ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Integrated Management Interface
Developer Guide

December 2015

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

Contents

Preface v
Audience v
Conventions v
Contents v
Related Documents v
Support vi
Comments vi

CHAPTER 1 Management Interface Modules 5
Integrated Management Interface 5
IMI Shell 6
Tunneling and Transition 6

CHAPTER 2 Integrated Management Interface 7
IMI Components 7
IMI Architecture 8
Configuration Interfaces 8
NSM and IMI Interaction 9

CHAPTER 3 IMI Shell 11
System Commands 11
ZebOS-XP Commands 11
Communication Methods 11
Implementation 11
Special Commands 13
Message Format 13

CHAPTER 4 VLOG Module 15
Features 15
Debug Support for Protocol Modules 15
VR Builds 15
VLOG Users 15
Non-VR Operation 16
Architecture 16
External interfaces 16
Software Components 17
System Domain 19
Display 19
Log Files 20
IMISH Builds 20
Protocol Module Connectivity 20
IMI Restart 20
VLOG Restart 21
Protocol Module VR Debug Support 21

Data Flow 21
Static Model 22
Object Definitions 22

CHAPTER 5 Tunneling and Transitioning 25
Tunneling 25
Transitioning 25
IPv6 Transition - 6to4 25
IPv6 Transition - ISATAP 25
IPv6 Transition - GRE Tunnel 25

Index 27

Preface

This guide describes the application programming interface (API) for the Integrated Management Interface (IMI) component in ZebOS-XP.

Audience

This guide is intended for developers who write code to customize and extend IMI.

Conventions

Table P-1 shows the conventions used in this guide.

Table P-1: Conventions

Convention	Description
Italics	Emphasized terms; titles of books
Note:	Special instructions, suggestions, or warnings
`monospaced type`	Code elements such as commands, functions, parameters, files, and directories

Contents

This guide contains these chapters:

- Chapter 1, *Management Interface Modules*
- Chapter 2, *Integrated Management Interface*
- Chapter 3, *IMI Shell*
- Chapter 4, *VLOG Module*
- Chapter 5, *Tunneling and Transitioning*

Related Documents

The following guides are related to this document:

- *Integrated Management Interface Command Reference*
- *Installation Guide*

Note: All ZebOS-XP technical manuals are available to licensed customers at http://www.ipinfusion.com/support/document_list.

Support

For support-related questions, contact support@ipinfusion.com.

Comments

If you have comments, or need to report a problem with the content, contact techpubs@ipinfusion.com.

CHAPTER 1 Management Interface Modules

The ZebOS-XP management interface modules are a comprehensive set of tools to manage and control the ZebOS-XP routing and switching protocols:

- Integrated Management Interface (IMI)
- IMI Shell (IMISH)
- IPv6 Tunneling and Transition

These tools enable hardware manufacturers to seamlessly integrate a complete management plane with an Command Line Interface (CLI) to their routing and switching equipment, as well as rapidly build and provision enhanced IP services solutions for access equipment.

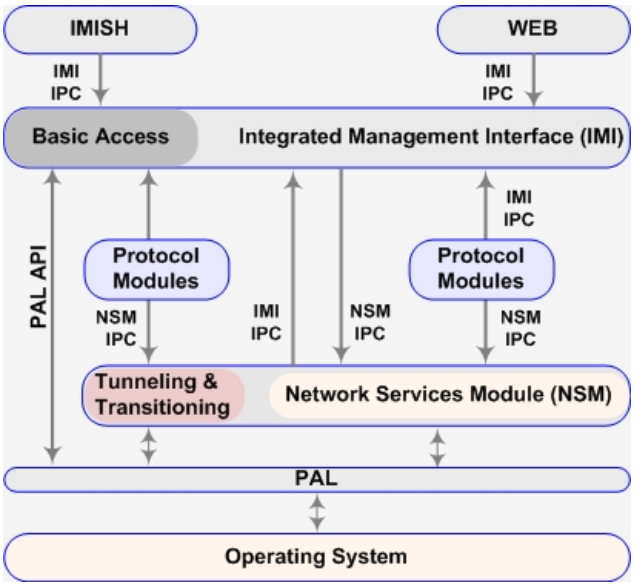


Figure 1-1: Management Interface Module Architecture

Integrated Management Interface

The IMI module offers complete, unified management of Network Services Module (NSM) and the individual ZebOS-XP protocols. It allows a system administrator to configure and monitor all ZebOS-XP daemons through one centralized user connection.

As a stand-alone management daemon, the IMI maintains a persistent connection to routing daemons, stores configuration data and offers extensive monitoring and logging capabilities.

IMI provides a feature-rich, hierarchical CLI that includes Exec, Privileged Exec, Configure, Router, and Interface command modes. It also supports syntax checking, command auto-completion and context-sensitive help.

IMI also allows the end-user to easily configure and manage network services featured in the Linux operating system, including DNS, DHCP, NAT, ACL, and PPPoE, using the Basic Access Module. The Basic Access Module offers a series of interfaces that enable equipment vendors to easily and rapidly integrate access software into their products and pass on the ease of manageability to their end users.

