FPGA startup Flag
...
  <ROM banner and parameters displayed here again for verification.>
...
Waiting For MXD message
Ready to receive image on MXD
.....

dcc2000.elf, Release 1.0.0.67 [06/08/00 02:00:10 PM]
Copyright (C) 2000, Periphonics Corporation

Initializing SCC2 Hardware!!

Periphonics Net5 ISDN (User-Side)
```

- e. Return to step 1 for each DCC in the TMS, or proceed to ***TMS Boot ROM*** on page 256.

TMS Boot ROM

1. Connect a serial console to the **CONSOLE** port of the TMS. (See illustration of *TMS Front Panel* on page 252 and *Debug Terminal Connection* on page 250.)
2. Cycle power on the TMS. The ROM banner and current settings will be displayed on the serial console. Press any key within 5 seconds to suspend the system from booting.

```
TMS-2000 Rom Version V3.0, Checksum = 0x0434b441
tms_rom.elf, Release Developmental [02/14/00 02:00:10 PM]
Copyright (C) 2000, Avaya Inc
-----
STARTUP MODE:
  Run Application
NETWORK INTERFACE PARAMETERS:
  LAN IP address will be obtained from etc/bootptab
  LAN interface's subnet mask is 0xffff0000
HARDWARE PARAMETERS:
  Serial channels will use a baud rate of 9600
  This board's Ethernet hardware address is 00:80:01:80:01:9C
  This board's clock frequency is 80 MHz, bus ratio is 1:1
  This board has 32 Megabytes of DRAM
SOFTWARE WATCH DOG TIMER STATUS:
  The watch dog timer is ENABLED and will timeout / RESET in 120 seconds.
TFTP BOOT LOADER PARAMETERS
  The IP and download file to start will come from the /etc/bootptab file
STARTUP DELAY:
  Will wait 5 seconds before start up to allow parameter modifications
-----
To change any of this, press Enter, Return, or space key within 5 seconds

(M)odify any of this or (C)ontinue? [C]
```

3. Verify the items are set as follows in ROM:
 - a. IP address of TMS is obtained from /etc/bootptab.
 - b. Watchdog timer is ENABLED.

4. If displayed values are all correct, select (C)ontinue option. To modify any current settings:
 - a. Select the (M)odify option:

```
(M)odify any of this or (C)ontinue? [M] m
```

- b. Press <RETURN> to accept current values of items to remain unchanged.

```
For each of the following questions, you can press <Return> to select the value shown in braces, or you can enter a new value.
```

```
How should the board boot?
```

1. pROBE+ standalone mode
2. pROBE+ waiting for host debugger via serial connection
3. pROBE+ waiting for host debugger via a network connection
4. Run the Application

```
Which one do you want? [4]
```

```
NETWORK INTERFACE PARAMETERS:
```

```
Do you want a LAN interface? [Y]
```

```
BOOTP, RARP or FIXEDIP can obtain this Board's IP.
```

```
The default method is BOOTP
```

```
Enter Y to change the default [N] y
```

```
Enter 0 for RARP, 1 for BOOTP, 2 for FIXED [1] 1
```

```
New configuration is: BOOTP
```

```
Use a subnet mask for the LAN interface? [N]
```

```
Should there be a default gateway for packet routing? [N]
```

```
HARDWARE PARAMETERS:
```

```
Baud rate for serial channels [9600]
```

```
Do you want to change the board's clock rate? [N]
```

```
Do you want to change the SWT status? [N]
```

```
Do you want to change the board's Ethernet address? [N]
```

```
ROM BOOTLOADER PARAMETERS:
```

```
STARTUP DELAY:
```

```
How long (in seconds) to delay before starting up? [5]
```

- c. To change an item's value, type the new value after the prompt. Continue until all values are done.

- 5. When all prompts are finished, the original verify screen is displayed again with new values. To make additional changes, select the (M)odify option again. To save the changes and boot, select (C)ontinue. The TMS will then boot, and a printout such as the following will be displayed:

```
-----
(M)odify any of this or (C)ontinue? [C]
Writing the modified parameters to the removable EEPROM

Starting the TFTP download...
.....
.....
TFTP download completed...
Transferring control to the downloaded code.

tms_860.elf, Release 1.0.0.67 [06/08/00 12:27:13 PM]
Copyright (C) 2000, Avaya Inc

Chassis: 04Backplane Slot Number: 04

Available Processor Memory: 26MB

Connecting to Host 192.168.212.2 (port#5500).....ESTABLISHED

Detecting Cards:

Card Revision          SlotCardType
-----
5036484011_           0 0   TMS-2500
5036484011_           0 1   MDM/TMS-2500
503643701E_.503643601B_ 4 0   DCC-2000-T1
```

Loading FPGAs:

```
Loading TMS-2500 in slot 0, card 0 with tms484a.sm
.....
.....
.....
.....
.....
.....
```

Load Successful

No FPGA load required for MDM/TMS-2500 in slot 0, card 1

```
Loading DCC-2000-T1 in slot 4, card 0 with xldccpli.sm
.....
.....
.....
```

Load Successful

No FPGA load required for in slot 6, card 0

Initializing Device Drivers:

```
TDM Driver.....SUCCESSFUL
VMEM Driver.....SUCCESSFUL
PKG Driver.....SUCCESSFUL
ALI Driver.....SUCCESSFUL
TSS Driver.....SUCCESSFUL
MXD Driver.....SUCCESSFUL
PLL Driver.....SUCCESSFUL
```

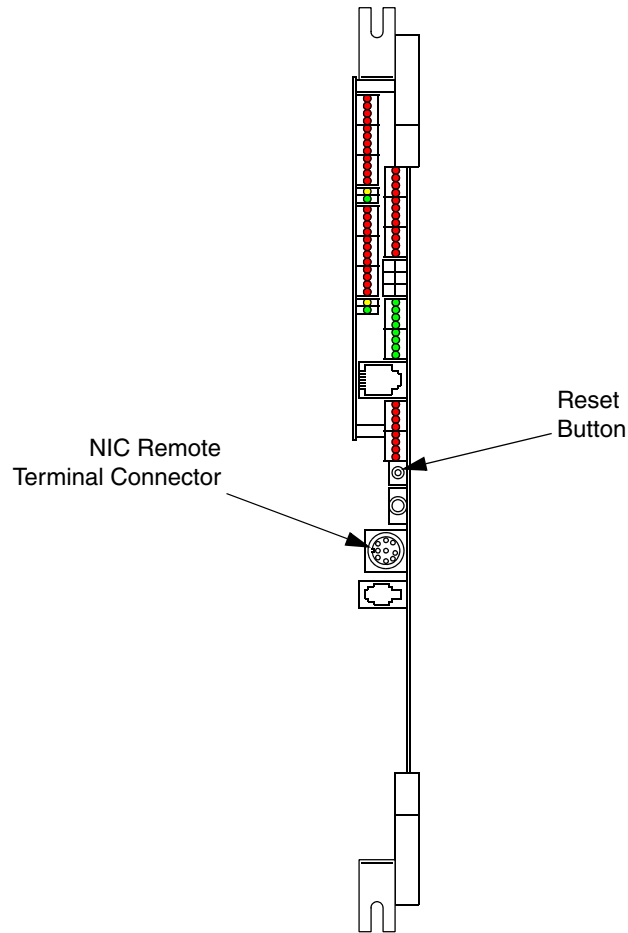
Driver Initialization Complete

6. When the TMS has booted, the Alarm Viewer indicates the system is up. If additional TMSs are installed in the system, return to [DCC Boot ROM on page 252](#) and [TMS Boot ROM on page 256](#), and repeat these procedures for each.

NIC Boot ROM

The boot ROM on the NICs should be verified and/or modified upon initial installation or when a NIC is replaced.

NIC Edge View



1. Connect a serial console to the remote terminal connector on the NIC. (See [Debug Terminal Connection](#) on page 250.)
2. Reset the NIC. (See [Resetting the NIC](#) on page 264.) The ROM banner and current settings will be displayed on the serial console. Press any key within 5 seconds to suspend the NIC from booting.

```

NIC-2000 Rom Version V3.0, Checksum = 0x0413497c
nic_rom.elf, Release 1.0.0.57 [04/11/00 02:05:49 PM]
Copyright (C) 2000, Avaya Inc
-----
---
STARTUP MODE:
  Run Application
NETWORK INTERFACE PARAMETERS:
  LAN IP address will be obtained from etc/bootptab file
  LAN interface's subnet mask is 0xffff0000
HARDWARE PARAMETERS:
  Serial channels will use a baud rate of 9600
  This board's Ethernet hardware address is 00:80:01:B0:00:25
  This board's clock frequency is 28 MHz, bus ratio is 1:1
  This board has 32 Megabytes of DRAM
SOFTWARE WATCH DOG TIMER STATUS:
  The watch dog timer is ENABLED and will timeout / RESET in 60 seconds.
TFTP BOOT LOADER PARAMETERS:
  The IP and download file to start will come from the /etc/bootptab file
STARTUP DELAY:
  Will wait 5seconds before start up to allow parameter modifications
-----
---
To change any of this, press Enter, Return, or space key within 3 seconds
(M)odify any of this or (C)ontinue? [C]

```

3. Verify the items are set as follows in ROM:
 - a. IP address of NIC will be obtained from /etc/bootptab file
 - b. Watchdog timer is ENABLED.
4. If displayed values are all correct, select (C)ontinue option by typing
 - c. To modify any current settings:
 - a. Select (M)odify option by typing 'm':

```
(M)odify any of this or (C)ontinue? [M] m
```

- b. Hit <RETURN> to accept current values of items to remain unchanged.

For each of the following questions, you can press <Return> to select the value shown in braces, or you can enter a new value.

How should the board boot?

- 1. pROBE+ standalone mode
- 2. pROBE+ waiting for host debugger via serial connection
- 3. pROBE+ waiting for host debugger via a network connection
- 4. Run the Application

Which one do you want? [4]

NETWORK INTERFACE PARAMETERS:

Do you want a LAN interface? [Y]

BOOTP, RARP or FIXEDIP can obtain this Board's IP.

The default method is BOOTP

Enter Y to change the default [N] y

Enter 0 for RARP, 1 for BOOTP, 2 for FIXED [1] 1

New configuration is: BOOTP

Use a subnet mask for the LAN interface? [N]

Should there be a default gateway for packet routing? [N]

HARDWARE PARAMETERS:

Baud rate for serial channels [9600]

Do you want to change the board's clock rate? [N]

Do you want to change the SWT status? [N]

Do you want to change the board's Ethernet address? [N]

ROM BOOTLOADER PARAMETERS:

STARTUP DELAY:

How long (in seconds) to delay before starting up? [3]

- c. To change an item's value, type the new value after the prompt. Repeat until all values are done.
- 5. When all prompts are finished, the original verify screen is displayed again with new values. To make additional changes, select (M) odify option again. To save the changes and boot, select (C) ontinue. The NIC will then boot, and a printout such as the following will be displayed:

```
(M)odify any of this or (C)ontinue? [C]
Writing the modified parameters to the removable EEprom

Starting the TFTP download...
.....
TFTP download completed...
Transferring control to the downloaded code.
```

nic_860.elf, Release [Developmental] [06/19/00 02:10:23 PM]
 Copyright (C) 2000, Avaya Inc

Chassis: 04Backplane Slot Number: 07Role: Hardware SLAVE

Available Processor Memory: 14MB

Connecting to Host 192.168.212.2 (port#5500).....ESTABLISHED

Detecting Local Cards:

Card Revision	CardType
503639901E_	0NIC-2000
503644701A_	1NIC-ETHMEZ

Detecting Other Cards in Chassis:

Card Revision	BPSType
503639401F_	0TMS-2000
5036484011_	3TMS-2500

Detecting Power Supplies in Chassis:

Card Revision	BPSType
203643213A_	0NIC-PS
203643213A_	2NIC-PS
203643213A_	3NIC-PS

Loading FPGAs:

Loading NIC-2000 with nic399b.sm

Load Successful

No FPGA Load required for NIC-ETHMEZ

Initializing Device Drivers:

HUB Controller.....SUCCESSFUL
 TSS Driver.....SUCCESSFUL
 BPS Manager.....SUCCESSFUL
 PLL Driver.....SUCCESSFUL
 ST16554 Driver.....SUCCESSFUL

Driver Initialization Complete

Resetting the NIC

The NIC can be reset by doing either a hard or soft reset. A hard reset momentarily removes power from the NIC, allowing it to reboot when power is reapplied. To do a hard reset on the NIC, press and release the reset button on the edge of the NIC. (See [NIC Edge View](#) on page 260.)

A soft reset of the NIC, is done at the command line in a V-shell. Move to the `tmscomm.N` component and enter

```
ncd nicbpsreset '<NicCh> <NICType> <bps> <toggle_hold>'
```

where:

NicCh = Chassis in which NIC is present (**1, 2, ... 12**)

NICType = Type of NIC - master **<m>** or slave **<s>**

bps = Slot to reset (**7** or **8**)

toggle_hold = (uint32): Set to **1** for toggle, **0** otherwise

(Note that arguments are enclosed in single quotes ('').)



To determine which NIC is currently set as the master/slave, use the `ncd systemstatus` command.

Example:

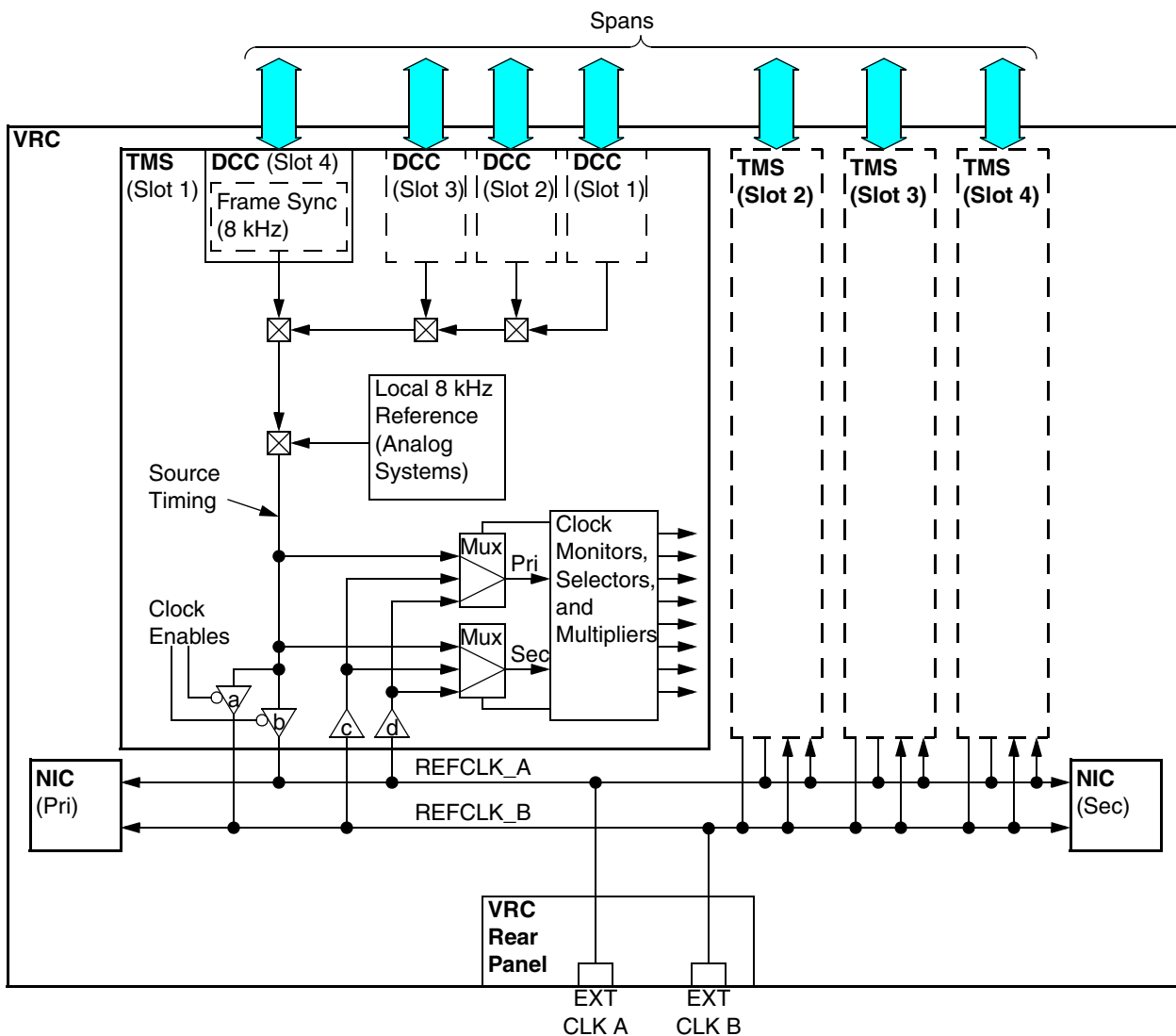
```
ncd nicbpsreset '2 m 7 0'
```

Resets the master NIC in chassis 2, slot 7. The value of **toggle_hold** is not important. The preceding help information can be displayed in a V-shell while in a `commN` component by entering `ncd help nicbpsreset`.

TMS Computer Telephony (CT) Bus Clocking

For the purpose of synchronizing CT communications across the PCM highways of each chassis and ATM switching fabric between chassis, each node is configured to use primary and secondary reference clocks from any one of several sources. (For configuration, see [Synclist Configuration Section on page 119](#).) The TDM backplane bus includes two reference clocks (REFCLK_A and REFCLK_B). Each TMS can drive or receive either or both of these clocks, using any single source on a particular TMS. However, the most reliable degree of redundancy is obtained when the reference sources reside in different spans, DCCs, TMSs, and chassis, if system scale permits.

Digital systems use the 8 kHz frame sync from a DCC span. Any DCC on the node can be configured as the source of either REFCLK_A or REFCLK_B. Analog systems use a local 8 kHz reference on a TMS. The local reference oscillator on any TMS can be used for either reference clock.



For example, in the illustration above, consider that the selected source for REFCLK_A is the frame sync from TMS1/ DCC4. Clock driver **b** is enabled to drive REFCLK_A and clock driver **a** is inhibited. Clock receivers **c** and **d** provide the system REFCLKs to the input of the primary and secondary multiplexers where these references are selected for the local TMS. The primary or secondary reference is used to generate all the clocks for the TMS CT bus timing.

Multiple sources can be specified for both REFCLK_A and REFCLK_B in the synclist section of the `tms.cfg` file. (See [Synclist Configuration Section on page 119](#).) Clock monitoring and selection provides several prioritized layers of redundancy so that even multiple failures are compensated for by switching to other clock sources.

The VRC rear panel provides two BNC connectors (EXT CLK A and EXT CLK B) for connection to external timing sources. These are typically provided by customer supplied equipment available on site. If an external timing source is used, the `[SYNC_LISTS]` section of the `tms.cfg` file is commented out and no local clocking is used. The TMS automatically detects the presence of a clocking source on either external input.

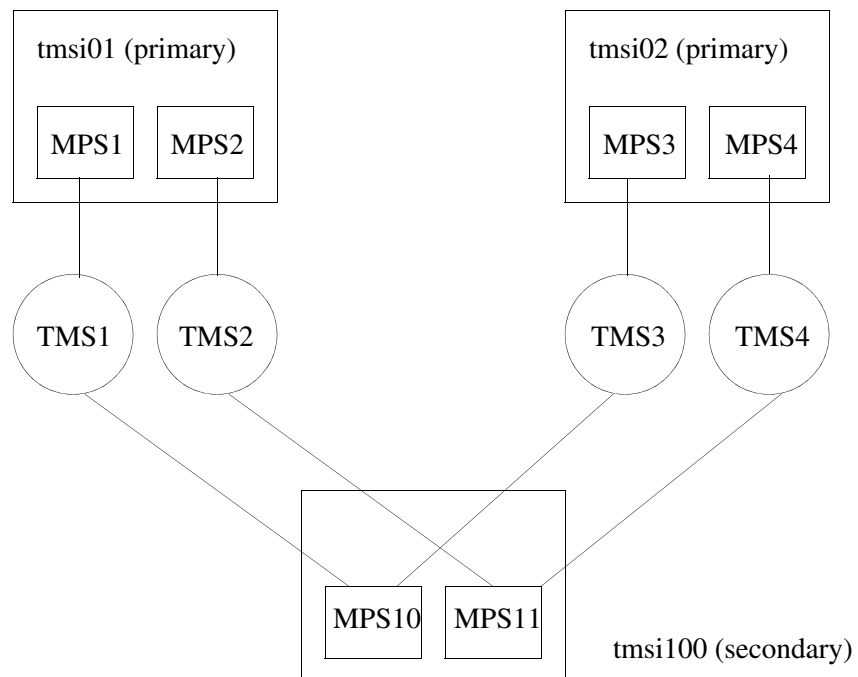
N+1 Redundancy

N+1 Redundancy is a system backup feature that protects systems from software and/or hardware failures by automatically switching to a second Application Processor. In an N+1 configuration, one node serves as a backup or secondary node for multiple operational or primary nodes. The group of primary components and their backup component(s) is referred to as a *cluster*. The figure below shows a basic MPS 1000 N+1 redundancy system configuration.



MPS 500 supports 1+1 redundancy only.

Sample MPS 1000 N+1 Redundancy System Configuration



N+1 redundancy also works in non-homogeneous configurations. The primary nodes can have different MPS and/or common component configurations than the secondary (backup) node.



N+1 redundancy is designed to protect against MPS/node function failures that result in the inability of the TMS hardware to communicate with its controlling MPS/node. N+1 cannot protect against TMS hardware failures.

During normal system operation, the backup node only monitors the primary nodes, referred to as “secondary/standby mode”. When an MPS component on one of the primary nodes fails, the secondary node assumes the configuration of the primary node, picks up all current call activity, and continues call processing within a short period of time, referred to as “secondary/active mode”. After the problems on the primary components are resolved and they are running normally again, the backup node can automatically return control back to the primary node (if auto-failback is

enabled), or a user can manually return control back to the primary using the TRIP **manual-failback** command. The backup node then resumes primary node monitoring.



Once the backup node enters active service, there is no longer an available backup to compensate for component failure of remaining primary nodes. Isolate and fix the problem on the primary nodes as soon as possible to re-establish the backup facility.



Resources (such as oscar, fax, etc.) might not be immediately available to the secondary node after failover. The resources first need to be freed before they become available to the secondary node.

TRIP Failback

The TRIP **auto-failback** command (used in `trip.cfg`) allows the primary node to automatically resume primary call processing once the primary node is up and running again. If, during startup, TRIP on a primary node detects that a TMS is controlled by a secondary node (and auto-failback is enabled), the primary node TRIP sends a failback request to the secondary node TRIP and the secondary node TRIP releases the TMS. The primary node then acquires the TMS and continues call-processing.

The TRIP **manual-failback** command provides the user the flexibility of determining when control should be given back to the primary node, and is only used when auto-failback is set to "off". This is accomplished via the following TRIP console/vsh command:

```
vsh-> trip manual-failback hard
```

There are two types of failback: hard and soft. During hard failback, the secondary node releases the TMS immediately and all calls in progress are disconnected immediately. During soft failback, the secondary node releases the TMS only after all calls and conferences in progress terminate normally. This prevents loss of any call service during failback, but causes failback to take an unpredictable amount of time to complete.



As of this writing, soft failback is not available. However, this can be accomplished by soft terminating applications on the secondary node prior to issuing the TRIP **manual-failback** command on the primary node.

The **trip restart-tms** command (used in `trip.cfg`) allows the primary TRIP process to avoid restarting the TMS (if the TMS is in the READY or STARTUP state) when the primary TRIP resumes control. This can speed up the total primary node restart time. However, not restarting the TMS forces the TMS to use the last loaded configuration. If the TMS requires a new configuration, it has to be manually restarted (usually by power cycling the system).

Refer to the *MPS Command Reference Manual* for details of these TRIP commands.



Depending on system configuration, the `tmscomm` component can run on a secondary node. However, it will not be restarted during failover and failback.

Directory Layout on a Secondary (Backup) Node

The secondary node contains standby configurations for local components in the directories `$MPSHOME/mpsN.standby` and `$MPSHOME/common.standby`. Standby configuration starts a minimal number of processes needed to maintain the system in the standby mode. MPS components contain only TRIP and TCAD processes.

The secondary node also stores copies of all remote common and MPS components in a cluster. MPS component configurations are stored in `$MPSHOME/mpsN` directories where `N` is the ID number of the primary component. Common component configurations are stored in `$MPSHOME/common.<nodename>` directories where `nodename` is the hostname of a primary node. Miscellaneous files from any particular node are stored in `$MPSHOME/miscfiles.<nodename>` directories. Media files for a particular component are stored in `$MEDIAFILEHOME/mpsN` directories.

Symbolic links `$MPSHOME/common` and `$MPSHOME/mpsN` determine current configuration of the secondary node. If they point to standby configuration directories, the node is in the standby mode. If they point to copies of primary components, the node is in the active mode.

Table 1: Sample Secondary Node MEDIAFILEHOME Directory Layout (Standby or Active)

```
(in $MEDIAFILEHOME/)
mps1/digitTable      /* Will contain dtmf.mmd and
                    dtmf.mmi */
mps1/system          /* All System wide MMFs for MPS1
                    will be in this dir */
mps1/system/record   /* The system wide record MMF file
                    for MPS 1 goes in this directory */
mps1/appname1        /* All MMFs specific to appname1
                    will be in this dir */
mps1/appname1/record /* The appname1 specific record file
                    goes in this dir */
. . .
. . .
. . .
mps1/appnameN        /* All MMFs specific to appnameN
                    will be in this dir */
mps1/appnameN/record /* The appnameN specific record file
                    goes in this dir */

mps2/...             /* SAME AS MPS1 */
mps3/...             /* SAME AS MPS1 */
mps4/...             /* SAME AS MPS1 */
```

Table 2: Sample Secondary Node Directory Layout (Standby)

```
(in $MPSHOME/)
common.standby/
mps10.standby/
mps11.standby/

common.tmsi01/
mps1/
mps2/
miscfiles.tmsi01/

common.tmsi02/
mps3/
mps4/
miscfiles.tmsi02/

mps0.empty/

common (symbolic link) --> common.standby
mps10 (symbolic link) --> mps10.standby
mps11 (symbolic link) --> mps11.standby
```

Table 3: Sample Secondary Node Directory Layout (Active)

```
(in $MPSHOME/)
common.standby/
mps10.standby/
mps11.standby/

common.tmsi01/
mps1/
mps2/
miscfiles.tmsi01/

common.tmsi02/
mps3/
mps4/
miscfiles.tmsi02/

mps0.empty/

common (symbolic link) --> common.tmsi01
mps10 (symbolic link) --> mps1
mps11 (symbolic link) --> mps2
```

Least Cost Routing Daemon

If the Least Cost Routing (LCR) daemon is configured on a primary node then you must do the following:

- Run the `lcrinstall` script on the secondary node.
- Add the secondary node name to the `$MPSHOME/common/etc/nameservers.cfg` file on each primary node.

Redundancy Configuration Daemon (RCD)

Redundancy Configuration Daemon (RCD) is the process that runs in the `common/gen` group on every node in the cluster and is responsible for:

- Configuration setup on the secondary node during failover and failback.
- Configuration synchronization between primaries and the secondary.

The configuration synchronization feature keeps configuration, media, and other files in sync between primary and secondary nodes. To sync the files:

1. Secondary RCD (RCD residing on a secondary node) requests a list of files that needs to be kept in sync from every primary RCD (together with files last modification times).
2. Secondary RCD compares last modification times of the local and remote copies of the files and marks a file for update if the times do not match.
3. Secondary RCD pulls files that are marked for update from a primary node into a staging area and sets the last modification time corresponding to the remote copy to eliminate effect of non-synchronized clocks.
4. If all the files are transferred successfully, the outdated files are replaced with the fresh copy.



If files or directories are manually deleted from the primary node, the corresponding files or directories must also be manually deleted from the secondary node.

The following types of files can be kept synchronized:

- **Component-specific configuration files** (e.g., `trip.cfg`). Files and directories of this type reside in the `$MPSHOME/common` and `$MPSHOME/mpsN` directories on a primary node. Files located in the `$MPSHOME/mpsN` directory are copied to the same location on a secondary node, and files located in `$MPSHOME/common` are copied to the `$MPSHOME/common.<primary_node_name>` directory on the secondary node (except that `$MPSHOME` can be different). You can specify which component-specific files (or directories) will be kept in sync by using the RCD option **syncCompDirs** on a primary node.

- **Media files** (in the `$MEDIAFILEHOME` directory). Files and directories of this type reside in the `$MEDIAFILEHOME/mpsN` directory on a primary node and copied to the same location on a secondary node (except that `$MEDIAFILEHOME` can be different). You can specify which media files (or directories) will be kept in sync by using the RCD option **syncMediaDirs** on a primary node.
- **Miscellaneous files** (e.g., a database file). Files and directories of this type can reside anywhere on a primary node and are copied to a secondary node to a location rooted in the directory `$MPSHOME/miscfiles.<nodename>` (e.g., file `tmsi01:/opt/dbfile` would be copied to `tmsi100:/opt/vps/miscfiles.tmsi01/opt/dbfile`, assuming that `$MPSHOME` expands to `/opt/vps` on `tmsi100`). Upon failover, before going active, secondary RCD puts these miscellaneous files in places creating symbolic links in corresponding locations (e.g., `/opt/dbfile --> /opt/vps/miscfiles.tmsi01/opt/dbfile`). This is done to avoid clashes between files copied from different primary nodes. If a link cannot be created because a same-named directory exists, that directory is removed (if it contains no files, and all parent directories contain no files) and the link is created. The links are removed by RCD during failback (or by the `setstandby.pl` script).

Configuration synchronization can be done manually (using RCD option **sync**) or automatically (periodically) using RCD option **syncInterval**. RCD configuration commands are typically set in the `$MPSHOME/common/etc/rcd.cfg` file (on a primary), and the `$MPSHOME/common.standby/etc/rcd.cfg` file (on the secondary).

Refer to the *MPS Command Reference Manual* for details of RCD commands.

The Failover/Failback Process

1. Upon startup in a standby configuration, each secondary TRIP attempts to connect to all TMSs for which it is listed as secondary in the `$MPSHOME/common.standby/etc/tms/tms.cfg` file. All primary components controlling these TMSs reside on different nodes.
2. Upon receiving a signal from the TMS that it lost its primary controller (i.e., the primary component failed), TRIP informs the local RCD that failover is required. RCD sets its state to “failover initiated” and sets the secondary node configuration to match the configuration of the primary node (using backup local copies of primary MPS and COMMON components). If the primary node has fewer components than the secondary node, unused components are pointed to empty configuration (`$MPSHOME/mps0.empty`). Once failover is initiated by RCD, all subsequent failover requests from TRIPs are ignored (i.e. the first primary node whose failure is detected by the secondary will be taken over). After changing configuration to the primary node’s configuration, RCD restarts local COMMON and MPS components to force them to load the new configuration.
3. When components are restarted, RCD waits 5 minutes for local TRIPs to acquire TMSs. If the TRIPs cannot acquire TMSs, the TMSs may need to be restarted. This is reported as the “grace period” in the “Local MPS States” panel in the RCD status report. The grace period ends when either:
 - all local MPS components (excluding those loaded empty configuration) acquired their TMSs.
 - 5 minutes have elapsed
4. Redundancy is implemented on a node-by-node basis. Once entering the active state, the secondary node protects all MPS components on the failed primary node for current and future failures but does not protect other primary nodes.

For example a primary node runs two MPS components. When one of them fails, secondary node will enter the active state and take over the failed component and also will monitor the second primary component for failures. When the second component failure is detected, the secondary node will take it over. The RCD status report shows whether a secondary MPS component acquired the TMS.

5. Once the failed primary node is operational again, it can request failback (see [TRIP Failback on page 268](#)) from the secondary node. Once failback is requested, the secondary TRIPs release their TMSs, allowing primary TRIPs to take control. Once all TMSs are released, the secondary node goes back to standby state, again protecting all primary nodes in the cluster.

Alarms are displayed as systems change state. See the *MPS Alarm Message Reference Manual* for information regarding the alarms and their meanings.

Installation and Configuration

Create the Secondary Node

The secondary node must have all standard software packages installed to make a fully functioning MPS (PERIfw, PERIglobl, PERItms, PERImps, etc.). The secondary node must also have the same number of MPS components installed as the highest populated primary node. For example, if the largest primary node has four MPS components, the secondary node must also have four MPS components installed.

After installing all software packages and associated patches, run the script `config_redundant_node.pl` to convert the node to a secondary node.



Make sure all Avaya services are not running.