IMI Shell

IMISH is a client application that connects to the IMI and enables SSH or TELNET management and may be deployed on the routing equipment or on a separate management console. IMISH supports role-based management, leveraging the secure authentication method of the operating system to manage and validate user names and passwords. In addition, IMISH supports sophisticated input and output functions, for example to allow an administrator to save the output of a show command to a file.\

Tunneling and Transition

The IPv6 Tunneling and Transition Module provides tools and a CLI via NSM to configure and set parameters for IPv6-to-IPv4 transition and tunneling protocols such as 6to4, Intra-Site Automatic Tunnel Addressing Protocol (ISATAP), Generic Routing Encapsulation (GRE), and IP-in-IP encapsulation, provided by the Linux kernel. These features deliver enhanced performance, easy integration and management for access product OEMs and ODMs.

CHAPTER 2 Integrated Management Interface

The Integrated Management Interface (IMI) and the IMI Shell (IMISH) support all Layer 2 and Layer 3 components. IMI is an advanced management plane, integrating all features of the ZebOS-XP Layer 2 and Layer 3 components, key operating system components, and third-party applications.

IMI Components

IMI provides a centralized, persistent daemon for managing the ZebOS-XP configuration. It provides these advantages:

Persistent daemon. IMI maintains up-to-date configuration information about the entire system. In the event that protocol daemons are restarted, IMI can keep the configuration of the daemon before it is stopped, and configure them again after restart.

Configuration preservation. IMI saves configuration changes and uses them when protocol daemons are restarted. To save configuration information, users must use either the `write file` or `write memory` command. When a protocol module connects for the first time, it initiates a connection and requests IMI for the current saved configuration. IMI reads the configuration file and passes the configuration corresponding to the specific protocol module.

Central information control. IMI provides central control over the running and static configuration of all of ZebOS-XP protocols. The information can be merged together for display, configuration write, and persistent backup.

Master copy of common data. Several ZebOS-XP components receive the same types of information that each protocol uses for various purposes. IMI maintains a centralized copy of this information, off-loading several functions (such as display of active configuration) from the protocol modules.

Multiple access types. IMI provides a foundation for enabling several types of configuration options for the system. IMI supports the TELNET CLI, the IMISH, and Web configuration for Layer 2.

IMI Architecture

The IMI runs as a Server of the NSM and routing protocols, utilizing mechanisms established in the ZebOS-XP IMI IPC.

- NSM listens on NSM IPC and protocol modules and IMI initiates the connection.
- IMI listens on IMI IPC and NSM, protocol modules and IMISH initiates the connection.

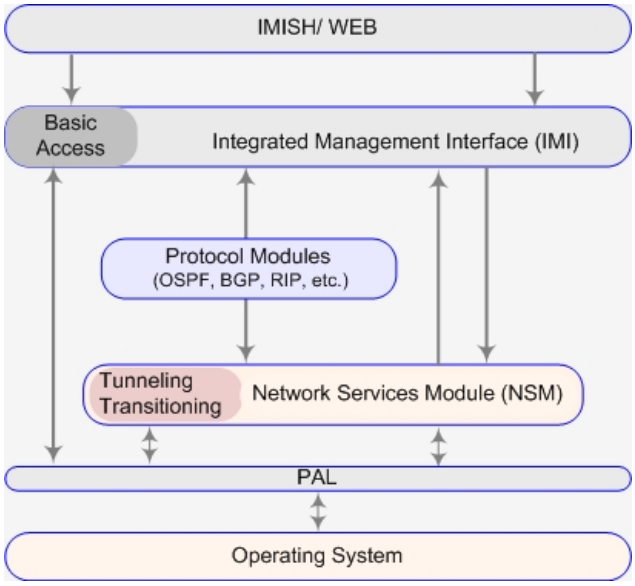


Figure 2-1: Integrated Management Interface Components

The IMI maintains a master database of data such as prefix-list, route-map, key-chain, access-list, and distributes this information to the protocols. This database enhances the capability and response to commands such as “show” or “write file”; the IMI does collect and organize data from the protocols, but displays the data directly.

Configuration Interfaces

Configuration of ZebOS-XP through the IMI can be accomplished in either of two ways: through the CLI directly or through a Browser-based application.

CLI API

The Command Line Interface for configuring ZebOS-XP is much the same as the IMI interface. The CLI contains only those commands that are valid for the ZebOS-XP protocols compiled.

IMI combines the techniques of the IMISH Line and the ZebOS-XP CLI. From the IMISH Line, IMI leverages the connections between IMI and each of the protocols, and the generation of a complete CLI list. From the ZebOS-XP CLI, IMI leverages the connection between the IMI and the user.

1. User starts IMI first, then NSM and other protocol daemons.
2. The NSM and the protocol modules connect to the IMI using Unix domain sockets.
3. User opens a TELNET session to the IMI port. For some applications, the IMI may run at the default TELNET port (23) so that users are automatically directed to the IMI. In particular, this might be useful for home gateways, where logging into the operating system might be prohibited.

4. User executes commands; IMI validates syntax, executes system application commands locally or calls NSM or protocol CLI over Unix domain socket connection.
5. When user exits CLI, the TELNET session is closed, but the daemon continues to run.

NSM and IMI Interaction

IMI Command List

The IMI uses the IMI Line Perl script to generate a list of all of the CLI commands that are valid for the NSM and routing protocols.

System Startup

Start the IMI first, followed by the NSM and protocols.

IMI -> NSM Configuration

The IMI runs as a Server of the NSM and other protocols and uses Unix domain sockets for propagating configuration commands from the IMI to the NSM.

NSM -> IMI Data and Events

The IMI maintains certain information about the current state of the system. This includes interface up/down, IP addresses, and so on.

To receive and process this data, the IMI runs as a normal client of the NSM, similar to the routing protocols. This allows the NSM to dynamically update the IMI whenever system changes occur.

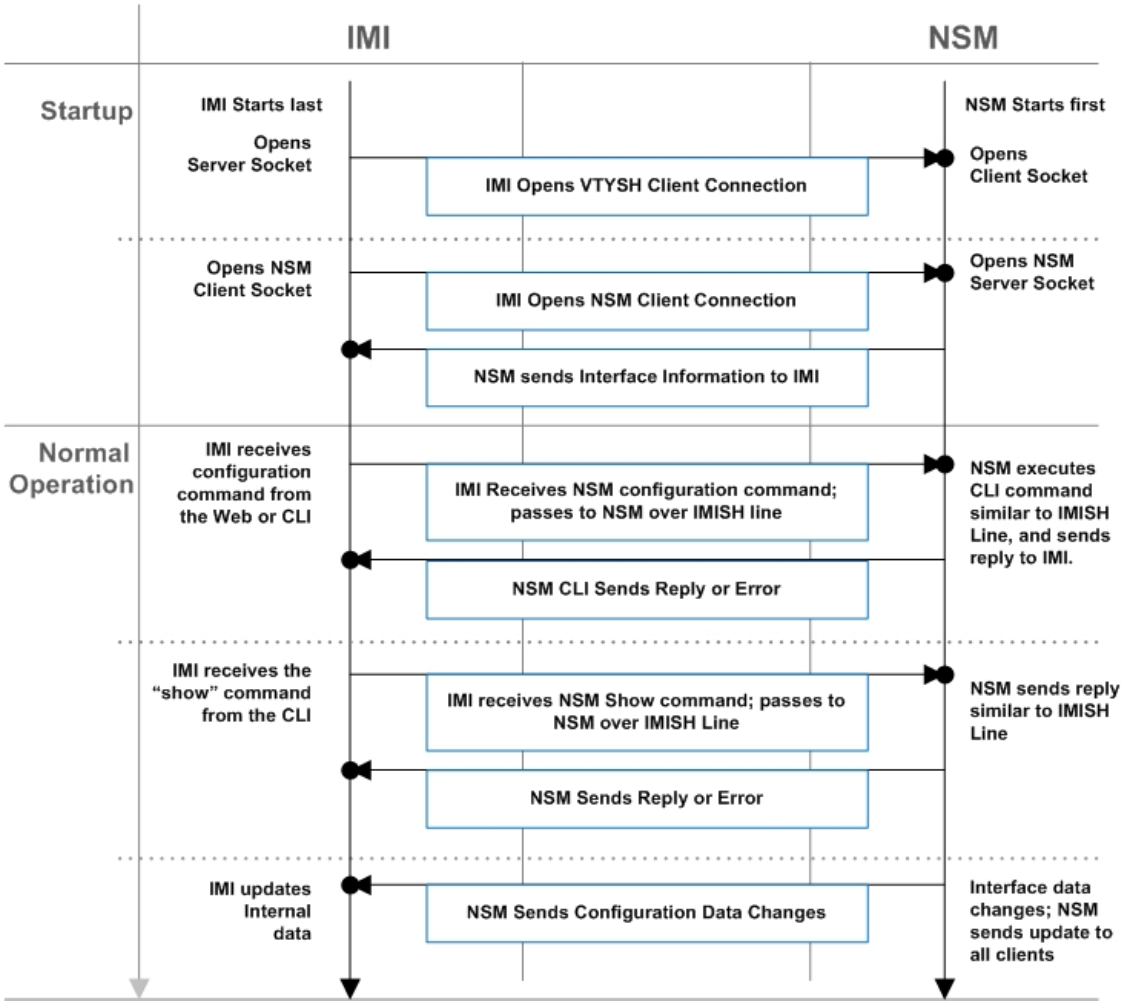


Figure 2-2: Data Flow Between IMI and NSM

CHAPTER 3 IMI Shell

The IMI Shell (IMISH) resides in its own layer on top of IMI and facilitates user access to both the IMI CLI and operating system commands including:

- telnet
- shell
- ping
- traceroute

A separate instance of the IMISH is instantiated for each user of the system. The order of operations is as follows:

1. User connects to the device.
2. IMISH is executed.
3. User authentication is performed using IMISH or system.
4. IMISH establishes a connection with ZebOS-XP through the IMI.

System Commands

To handle system commands such as ping, the IMISH interacts directly with the operating system.

ZebOS-XP Commands

The IMISH uses a Perl script to generate a global command list/tree for the CLI mechanism.