```
perl $MPSHOME/bin/config_redundant_node.pl
```

This script performs the following:

- Renames `$MPSHOME/common` to `$MPSHOME/common.standby` and replaces `gen.cfg` with “minimal” `gen.cfg` that sets RCD to run in standby mode. The original `gen.cfg` is moved to `gen.cfg.bak`.
- Renames each local component directory `$MPSHOME/mpsN` to `$MPSHOME/mpsN.standby`, replaces `vos.cfg` with “minimal” `vos.cfg` that starts only TCAD and TRIP in standby mode, and removes `ase.cfg` and `aseLines.cfg`. All old configuration files are stored in the files with `.bak` extension.
- Creates `$MPSHOME/mps0.empty` directory.

TMSCOMM Component Configuration

For information about configuring the TMSCOMM component, refer to:

- [The `\$MPSHOME/common/etc` Directory](#) on page 88.
- [The `\$MPSHOME/tmscommN` Directory](#) on page 138.
- [Configuring the MPS Application Processor](#) in Chapter 3 of *Installing MPS Software on the Solaris Platform*.

Edit the vpshosts File

Edit the `vpshosts` in the `$MPSHOME/common.standby/etc` directory to list all MPS components (local and remote) in the cluster. Include `tmscomm` if `tmscomm` is running on the secondary node. **DO NOT include Speech Server components:**

```
$1
#
# Sample VPSHOSTS for a secondary system
#
# COMP NODE      TYPE
   1   tmsi01    mps
   2   tmsi01    mps
   1   tmsi01    tmscomm
   3   tmsi02    mps
   4   tmsi02    mps
   2   tmsi02    tmscomm
  10   -         mps
  11   -         mps
```

The `vpshosts` file on each primary system must list all MPS components in a cluster, including all secondary MPS components. The following is sample `vpshosts` for a primary system:

```
$1
#
# Sample VPSHOSTS file for a primary system
#
# COMP NODE      TYPE
   1   -         mps
   2   -         mps
   1   -         tmscomm
   3   tmsi02    mps
   4   tmsi02    mps
   2   tmsi02    tmscomm
  10   tmsi100   mps
  11   tmsi100   mps
```



All hostnames listed in `vpshosts` must also appear in the `/etc/hosts` file, and must be associated with an IP address on the Avaya LAN. Hostnames cannot exceed 18 characters in length.

Edit the tms.cfg File

Edit the `tms.cfg` file to designate primary and secondary components for each TMS in the cluster. Note that in N+1 configuration a component can be secondary for multiple primary components. The following is an example of BIND statements in a `tms.cfg` file:

```
-----  
;      Chassis  Backplane  TMS      Primary  Secondary  
;      Num      Slot (BPS)  Num      Comp#    VOS Comp#  Config  
-----  
BIND  1         1         1        1        10        BasicConfig  
BIND  1         2         2        2        11        BasicConfig  
BIND  2         1         3        3        10        BasicConfig  
BIND  2         2         4        4        11        BasicConfig
```

The TMS Number column determines hostname of a TMS board; e.g., the TMS 3 has hostname `tms3`. Hostnames of all TMS boards listed in the `tms.cfg` file must appear in the `/etc/hosts` file.



The `tms.cfg` file must be the same across all systems in a cluster, including the `$MPSHOME/common.standby/etc/tms/tms.cfg` file on the secondary node.

Edit TRIP and RCD Configuration Files

Edit the `$MPSHOME/mpsN/etc/trip.cfg` and `$MPSHOME/common/etc/rcd.cfg` files on the primary nodes, and `$MPSHOME/common.standby/etc/rcd.cfg` on the secondary node to set the desired redundancy related parameters.

RCD: `compDirSpec`, `syncCompDirs`, `syncInterval`, `syncMediaDirs`, `syncMiscDirs`

TRIP: `auto-failback`, `restart-tms`

Refer to the *MPS Command Reference Manual* for details regarding TRIP and RCD commands.

Edit the gen.cfg file

Add/uncomment the following line in the `$MPSHOME/common/etc/gen.cfg` file on each primary, and the `$MPSHOME/common.standby/etc/gen.cfg` file on the secondary:

```
rcd - - 1 0 rcd
```

PMGR configuration

As the primary mps numbers will not match the secondary mps numbers, the pmgr on each primary may need to be configured for both the primary and secondary component numbers. This is only required when the pools defined on the primary are setup with specific MPS component numbers (instead of using the wildcard "*" character). Each of these pools that is setup with a specific MPS component must be duplicated in the `$MPSHOME/common/etc/pmgr.cfg` file with the corresponding secondary component number.

For example a primary (tmsi01) with MPS components mps1 and mps2 could be configured with the following pools in `$MPSHOME/common/etc/pmgr.cfg`:

```
defpool line.out
cfgrsrc line.out,phone.168-192.mps.1
cfgrsrc line.out,phone.121-144.mps.2

defpool fax.1
cfgrsrc fax.1,fax.*.mps.1
defpool fax.2
cfgrsrc fax.2,fax.*.mps.2
```

When this node (tmsi01) is configured to be part of an N+1 group where its backup components are mps10 and mps11 then the following pools must be added to the `$MPSHOME/common/etc/pmgr.cfg` file of tmsi01 (the primary):

```
cfgrsrc line.out,phone.168-192.mps.10
cfgrsrc line.out,phone.121-144.mps.11
defpool fax.10
cfgrsrc fax.10,fax.*.mps.10
defpool fax.11
cfgrsrc fax.11,fax.*.mps.11
```

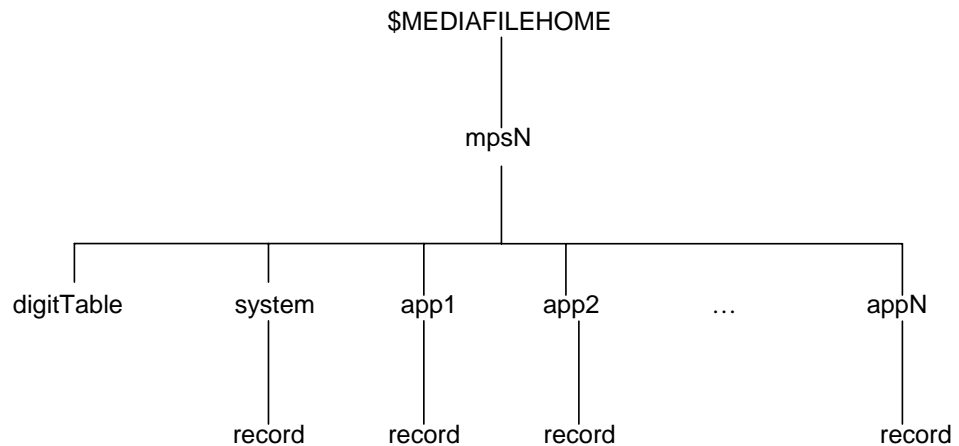
Media Directories

Each primary and secondary node must be modified to utilize the \$MEDIATYPEHOME directory structure for activating its MMF files (instead of using the \$MPSHOME/mpsN/etc/vmm.cfg and \$MPSHOME/mpsN/etc/vmm-mmfcfg files). Refer to [Activating MMF Files on page 166](#).

Primary Configuration

Follow these steps to configure each primary to utilize media directories:

1. Create the Media Directory (i.e. /mmf/peri).
2. Update \$MPSHOME/common/etc/siterc.sh or /etc/vpsrc.sh and /etc/vpsrc.csh for environment \$MEDIATYPEHOME settings.
3. Create a \$MEDIATYPEHOME/mpsN directory for each MPS component on that primary.
4. The directory structure of each \$MEDIATYPEHOME/mpsN directory on the primary must be set up as shown in the following figure:



Note that the app1 . . . appN directories are only required when you have application specific MMF files.

5. Comment out the "tonetable" entry in each \$MPSHOME/mpsN/etc/vmm.cfg file on the primary.
6. Move the dtmf.mmd and dtmf.mmi files into each \$MEDIATYPEHOME/mpsN/digitTable directory.
7. Comment out all "mmfload" entries in each \$MPSHOME/mpsN/etc/vmm-mmfcfg file on the primary.
8. Move the system wide MMF files for each MPS component to its matching \$MEDIATYPEHOME/mpsN/system directory.
9. Move the system wide record MMF file for each MPS component to its matching \$MEDIATYPEHOME/mpsN/system/record directory

10. Move the application specific MMF files for each MPS component to its matching `$MEDIAFILEHOME/mpsN/appN` directory.
11. Move the application specific record MMF files for each MPS component to its matching `$MEDIAFILEHOME/mpsN/appN/record` directory.
12. Reboot the primary

Repeat steps 1-12 for each primary in the N+1 cluster.

Secondary Configuration

The synchronization daemon creates component media directories off the `$MEDIAFILEHOME` directory if they do not exist. These directories will physically reside on the same disk partition as the `$MEDIAFILEHOME` directory. For performance reasons, you may want media files for different components to reside on different disk partitions. To implement this, manually create component media directories and mount on disk partitions.

For example, media files for MPS1 will reside on partition `/dev/dsk/c0t0d0s4` and media files for MPS2 will reside on partition `/dev/dsk/c0t0d0s5`. The following commands create the proper configuration:

```
sh> cd $MEDIAFILEHOME
sh> mkdir mps1
sh> mount /dev/dsk/c0t0d0s4 $MEDIAFILEHOME/mps1
sh> mkdir mps2
sh> mount /dev/dsk/c0t0d0s5 $MEDIAFILEHOME/mps2
...
```

Follow these steps to configure the secondary to utilize media directories. Note that all primaries must be up and running prior to executing these steps.

1. Create the Media Directory (i.e. `/mmf/peri`)
2. Update `$MPSHOME/common/etc/siterc.sh` or `/etc/vpsrc.sh` and `/etc/vpsrc.csh` for environment `$MEDIAFILEHOME` settings.
3. Reboot the secondary.
4. Using `vsh`, choose the common component, type "rcd sync all". This will cause RCD to synchronize with its primaries (including copying the media directories that were setup on each primary).
5. Using the `mkmf` command create duplicates of the record files, if any, from each primary in `$MEDIAFILEHOME/mpsN/system/record` and `$MEDIAFILEHOME/mpsN/appN/record` directories.

Refer to the *MPS Command Reference Manual* for details regarding the `mkmf` command.



Duplicate record files must be manually created on the secondary as record files are not synchronized.

First Startup After Configuration

After the cluster is properly configured and after initial startup, copy the configuration of all primary nodes to the secondary node:

1. Start up all primary nodes in the cluster. They all must have RCD in their common component.
2. Setup standby configuration on the secondary node by running the script `setstandby.pl`:

```
perl $MPSHOME/bin/setstandby.pl
```

Note that on node reboot, this script is automatically run by the `/etc/rc3.d/S17mps.setstandby` script.

3. Start SRP on the secondary node. All TRIPs and TCADs should enter the *standby* state.

If automatic configuration synchronization is enabled (e.g., the **syncInterval** option is set to non-zero), configuration will be automatically copied from all primary nodes. The first configuration synchronization is completed when the "Auto Sync Count" field in the RCD status report increments to non-zero.

If automatic configuration synchronization is disabled (e.g., the **syncInterval** option is set to zero), use **rcd sync** to initiate configuration synchronization:

```
vsh> rcd sync all
```

To monitor the synchronization, enable the **rcdu_updates** debug object before using the **sync** command:

```
vsh> rcd dlogDbgOn display,rcdu_updates
```

The format of the debug records is:

```
<code> <filename> <optional_text>
```

where code is one of:

- x - File does not exist and will be copied from the primary node
- e - Error checking file modification time
- s - File exists but is outdated and will be copied from the primary node
- - File exists and is up to date

After the **sync** command completes, all primary configurations are replicated on the secondary node and it is ready for failovers. Use the **srp -status** and **rcd status** commands to monitor the status of failover.

Table 4: SRP Status Report for Secondary Node in Standby

NODE:PORT	USER	PID	LINE	STATE	ENTERED STATE	FLAGS	CMDLINE
tmsi100:5999	root	19031	-	RUNNING	Jan 03 11:29:00	C	srp
Component: #common.0,gen/tmsi100							
tmsi100:40588	root	19032	-	RUNNING	Jan 03 11:29:01	C	alarmd
tmsi100:40592	root	19033	-	RUNNING	Jan 03 11:29:01	C	configd
tmsi100:40597	root	19034	-	RUNNING	Jan 03 11:29:01	C	conout
tmsi100	root	19035	-	RUNNING	Jan 03 11:29:01		rpc.riod
tmsi100:40601	root	19036	-	RUNNING	Jan 03 11:29:01	C	nriod
tmsi100:40605	root	19037	-	RUNNING	Jan 03 11:29:01	C	rcd
Component: #mps.101,vos/tmsi100							
tmsi100:40610	root	19038	-	STANDBY	Jan 03 11:29:08	C	trip
tmsi100:40621	root	19040	-	STANDBY	Jan 03 11:29:08	C	tcad
Component: #mps.102,vos/tmsi100							
tmsi100:40613	root	19039	-	STANDBY	Jan 03 11:29:08	C	trip
tmsi100:40624	root	19041	-	STANDBY	Jan 03 11:29:08	C	tcad

Table 5: RCD Status Report for Primary Node

```

RCD mode ..... primary
RCD state ..... ready

=====
Component Mappings
=====

Primary MPS#           Secondary MPS#         TMS#
-----
mps.1/tmsi01           mps.11/tmsi04         tms1
mps.2/tmsi01           mps.12/tmsi04         tms2

=====
Component subdirectories which are synchronized with secondary node
=====

Component Type         Subdirectories
-----
mps                    etc:apps
common                 etc

=====
Media subdirectories which are synchronized with secondary node
=====

Component Type         Subdirectories
-----
mps                    <all>

=====
Misc.subdirectories which are synchronized with secondary node
=====

/opt/dbfile
    
```

Table 6: RCD Status Report for Secondary (Standby)

```
RCD mode ..... secondary/standby
RCD state ..... ready

Auto sync period ..... 900 sec
Auto sync count ..... 1

=====
Component Mappings
=====

Primary MPS#      Secondary MPS#    TMS#
-----
mps.1/tmsi01     mps.11/tmsi04   tms1
mps.2/tmsi01     mps.12/tmsi04   tms2
```

Table 7: RCD Status Report for Secondary (Active)

```
RCD mode ..... secondary/active
RCD state ..... ready

=====
Component Mappings
=====

Primary MPS#      Secondary MPS#    TMS#
-----
mps.1/tmsi01     mps.11/tmsi04   tms1
mps.2/tmsi01     mps.12/tmsi04   tms2

=====
Local MPS States (grace period)
=====

mps.11 ..... active
mps.12 ..... idle
```

Verifying N+1 Functionality



This section assumes that the MPS cluster is fully configured, all nodes are up and running, the secondary is running its standby configuration, and the secondary has performed at least one successful file synchronization for each primary.

This test should be executed multiple times for primaries with varying numbers of MPS components (if available), as well as with different configurations of auto-failback commented out, and set to hard in the trip.cfg files on the primary.

Failover

Initiate a failover from one of the primaries in one of the following ways:

- Unplug the Ethernet cable from the MPS/node that connects that MPS/node to the TMS LAN (this requires that the MPSs are configured to communicate with each other over the private AVAYA LAN (which is the same as the TMS LAN)).
- Shutdown the MPS/node.
- Remove power from that MPS/node (this is a rather drastic test which may result in file system corruption resulting in the need for user intervention to bring the node back up).
- Stop SRP on the Primary MPS/node (via services on windows and /etc/rc3.d/S20vps.startup script on Solaris).



Depending on the method used to initiate the failover and the health check interval it can take 60+ seconds before the secondary attempts to take control of the TMS(s).

Follow these steps to verify that the failover was successful:

1. On the secondary, verify that SRP and all COMMON and MPS components are in the running state using:

```
vsh> srp -status
```

2. On the secondary, verify that the following alarm appears for every MPS component of the failed primary (the following is just an example of the alarm. The "#mps.10" is really "#mps.N", where N is the primary MPS component number that failed over).

Make sure that the number of these alarms exactly matches the number of primary MPS components (extra or too few alarms are an indication of a problem).

```
Fri Jun 10 12:21:41 <trip> 10001 Severity 1 Comp #mps.10/tmsi01  
Entering ACTIVE state for #mps.1
```

In the first line, #mps . 10 is the standby MPS.

In the second line, #mps . 1 is the failed primary MPS.

3. Verify that the secondary is running the correct applications for the primary it replaced.

4. Verify that the MPS and COMMON components started on the secondary are running the correct processes that were running on the primary (i.e., if MPS comp was running confmgr and/or faxmgr on the primary they must now be running on the secondary).

Failback

Initiate a failback to the failed primary. This should be done only after the above failover was successful (preferably wait at least 60 seconds). The steps to be taken depend on the steps performed to cause the failover:

- If the Ethernet was unplugged to cause the failover then:
 - pre MPS2.1 PB10 plug the Ethernet cable back in and restart the MPS.
 - post MPS2.1 PB10 simply plug the Ethernet cable back in.
- If the primary was shutdown then reboot it.
- If power was removed from the primary then reapply power and boot the system (fixing the file system if necessary).
- If the MPS software was stopped then restart it. Start SRP on the Primary MPS/node (via services on windows and /etc/rc3.d/S20vps.startup script on Solaris).

In all of the above cases a failback will either already be initiated upon restart of the MPS software if the automatic failback mechanism was enabled, or will require the user to enter the following vsh command from the previously failed primary to manually initiate the failback:

```
vsh> trip auto-failback hard
```

Follow these steps to verify that the failback was successful:

1. On the secondary, verify that SRP and COMMON are in the RUNNING state while all MPS components are in the STANDBY state using:

```
vsh> srp -status
```
2. On the secondary, verify that the following alarm appears for every MPS component the secondary is configured for:

```
Fri Jun 10 11:53:06 <trip> 10002 Severity 1 Comp #mps.10/tmsi01  
Entering STANDBY state
```

In the first line, #mps.10 is the standby MPS.

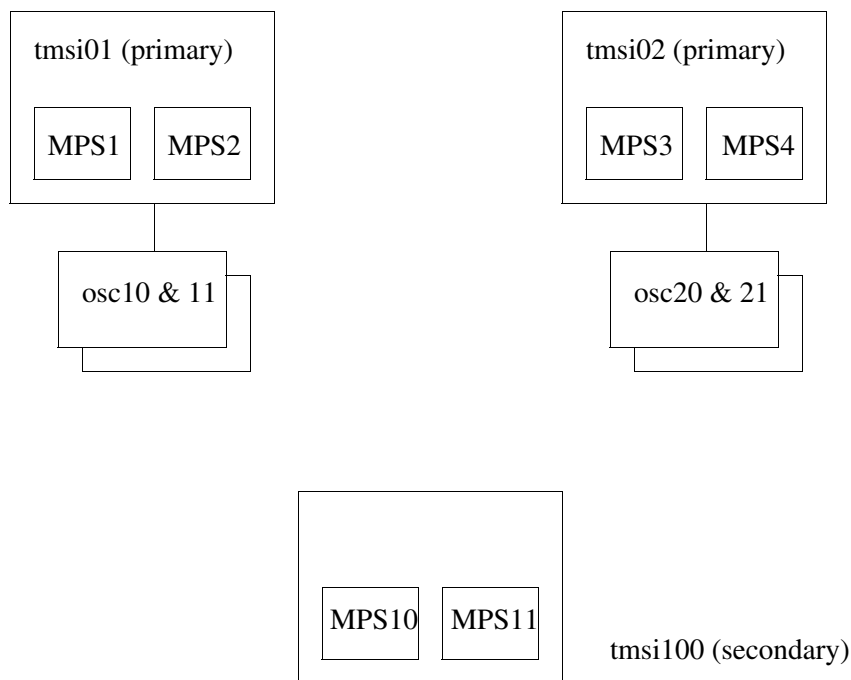
3. On the primary verify that SRP, COMMON, and all MPS components are in the RUNNING state using:

```
vsh> srp -status
```

Speech Server Resources in N+1 Redundancy

Speech Server resources must be included in the primary node(s) `vpshosts` file(s). During failover, the secondary node's Pool Manager (PMGR) determines which Speech Server resources to use according to the entries in the primary `vpshosts` file(s). Speech Server nodes must **NOT** contain entries in their `vpshosts` files for secondary (backup) components. The `$MPSHOME/common.standby/etc/vpshosts` file on the secondary node must **NOT** contain entries for the Speech Server nodes.

For example, in the following configuration:



the `vpshosts` files are:

```
# vpshosts (tmsi01)
#
# COMP NODE      TYPE
   1  -          mps
   2  -          mps
   3  tmsi02     mps
   4  tmsi02     mps
  10  tmsi100    mps
  11  tmsi100    mps
  10  osc10      oscar
  11  osc11      oscar
```

```
# vpshosts (tmsi02)
#
# COMP NODE      TYPE
   1 tmsi01 mps
   2 tmsi01 mps
   3 -        mps
   4 -        mps
  10 tmsi100 mps
  11 tmsi100 mps
  20 osc20    oscar
  21 osc21    oscar
```

```
# vpshosts (osc10)
#
# COMP NODE      TYPE
   1 tmsi01 mps
   2 tmsi01 mps
  10 -        oscar
```

```
# vpshosts (osc20)
#
# COMP NODE      TYPE
   3 tmsi02 mps
   2 tmsi02 mps
  20 -        oscar
```

```
# $MPSHOME/common.standby/vpshosts
# (tmsi100)
#
# COMP NODE      TYPE
   1 tmsi01 mps
   2 tmsi01 mps
   3 tmsi02 mps
   4 tmsi02 mps
  10 -        mps
  11 -        mps
```

When adding another Speech Server to an existing Speech Server group:

1. On the new Speech Server node, add entries into the `vpshosts` file for all MPS components that the Speech Server serves.
2. On the MPS nodes which the new Speech Server will serve, add an entry in the `vpshosts` file(s) for the new Speech Server.
3. Verify that the new configuration works. Force a synchronization using the **`rcd sync`** command.

Pool Manager (PMGR)

Pool manager (PMGR) is a process that runs on the MPS common component. PMGR provides a facility to configure MPS resources into pools. An application can request a given type of resource from a configured pool, or a specific instance of the resource class. Even if a specific resource instance is requested and is not available, PMGR can allocate another instance of the resource class from a configured pool.

In order for a resource to be available for configuration, allocation, or statistics collection by PMGR, the resource must be registered with PMGR by a resource provider. A resource provider is any process that manages resources. For example, CCM provides phone lines as resources. Resource providers automatically detect what resources are available upon system startup and registers those resources with PMGR.

Terminology

Resource Object

A reusable hardware (physical) or software (logical) entity that can be used by an application to perform a particular task. Commonly and simply referred to as 'a resource'. A resource object has the following attributes that define it:

Resource Object Attributes

Attribute	Description
<code>rsrcClass</code>	The type of resource (e.g., line, app, dtmf, tgen, etc.). Sometimes referred to as the resource base class.
<code>rsrcInst</code>	The instance (number) of this resource class. Needed for resource objects that have the same <code>rsrcClass</code> . The instance must be unique for each resource object of the same <code>rsrcClass</code> in the same component.
<code>rsrcCompType</code>	The type of component providing the resource (e.g., mps, oscar, etc.).
<code>rsrcCompId</code>	The resource component identifier (number).

Allocation/Deallocation

Allocation is the process by which PMGR makes a resource object available to an application. Once a resource is allocated to an application, that application becomes the owner of that resource and the resource is unavailable for allocation to other applications. The owner application must (implicitly or explicitly) deallocate the resource before exiting. Once a resource is deallocated, PMGR returns it to the pool in which it is configured, making the resource available for allocation to another application.

Pool

A grouping of zero or more resource objects that can be allocated by PMGR to applications. A pool is defined using a **pmgr defpool** command or statement in the `pmgr.cfg` file. (See [Configuration on page 289](#).) When initially defined, the pool is empty. Resources are added using the **pmgr cfgrsrc** command.

Resource Identifier/String

Uniquely identifies a single resource object (identifier) or multiple resource objects (string) on the MPS network. A resource identifier or resource string has the same format:

```
<rsrcClass>.<rsrcInst>.<rsrcCompType>.<rsrcCompId>
```

The fields that comprise the identifier/string are described under [Resource Object on page 288](#). The `<rsrcInst>` and `<rsrcCompType>` fields are number list type arguments. Either a single value or multiple values can be specified. A range of values is expressed by the starting and ending values (inclusive) separated by a hyphen (e.g., 1-24). Multiple non-contiguous values are expressed in a comma-delimited list (e.g., 3, 6, 12, 20). Any combination of ranges and comma-delimited lists can also be used (e.g., 2, 5, 9-13, 18-22, 24). The expression shall not contain spaces.

Example:

```
line.1-24.mps.1,5
```

Specifies the entire logical span 1 (T1 system) on MPS components 1 and 5, for a total of 48 lines.

Scheme

An algorithm that defines the rules by which PMGR will allocate resources from a pool. For the current release, only the round-robin (`rr`) allocation scheme is supported.

Configuration

Configuration of PMGR is performed by the `pmgr.cfg` file located in `$MPSHOME/common/etc`. A Pool is defined by including a **pmgr defpool** command statement in the `pmgr.cfg` file. Resources are configured into a defined pool by including one or more **pmgr cfgrsrc** command statements in the file. See the *PMGR Commands* section in the *Avaya Media Processing Server Series Command Reference Manual* for more information.

The sample `pmgr.cfg` file below defines three pools: `line.in` (for inbound lines); `line.out` (for outbound lines); and `line.ref` (referral lines). The outbound pool is configured on spans 1, 3, 5, and 7, and the inbound pool is configured on spans 2, 4, 6, and 8 on `mps12`. The referral pool is not used. Note that all lines for `mps11` are commented out, therefore, no pools are configured for `mps11`.

```
#
# Configuration file for PMGR process
#

#
# Enables debugging to a file
#
dlogDbgOn FILE,ERR
#dlogDbgOn FILE,GEN

#
# Enables debugging to STDERR
#
#dlogDbgOn STDERR,GEN
#dlogDbgOn STDERR,ERR

#
# Defines a new pool called 'poolname'.
#
#defpool poolname
#

defpool line.in
defpool line.out
defpool line.ref

#
# Configures the resources that belongs in each pool
#

cfgrsrc line,phone.*.vps.*

cfgrsrc line.in,phone.1-192.mps.11
#cfgrsrc line.out,phone.97-120.mps.11

#cfgrsrc line.in,phone.1-24.mps.11
#cfgrsrc line.out,phone.25-48.mps.11
#cfgrsrc line.in,phone.49-72.mps.11
#cfgrsrc line.ref,phone.73-96.mps.11
#cfgrsrc line.in,phone.97-120.mps.11
#cfgrsrc line.out,phone.121-144.mps.11
#cfgrsrc line.in,phone.145-168.mps.11
#cfgrsrc line.ref,phone.169-192.mps.11

cfgrsrc line.out,phone.1-24.mps.12
cfgrsrc line.in,phone.25-48.mps.12
cfgrsrc line.out,phone.49-72.mps.12
cfgrsrc line.in,phone.73-96.mps.12
cfgrsrc line.out,phone.97-120.mps.12
cfgrsrc line.in,phone.121-144.mps.12
cfgrsrc line.out,phone.145-168.mps.12
cfgrsrc line.in,phone.169-192.mps.12
```

Port Service States

By default, the system places all inbound-only ports out of service when an application is not bound to them. All other outbound-only and bi-directional ports are in an in-service state, regardless of whether or not an application is bound to them. For example, in the following configuration:

```
defpool line.in
defpool line.out

cfgrsrc line.in,phone.1-24.mps.1
cfgrsrc line.out,phone.25-48.mps.1
cfgrsrc line.in,phone.49-72.mps.1
cfgrsrc line.out,phone.49-72.mps.1
```

lines 1-24 are inbound only and are in an out-of-service state until an application is bound to them. Lines 25-48 are outbound only, and lines 49-72 are bi-directional, and they are in an in-service state until either the TMS takes them down or the user manually takes them down (using the CCM **outOfService** command, see the *Command Reference Manual*). Line pools, that are not explicitly defined as “line.in”, are assumed to be outbound only and stay in an in-service state.



Applications cannot bind to lines if the line is out of service and is not part of a “line.in” pool. Lines that are out of service and are not part of a “line.in” pool are automatically marked as unusable and removes those lines from ALL pools.

Network Failure Detection (Pinging)

The Pool Manager pings other remote PMGRs in the network to verify network operation. Pinging is enabled by default and should not be disabled (or its default values changed) unless call processing functions require maximum available network throughput.

Ping commands are set in `pmgr.cfg`:

pings on | off - Enable/disable remote PMGR pinging

pingmaxoutstanding # - Set the number of times a ping can be missed before PMGR considers the remote PMGR unreachable

pingperiod # - Set the amount of time between pings

Database Conversion

Conversion of databases from one format to another (HSAM, ISAM, DISAM) or between Solaris and Windows platforms is implemented with the PeriPro **dbcop**y application.

The **dbcop**y application (located in `ASEHOME/etc`) must be started on two different phone lines attached to the same `vmst` (any combination of line types may be used). One line serves as the reader (source database file) and the other line serves as the writer (target database file). The reader reads all the records from the existing database file and passes them, by means of phone-to-phone communications, to the writer, which stores the records in the target database file.

Platform Conversion

To convert a database file from one platform to another (e.g. from Solaris to Windows) the writer has to run on the target machine (in this case the Windows machine) while connected to `vmst` remotely. An empty target database file has to be created, using **dbman**, prior to the conversion. The source file must also have been defined using **dbman**, but *cannot* be empty. The following is an example of a platform conversion.

Starting a Reader

- A source database file must exist and be defined by **dbman**.
- Start either a real or simulated `vmst`.
- Start a `vengine` on line 201 (for example), specifying the source filename and the writer's phone line number using the **-O "read:(file:srcname) write:(line:number)"** option.

For example, to read from the (source) file `abc` use:

```
vengine -p201 -v1 -O "read:(file:abc)  
write:(line:202)" dbcopy
```

Starting a Writer

- A target database file must exist and be defined by **dbman**.
- Start `vengine` on line 202 (for example), specifying the target filename and reader's phone line number using the **-O "read:(line:number) write:(file:target)"** option.

For example, to write into the (target) file `xyz` use:

```
vengine -p202 -v1 -O "read:(line:201)  
write:(file:xyz)" dbcopy
```

- If the target machine is different from the source, then `vengine`'s command line has to define the node running `vmst` with the `-v` option for both the reader and writer (only one example is shown here):

```
vengine -p202 -v hostname:mps# -O "read:(line:201)
write:(file:xyz)" dbcopy
```

Database Format Conversion

Performing a database format conversion is similar to a platform conversion, except that the target database file (writer) created by `dbman` is of a different format.



Considerations:

- If the record length of the new file is less than the original, records are truncated.
- If the new file format doesn't allow copying of all the records (number of records for HSAM or unique/non unique keys for KSDS/DISAM) records are copied up to the violating record.

Reader/Writer Synchronization

By default, the reader waits one minute to connect to the writer. To change this timer, modify the `wait:hhmmss` parameter in the `-O` option:

```
vengine -p201 -v1 -O "read:(file:abc) write:(line:202)
wait:300" dbcopy
```

This example represent a 3 minute delay (00 hours, 03 minutes, 00 seconds).

File Size Limitations

Because of the `vmst` message size limitations, the maximum record size is limited to 4,000 bytes. Use TDQ files for larger records.

For further information on databases, applications, and their manipulation, see the *PeriProducer User's Guide*.

Call Monitoring

If you have access to the equipment, the MPS allows you to listen to calls in progress. Headphones/speakers can be connected to audio jacks in the MPS to play out the speech occurring on a chosen line. Only one side of a “conversation” can be monitored at a time.



You should use call monitoring (listening to the actual calls) only to confirm that caller/MPS interaction is going smoothly and that the MPS is operating correctly. There may be restrictions regarding the monitoring of telephone conversations. Check your local laws for guidelines on monitoring calls. This facility is intended ONLY for verification of operating speech paths from the caller and MPS sides.

Listening to Calls



When you have the system front panel removed, observe all electrical safety rules. Do not touch any exposed electronic circuits. After you have completed the monitor jack hookup, replace the front panel immediately.

In addition to usual electrical safety rules, you should also reduce or eliminate the chance of damage to your equipment by Electrostatic Discharge (ESD). Electrostatic discharge is a discharge of stored static electricity that occurs when electronic components are improperly handled and can result in complete or intermittent failures.

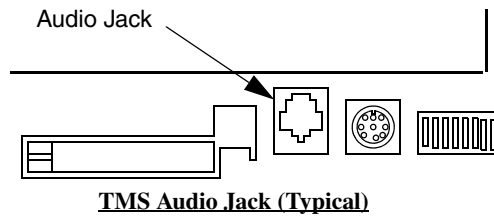
Before you open a chassis, ensure that power to the unit is turned off, but that the power cord is connected to the wall receptacle. Having the power cord connected will ensure a ground path for any ESD voltages. Wrist straps and work surfaces must also be properly grounded for protection against electrostatic discharge.

In a properly wired building the nearest reliable ground will likely be the center screw of a standard 110 VAC outlet. Avoid contact between equipment and clothing. The wrist or ankle strap only protects the equipment from ESD voltages on the body; ESD voltages on clothing can still cause damage.

The TMS front panel contains an audio jack used to monitor calls (see [Telephony Media Server \(TMS\) on page 28](#)). A telephone handset/headphone or a speaker is connected and the desired line can be monitored using the **ccm listen** command. Only one side of the conversation can be monitored from a single phone line at a time. To listen to calls:

1. Remove the front panel on the chassis containing the TMS.
2. Check that there is an active call on the line. Use the syntax **c#[-#] status** for that line (or range of lines), or **ccm status** for a report on all lines.

3. Plug the telephone handset into the AUDIO jack.



4. Issue the **ccm listen** command from the **vsh** shell on the node the TMS is associated with (see [vsh on page 60](#)). Use the following syntax:

```
c<line#> listen <option>
```

where **line#** is the number of the line you wish to monitor. The variable option can be one of:

- **inon** - turns monitoring on for inbound line
- **inoff** - turns monitoring off for inbound line
- **outon** - turns monitoring on for outbound line
- **outoff** - turns monitoring off for outbound line

5. Use the telephone headset, headphones, or speaker to monitor the call.



Once you select a line and side to monitor, they remain set until you change them. After your monitoring is complete, be sure to issue the **inoff** or **outoff** command option to quit active monitoring. Otherwise, the most recent **inon** or **outon** command prevails.

This page has been intentionally left blank.



5

Security

This chapter covers:

1. Antivirus
2. Secure Shell

Antivirus Software

To ensure the highest level of security, Avaya recommends (but does not require) the use of antivirus software on MPS Windows Application Processors (AP). MPS software has been extensively tested for compatibility with the McAfee Virus Scan and Norton AntiVirus software packages.

Avaya believes that the risk of virus infection on an AP is minimal due to the limited administrative access required to support the server. However, the need for antivirus software on a Windows-based PC is self evident. Avaya provides the following recommendations regarding antivirus software:

- When installing or upgrading system software, all antivirus functions must be disabled (i.e., firewalls, passive scanning, auto updates, etc.). Do not start the antivirus functions until all software installation procedures are complete.
- Virus scan software places additional load on the AP. Set automatic virus scans to run only during off-peak hours. If it is necessary to run manual scans during normal operations, the Windows Performance Monitor should be employed to monitor CPU utilization.
- Antivirus software can be configured to repair infected files automatically. If an infected file cannot be repaired automatically, do not attempt to replace or remove it. Contact your distributor or support representative for assistance.
- Do not connect the AP directly to the internet for downloading virus definitions and/or file updates. Instead, virus definitions and file updates should be downloaded to another agent on the network and manually loaded into the AP. This limits access to the internet and reduces the risk of downloading infected files.
- Scan CD-ROMs and floppy disks before installing or uploading to the AP. This minimizes any exposure to infected files from external sources.
- The antivirus software should be the latest available versions. Avaya recommends the following packages:
 - McAfee Virus Scan (version 6.02 or later)
 - Norton AntiVirus Corporate Edition (version 7.6 or later)



It must be known that system performance may get affected by the use of antivirus software if many resource-intensive applications are running together.

Secure Shell

Avaya NPopenSSH is a consolidated package of the Secure Shell openssh product and its requirements, such as `ssl` and `zlib`. This package is certified by Avaya Verification and Validation labs after tests on the sun4u architecture.

In case, your organization's security policy mandates use of your specialized implementation of secure shell, use the `pkgrm` command to remove NPopenSSH. Once it is removed, you can install your version implementation of the secure shell.

If your system was built by Avaya Manufacturing, then NPopenSSH will be installed automatically. To turn on `openssh`, execute the following as the root user:

- `cd /etc/rc2.d`
- `mv s98openssh S98opensshd`
- `./S99openssh start`

Securing the system:

After installing Secure Shell, perform the following actions to secure your system:

a) Disabling remote services

The `rsh` and `ftp` commands are replaced by `ssh` and `sftp` respectively. Remote services like `telnetd`, `rshd`, and `ftpd` can be disabled to force the use of secure shell. To disable them, edit `/etc/inetd.conf` and comment out the services.



You can disable services not in use. However, **DO NOT** disable `tftp`, `bootpd` or CCLP services in the `etc/inetd.conf`. as they are required to boot the `tms` and `mps` processors.

The following services should remain enabled if you use features that require them.

- `uucp` - required for `ppp` remote Avaya support
- `rstatd` - required for the `openwindows` performance monitor, and `PERIprpt` statistics reporting.
- `daytime` - required `synctime` to synchronize the time on one machine from with another.

b) Disabling X Windows (X11)

X Windows (X11) can be forced to operate over the secure shell, by disabling it from listening to the `tcp` connections.

Follow the following steps to disable X Window from listening to `tcp` connections:

- 1 Locate the `Xservers` files residing in `/etc/dt/config` and `/usr/dt/config`. Add the string `"-nolisten tcp"` to the uncommented line that contains `Xsun`.

Example:

```
:0 Local local_uid@console root /usr/openwin/bin/Xsun
:0 -nobanner -nolisten tcp
```

If there are no Xservers files in /etc/dt/config then copy Xservers from /usr/dt/config to etc/dt/config

Solaris patches may replace /usr/dt/config/Xservers, and /etc/dt/config which are considered to contain customized copies.

2. Disable the dtlogin from starting.
mv /etc/rc2/d/S99dtlogin /etc/rc2.d/s99dtlogin
The co line in /etc/inittab should specify respawn for the console login prompt to appear.

Example: c0:234:respawn:[...remainder-of-line...]

3. Reboot after performing both or either of the mentioned tasks.

General secure shell examples:

Using ssh:

Use "ssh machine1 -l user1" instead of "rsh machine1 -l user1"

Using sftp:

Use "sftp -oUser=user1 machine1" instead of "ftp machine1"

Tunneling X over the secure shell:

```
ssh -X machine1 -l user1
```

(Do not set the DISPLAY variable after you are logged in. Secure Shell will automatically forward the display to the machine you logged in from.)

Refer to the man pages for more information.

Index

Symbols

65, 66

\$MEDIAFILEHOME 166

A

ADSM port 47

alarm 51

Alarm, Diagnostic, and Statistics Management (ADSM) port 47

ALARMD 39, 40

ALI 28, 29, 30
and FPGA 32

Analog Line Interface
see also ALI

application processor 26

applications

administrative 35

assignment to phone lines 35, 36, 54, 57, 140

configuration files 140–141

development of 54

executable files 139, 140

IVR-to-Internet 59

shared libraries 139, 140

statistics collection and reporting 38, 50, 55, 58
PeriReporter 38, 50, 58

web-based 59

ASE (Application Services Environment) 35, 36–38, 148

SRP (Startup and Recovery Process) 65, 70–81

ENGINE 36, 54, 140

VFTPD (VPS File Transfer Protocol Daemon) 240–242

VMS (Voice Message Server) 225

VMST 36, 215

VSUPD 36, 38, 50, 127

VTCPD (VPS Transmission Control Protocol Daemon) 220

ASE/VOS integration layer 39

audio elements 56

auto-failback command 268

B

back end and front end 48

bays, drive 26

boot ROM 29, 32, 252
and TMS 32

DCC 252

NIC 260

TMS 256

bus synchronization 28

C

Call Control Manager

See also CCM

call monitoring 294

Call Progress Detection (CPD) 31

Call Simulator

preparing telephone lines 199

special considerations 199

standard command set 195

syntax errors 196

Call SIMulator (SIM) port 48

Caller Message Recording (CMR) 165

ethernet based 49

Call-Protect 288

CCM 39, 42

clocking

CT bus 265

synclist 119

commands

ncd nicbpsreset 264

tip 250

COMMGR 39

comp command 60

component list 60

CONFIGD 39, 43

configuration

file (tms.cfg) 106

PMGR 289

protocol (<proto>.cfg) 123

configuration commands

srp 75–81

vmm 166

See also Phone Line Manager, commands.

configuration file

CLASS 108

DTC_MAP 112

line card protocol package definitions 114

PARAM_SYS 106

RSET_PROFILE 106

RSET_TABLE 118

RSRC_CONFIG 108

SYNC_LISTS 119

SYSTEM 106

VPS_LINE_DEF 116

configuration files

.acfg 140–141

- .so 140
 - .vex 140
 - .xhtrahostsrc 86
 - .netrc 241
 - ase.cfg 148
 - aseLines.cfg 141, 149
 - ccm_phoneline.cfg 151, 155
 - commgr.cfg 43, 144
 - gen.cfg 96–98, 241
 - global_users.cfg 98
 - hosts 66, 83
 - nriod 45
 - perirc.sh 65
 - periview.cfg 102
 - rpc.riod 46
 - S20vps.startup 65, 67, 70
 - S30peri.plic 65, 67
 - services 129–132, 240
 - siterc.sh 66
 - srp.cfg 89–92
 - tcad.cfg 158
 - tcad-tms.cfg 157
 - trip.cfg 159
 - vftpd.cfg 242
 - vmm.cfg 145
 - vmm-mmf.cfg 146
 - vos.cfg 143–144
 - vpshosts 93–94
 - vpsrc.sh 65
 - configuration procedures
 - adding spans 244
 - changing pool/class names 245
 - modifying span resource set 244
 - CONOUT 39, 44
 - console connector, serial 29
 - CONSOLED 39, 44
 - control panel, front 22
 - conventions
 - manual 13
 - current component 60
 - current Media Processing Server
 - status 60
- D**
- database conversion 292
 - database format conversion 293
 - file size limitations 293
 - platform conversion 292
 - reader/writer synchronization 293
 - DCC 28, 29
 - and FPGA 32
 - boot ROM 252
 - debug (dumb) terminal 250
 - determining local node when remote 61
 - device, image definition for 32
 - Diagnostics, Logging, and Tracing (daemon) 53
 - Digital Communications Controller
 - See also* DCC
 - Digital Signal Processor
 - See also* DSP
 - directory structure 64, 87–89, 125, 137, 139–142
 - dlog** 51, 52
 - dlt** 51, 53
 - DSP 31
 - DTC_MAP 112
 - DTMF receiver 31
- E**
- eliciting component info 60
 - Ethernet
 - and NICs 27
- F**
- failback 268
 - Field Programmable Gate Array
 - See also* FPGA
 - file
 - tms.cfg 106
 - FPGA 32
 - Front Control Panel (FCP) 22
 - front end and back end 48
 - FTP (File Transfer Protocol) 240–242
- G**
- Graphical User Interface
 - See also* GUI
 - GUI 18
 - GUI manipulation of the Avaya Media Processing Server Series system 57
- H**
- host communications
 - TCP/IP 220
 - hostname** command 61
 - How To
 - enable statistics collection 249

- introduce a new MPS node [248](#)
- listen to calls (on the system) [294](#)
- rename a Solaris MPS node [246](#)
- rename a Windows MPS node [247](#)
- renumber a component [245](#)

Hub-NIC [27](#), [32](#)

hypertext links

- types of [11](#)
- use of [11](#)

See also manual: hypertext links in, using on line

I

image definitions, locations of [32](#)

K

killing SRP [245](#)

L

LAN

- and Ethernet interface [32](#)

LCR [271](#)

licensing, package [65](#)

listening to calls

- on the system [294](#)

Load Resource Management (LRM) port [47](#)

local [60](#)

local node, identifying [61](#)

location of hard drives [26](#)

log [51](#), [53](#)

LRM port [47](#)

ls [61](#)

M

manipulating audio elements [56](#)

manipulating the Media Processing Server Series system [57](#)

manual

- how to use [11](#)
- hypertext links in [11](#)
- using on line [11](#)

manual-failback command [268](#)

MDM [28](#), [30](#), [31](#)

- and FPGA [32](#)

Media Processing Server 1000

- major hardware components [21](#)

Media Processing Server Series

- network environment [19](#), [57](#), [93](#), [143](#)

- system utilities [51–61](#)

- alarm** [51](#)

- Alarm Viewer [57](#)

- dlt** [53](#)

- log** [53](#)

- PeriProducer [54](#), [58](#)

- PeriReporter [55](#)

- PeriStudio [56](#), [58](#)

- PeriView [57](#), [102](#), [140](#)

- PeriWeb [59](#)

- vsh** [60](#)

Media Processing Server Series processes

- and startup by SRP [35](#)

- what and where are they? [34](#)

Media Processing Server Series software environment defined [35](#)

MMF

- See also* MultiMedia files

monitoring

- calls [294](#)

Multi-Media File (MMF) [164–194](#)

- activating [167](#)

- application-specific files [174–175](#)

- creating [164](#)

- data (.mmd) file [164](#)

- deactivating [167](#)

- default files [175](#)

- delimited loading of vocabulary labels [168](#)

- disk-based speech output [169](#)

- elements [56](#), [164](#), [165](#), [168](#)

- hash tables [172–175](#)

- index (.mmi) file [164](#)

- loading elements into memory [171](#)

- memory management [165](#)

- partial loading of vocabulary labels [168](#)

- status commands [176](#)

- system-wide files [173](#)

- used as a vocabulary file [165](#)

- used with CMR [165](#)

- used with multiple applications [164](#)

MultiMedia files

- synchronizing across nodes [177](#)

Multi-Media Format (MMF) file [56](#)

- See also* MMF

Multiple DSP Module

- See also* MDM

mvcmp [245](#)

mvvps *See* **mvcmp**

MX API [39](#)

MXVMT [39](#)

N

N+1 redundancy 267
NCD 39, 45
Network Interface Controller
 See also NIC
NIC 27, 32
 boot ROM 260
 resetting 264
nriod 39, 45
NT startup services, 69

P

package licensing 65
PERIglobl 69
PERIgrs 69
PERIplic 69
PERIplic licensing mechanism 65
PeriPro 51, 54, 58
PeriProducer 51, 54, 58
PeriReporter 51, 55, 58
PERIsnmp 69
PeriStudio 51, 56, 58
PERItms
 directory structure 135
PeriView 51, 57
PERIweb 70
PeriWeb 51, 59
Phase-Lock Loop (PLL) 28
PMGR 40
 configuration 289
 feature 288
 terminology 288
power supplies
 and NIC slot 24
 in VRC 24
power switches
 for VRC slots 1 through 4 - 22
 for VRC slots 5 and 6 - 26
process
 PMGR 46
process, software diagram 34
processing, transaction 26
processor, application 26

R

rear panel, VRC 26
redundancy
 N+1 267

Redundancy Configuration Daemon 271
remote 60
resource class 108
resource limitations 111
ROM, boot 29
rpc.riod 40, 46
RSET_TABLE 118
RSRC_CONFIG 108

S

S20vps.startup, using 245
secure shell 299
serial console connector 29
SIM port 48
software process diagram 34
Solaris
 SPARC processor 21
SRP
 and startup of Media Processing Server Series
 processes 35
 restarting 245
 stopping 245
Startup and Recovery Process
 See also SRP
status, Media Processing Server 60
stopping SRP 245
synclist 119
system
 listening to calls 294
system hardware 21–33
system software 34–50

T

TCAD 40, 46, 269, 274
TCCP 33
TCP/IP connections 32, 220, 222, 224
TelCo connector panel (TCCP) 33
Telephony Media Server
 see also TMS
TMS 28
 and boot ROM 32
 and FPGA 32
 and LAN interface 32
 as replacement for CPS 18
 boot ROM 256
 in VRC 22
 motherboard 28
 number of supportable analog lines 29

number of supportable spans/lines [29](#)
phone line interfaces to [28](#)
redundancy/failback [268](#)

tms.cfg [276](#)

To [71](#)

Tone Generator (TGEN) [31](#)

transaction processing [26](#)

translation services between processes [39](#)

TRIP [40](#), [48](#), [274](#)

trip.cfg [268](#)

V

Variable Resource Chassis

See also VRC

VDM [49](#)

VENGINE [35](#), [36](#), [54](#)

VENGINE Message Server - Extended

See also VMST

VMM [40](#), [49](#)

and VDM [49](#)

VMS [215–219](#)

See VMST

VMST [36](#), [215](#)

as aliased [37](#), [129](#), [148](#), [215](#)

vocabulary file. *See* Multi-Media File (MMF).

Voice Data Memory

See also VDM

Voice Memory Manager

See also VMM

VOS (Voice Operating Software) [35](#), [39–50](#), [143](#)

ALARMD (alarm daemon) [40](#), [51](#)

CCM (Call Control Manager) [42](#)

COMMGR (Communications Manager) [43](#)

CONFIGD (configuration daemon) [43](#)

CONOUT (Console Output Process) [44](#)

CONSOLED (console daemon) [44](#)

NCD (Network Interface Controller daemon) [45](#)

nriod (remote input/output daemon) [45](#)

rpc.riod (remote input/output daemon) [46](#)

TCAD (TMS Configuration & Alarm Daemon)
[46](#)

TRIP (TMS Routing Interface Process) [48](#)

VMM (Voice Memory Manager) [49](#), [144](#)

VSTAT (VPS Statistics Manager) [50](#)

VPS_LINE_DEF [116](#)

vpshosts [275](#)

VRC [22](#)

backplane [32](#)

front control panel [22](#)

rear panel [26](#)

rear view and hardware in [24](#)

vsh [51](#), [60](#)

VSTAT [40](#), [50](#)

VSUPD [36](#), [38](#)

VTCPD [220](#)

W

Windows NT

certified programs [98](#)

control of SRP [71](#)

Intel processor [21](#)

pinging [72](#)

starting/stopping the system [71](#)

system startup [69–70](#)

Z

ZAP [177](#)

Zero Administration for Prompts (ZAP) [177–194](#)

alarm messages [44](#), [191](#)

distributed vs. selective synchronization [192](#)

exception processing [194](#)

MMF Abbreviated Contents (MAC) file [192](#), [194](#)

reference file [192](#)

VMM process [194](#)

zap command [191–193](#)