Communication Methods

There are two communication methods for the user interface: `Line` and `Show`. The `Line` method manages authentication and configuration. The `Show` method manages all the show commands.

The `Line` connection is established during the startup of IMISH and remains running all the time.

Every time a request is received, the `Show` connection is established, command is executed, then the `Show` connection is torn down.

Implementation

The `Line` and `Show` methods are provided in the ZebOS-XP library (/lib).

`lib/line.h`

`lib/line.c`

`lib/show.h`

`lib/show.c`

Line Method

The `Line` method manages authentication and configuration. When the IMISH starts, it initiates a `Line` connection to IMI. IMI checks whether the connection is acceptable or not.

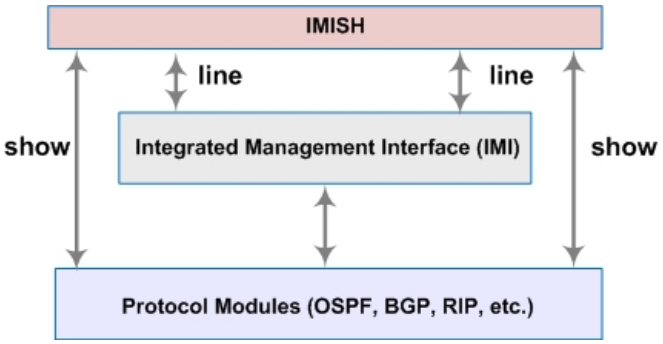


Figure 3-1: Line Method

Show Method

The `Show` method manages the display of information to users. The `Show` connection is established each time a user requests for information. The IMISH connects to the necessary process directly (not using IMI). When the command needs information from IMI, it initiates the `Show` connection against IMI. When the command needs information from a protocol module, it directly connects to the protocol module.

The IMISH continues to display information until all information has been displayed.When IMI or the protocol module has a large output, IMI or the protocol module can register a call-back function to the `Show` connection that allows IMI or the protocol module to limit the amount of information displayed at one time. When the user displays the first part of the information, a call-back function is invoked, and the IMI or the protocol module writes the next part of the output to the `Show` connection.

When the user terminates the output during display, the `Show` connection is closed. IMI or the protocol module can detect the socket close and completes the appropriate post processes on the connection. If there is a call-back function the `Show` method unlocks the next information pointer.

Special Show Commands

There are some show commands, for example, `show interface`, `show running-config` and `show memory`, that require information from different protocol modules at the same time. For such commands, IMISH initiates the `Show` connection to IMI and IMI collects information from different protocol modules.

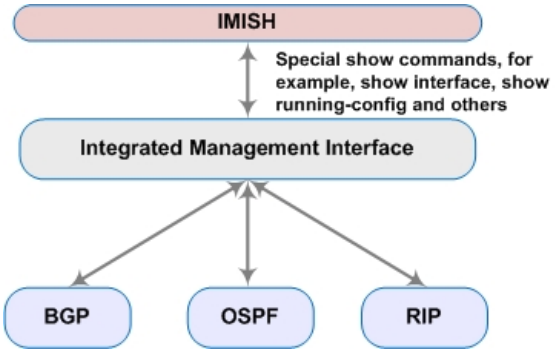


Figure 3-2: Special Show Commands

Special Commands

Some commands such as `enable`, `end`, `exit` and `configure terminal`, must be executed in IMISH and in other ZebOS-XP daemons. These commands are installed using the `cli_install_imi()` function. Different daemons control different parts of some commands, for example:

interface IFNAME

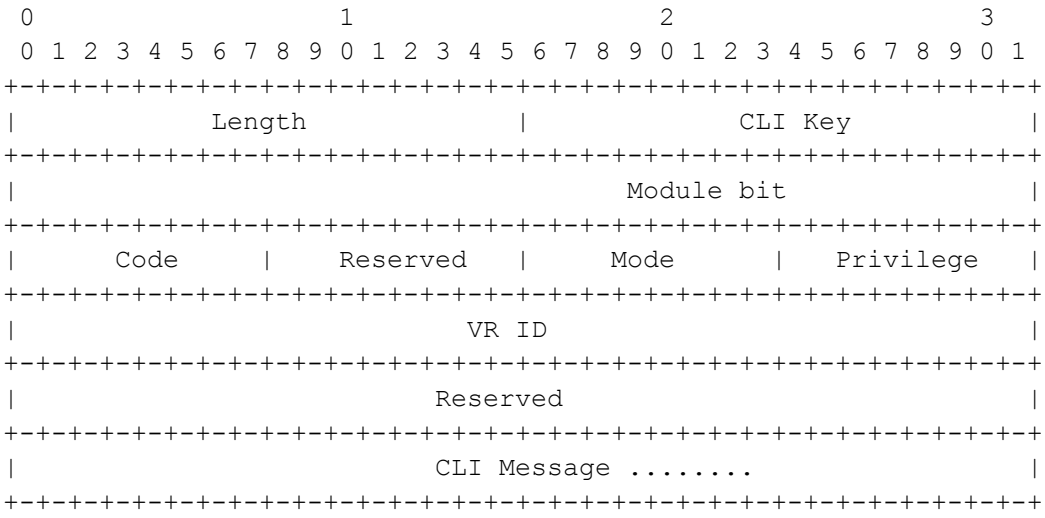
- IMISH moves user to `Interface` mode.
- IMI co-ordinates command from IMISH to protocol modules.
- NSM creates the interface.

enable

- IMISH moves user to the `Exec` mode.
- IMI coordinates the lock or unlock.

Message Format

For passing commands between daemons, the following message format is used:



CHAPTER 4 VLOG Module

You can debug Virtual Routers (VRs) with the ZebOS-XP VLOG feature.

Features

- A non-Privileged Virtual Router (PVR) user may enable debugging on a VR where the user is logged in
- A non-PVR user can view debug information for the VR where the user is logged in
- A PVR user may enable debugging in the PVR context, and enable other global debugging that is not VR-specific
- A PVR user is able to view all debugging, including debugging information generated in the context of non-privileged VRs
- VLOG excludes log throttling (duplicate debug messages are not handled)
- VLOG excludes user permissions for Log file

Debug Support for Protocol Modules

- A VLOG build allows operators to enable debugging in a specific VR context for BGP, OSPFv2, OSPFv3, IS-IS, RIP and RIPng.
- The commands entered in global VR configure mode allow a VR user to configure a system for a specific type of debugging.
- VR-specific debug output can be written to the terminals where the VR user is logged in.
- A PVR user may view all debug output, including that generated in the context of specific virtual routers.
- Debugging for protocol modules OSPFv2, ISIS, OSPFv3, RIPng, RIP and BGP is VR context-sensitive.

VR Builds

The VLOG daemon (`VLOGD`) supports the following build options and operational modes:

Non-VR Build

A non-VR build uses the debugging and logging functionality in ZebOS-XP.

VR Build With VLOG Disabled

A VR build with VR disabled also uses the debugging and logging functionality in ZebOS-XP.

VR Build with VLOG Enabled

To generate this build, two build options must be enabled: `--enable-vr` and `--enable-vlogd`.

Note: For details about ZebOS-XP build options, see the *Installation Guide*.

VLOG Users

There are two types of VLOG users, VR users and PVR users.

VR users may:

- Enable or disable VR debugging
- View the debug messages of their own VR
- Log the debug messages to a specified log file
- Specify a VR log file name; otherwise a default name is used.

PVR users may:

- Enable or disable PVR and VR debugging
- From a PVR terminal session, view both the PVR and any VR debug output
- Log the debug output for both the PVR and all VRs to a log file
- Specify a local or global log file name, otherwise a default name is used

Non-VR Operation

ZebOS-XP writes debug messages to `syslogd` using the `syslog` system call. When you give the `terminal monitor` command, IMI writes the terminal device name to the `/etc/syslog.conf` file, instructing `syslog` where to write the debug output. The `syslog` function can forward debug output to more than one user terminal.

When processing a ZebOS-XP log message, the `syslog` function inserts the date and time at the beginning of the message and forwards it to `syslogd`. The `syslogd` forwards the messages to all enabled terminal monitors, and writes it to the syslog file. In this case, the `syslogd` is not differentiating the per VR messages; it just displays the debug messages to the terminals. The following figure depicts the implementation.

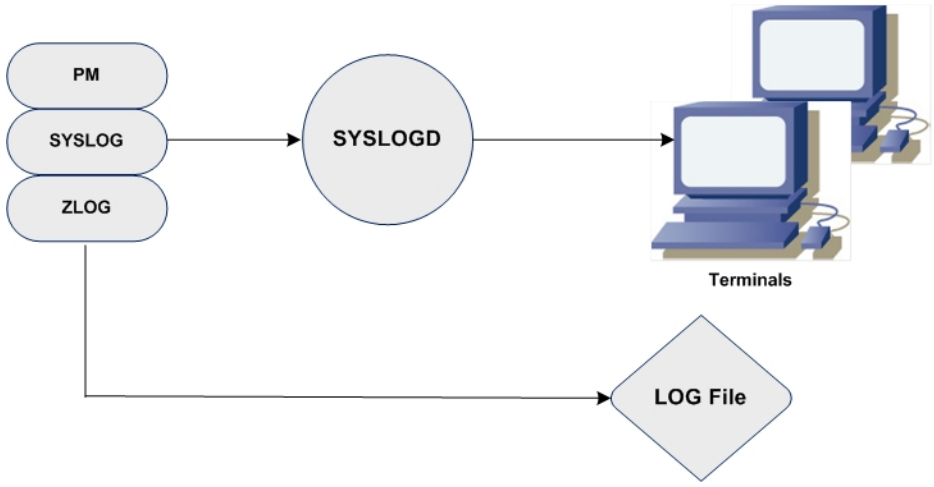


Figure 4-1: SYSLOG Diagram

Architecture

This section describes the architecture of the VLOG system.

External interfaces

There are three external interfaces relevant to the implementation of the VR-context-sensitive logging:

1. User interface — IMI

2. `syslogd` — Syslog daemon
3. File system

User Interface

A PVR user is able to view debug output for any VR, a subset of VRs, or all of them. The system provides the user with the capability to make these selections.

SYSLOG Daemon

A `syslog` is enabled by default during system startup. A PVR user can disable or enable it again. When enabled, for the PVR user's convenience, all debug output (including all of its VRs) is directed to the `syslog` message file. To enable writing to the syslog file the PVR user issues the "log `syslog`" command. Non-PVR users are not allowed to control writing to the syslog file.

Note: Debug output is written to the `syslog` directly from the protocol daemons; it is not routed via the VLOGD.

File System

A PVR or VR user has the capability to name a log file as a destination for debug output. Log files for VR are stored in the `/var/local/zebos/log/<vr-name>` subdirectory. The size of the log file is limited to 10MB. When the maximum size is reached, a new log file is opened and debug output is directed to the new file. Log files are versioned, and once a log file is filled up, it is closed and a version number is added to the file name.

Software Components

The system-level design for VR context-specific debugging consists of the following software units.

VLOG Daemon

A VLOG daemon is implemented to manage the process for forwarding log messages to different terminals and log files in a VLOG-enabled build. The VLOG daemon:

- implements the VLOG server for client-server communication with protocol modules
- uses the IMI client to communicate with the IMI daemon
- uses the IMISH session terminal devices to forward debug output to the locations the user has requested
- controls and writes to the PVR-VR log files

IMISH

In VLOG builds, IMISH provides the user interfaces while at the same time becomes a terminal device where the debug context is directed. In VLOG builds, IMISH:

- is used to enter CLI commands and display their output
- is used to display the debug output
- uses the IMI client to connect to the IMI server.

IMI

IMI provides access to the user interface, and:

- implements the server to allow IMI clients (PMs, VLOGD or IMISH) to connect
- forwards user CLI commands to `VLOGD`
- provides operational data to `VLOGD` using internal CLI commands.

Protocol Module

A Protocol Module (PM) refers to any protocol daemon that forwards debug output to the VLOG daemon. A PM uses the VLOG client library module to communicate with the VLOG server.

System Domain

The diagram below depicts VLOG in the context of the ZebOS-XP system:

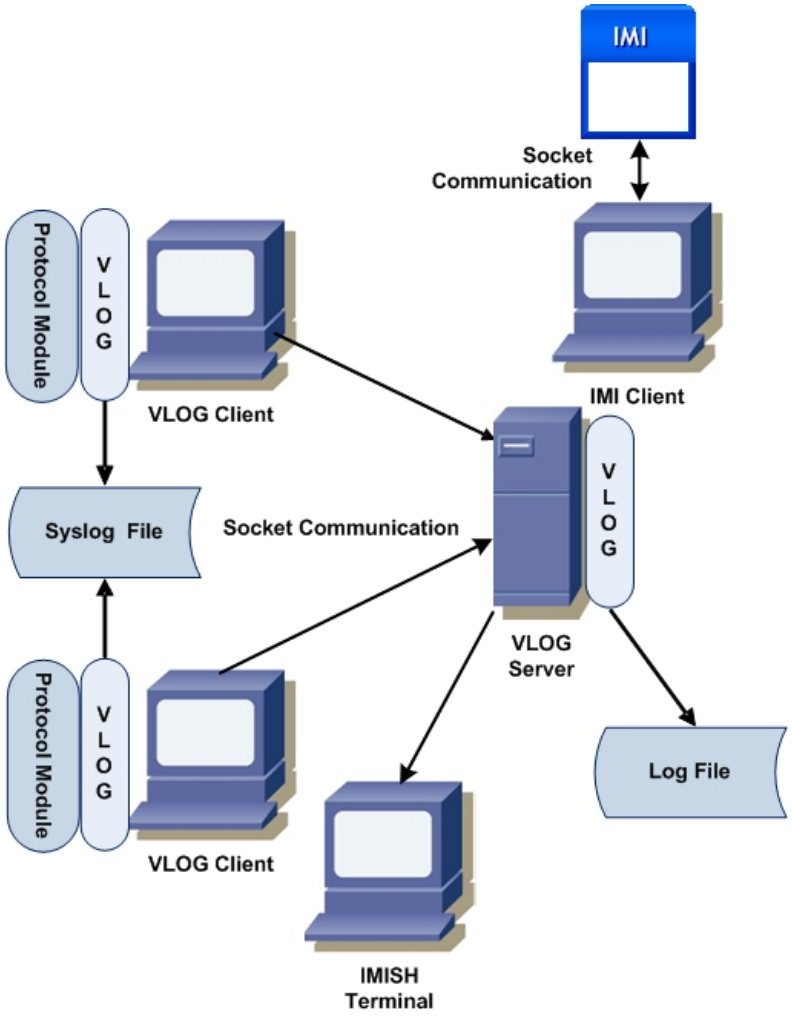


Figure 4-2: VLOG System Domain Diagram

Display

A new message type is defined for sending log messages from a PM to `VLOGD`. It contains the VR ID, priority of log message, protocol module ID, message length and the textual message itself.

Based on the VR ID, `VLOGD` can enter the proper VR context and forward messages to all attached VR terminals and the log file.

All messages displayed on a PVR user terminal or written to the PVR log file have a VR name placed as the first item in the row. The VR name does not appear in messages displayed on a VR user terminal or sent to the VR log file.

Log Files

The PVR and each VR may have a separate log file. The PVR log file receives all debug output that is generated in the system by any VR. The following directories have been established as default locations for log files:

1. For PVR log files:

   /var/local/zebos/log/pvr

2. For VR log files

   /var/local/zebos/log/<vr-name>

In order to request `VLOGD` to write messages to log files, the user must execute the `log file` command.

The size of log files is limited to LOG_FILE_SIZE (10 MB). If the size is exceeded, a new log file is created with the original filename and appended with sequence number. For example, if the filename is `log_name`, after exceeding the maximum file size, a new log file is created and named `log_name1`.

IMISH Builds

In an IMISH build, a user connects to IMI via IMISH. When required, the IMISH/IMI validates user credentials. The user may start a debugging session by issuing the `terminal monitor` command.

IMI retrieves the current IMISH TTY device name and the identity of the VR that needs to be monitored, and forwards the information to `VLOGD`. The `VLOGD` opens the terminal device, binds the terminal to the monitored VR log output, and starts forwarding log messages to the user terminal.

A PVR user may issue several `terminal monitor` commands, each requesting binding to another VR. A PVR user may also request binding to all VRs at once. When using the latter option, any new VR that is added to the configuration during the debug session is automatically bound to the PVR user terminal.

Protocol Module Connectivity

When a PM is disconnected from the VLOG daemon, it tries to reconnect using the VLOG reconnect callback for every reconnect_interval (5 seconds). In a case where the vlog_client is unable to connect to the vlog_server, debug messages are discarded.

IMI Restart

When IMI goes down, VLOG detects the IMI client-server connection failure and starts periodic attempts to reconnect. Once IMI comes back up, the `VLOGD` downloads and overwrites existing operational data with the current configuration.

Note: An assumption is made that after IMI comes back up, the system configuration, in particular the VR configuration, is the same as it was before IMI went down.

With respect to terminals with enabled debug output, when IMI goes down, the `VLOGD` starts a shutdown timer and continues to log activities of all currently attached terminals without losing any important debug information. In order to terminate debug output, the user needs to terminate the OS terminal session. A terminal session is also terminated when the shutdown timer expires.

After IMI restarts, the user can reconnect to the previous session and continue it by starting IMISH and issuing the `terminal monitor` command again.

VLOG Restart

If the VLOG module reboots or disconnects, it flushes all of its VR information. Upon restart, it connects to IMI and downloads the main configuration file. Then the IMI downloads to `VLOGD` the VR-name-to-ID mappings. IMI issues a series of internal CLI commands, vr_instance <vr-name> <vr-id>, one per VR, in response to which the `VLOGD` requests a download of the VR configuration files.

Protocol Module VR Debug Support

PMs have to initiate a connection to the VLOG server using vlog_client. The communication between a PM and the `VLOGD` module is socket-based.

PMs call the function `vlog_client_create` during instantiation of the local to the PM log object. Once it is connected, a call is made to `vlog_client_start` to initiate VLOG services. To disconnect VLOG services during termination of a protocol module, a call is made to `vlog_client_stop`.

ZebOS-XP maintains debug flags per-VR for NSM, OSPFv2, ISIS, OSPFv3, RIPng and RIP. Debugging must be based on VR context.

Data Flow

A protocol module sends debug messages to vlog_client, which converts it to a vlog_msg and sends it to the VLOG module. VLOG identifies the VR and forwards the debug message with prepended date and time data to the attached VR terminal or the VR log file.

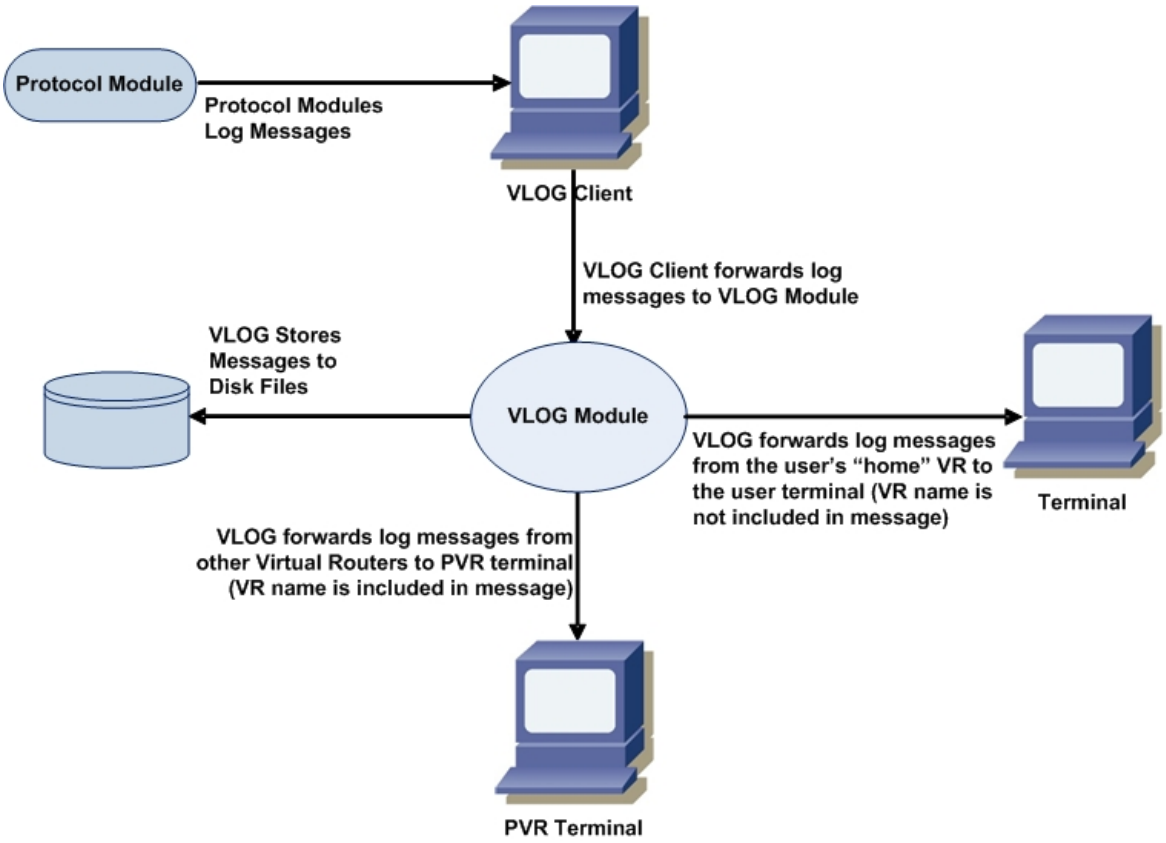


Figure 4-3: VLOG Data Flow

Static Model

The following diagram is a static model of the VLOG objects. Definitions of each object follow the diagram.

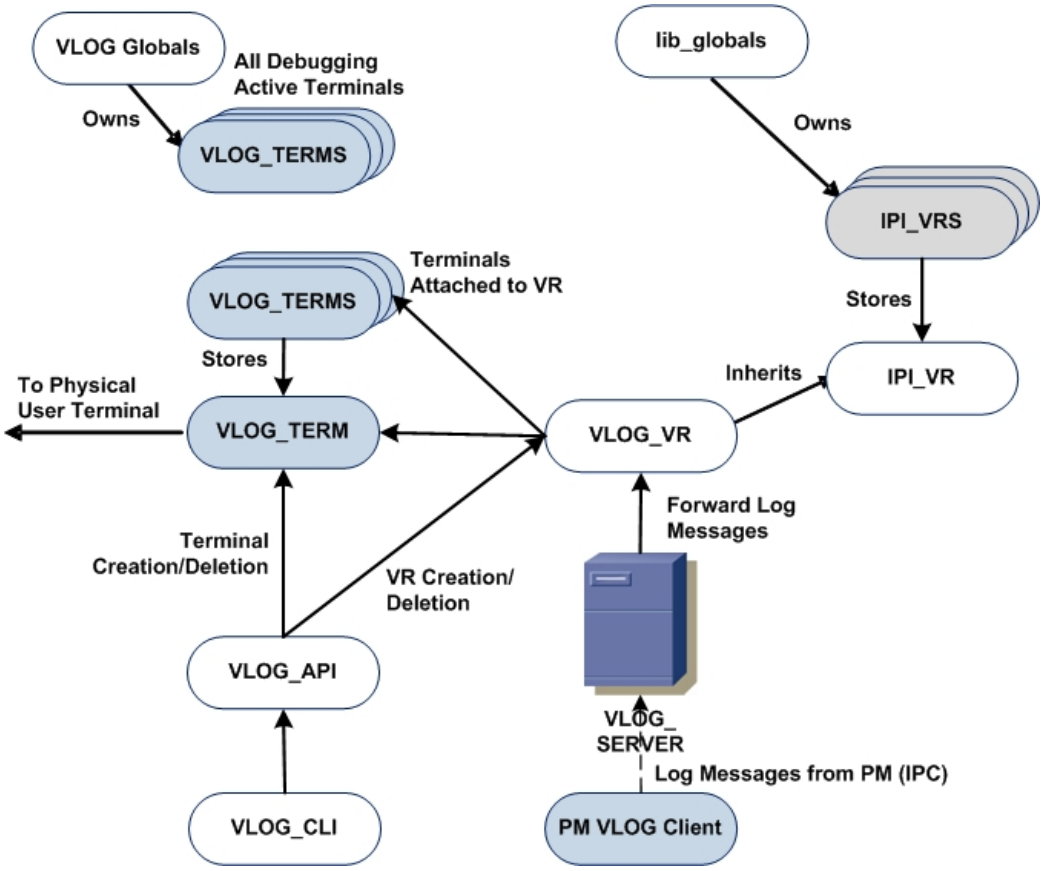


Figure 4-4: VLOG Static Model

Object Definitions

The following objects are defined for VLOG support.

vlog_api

This is a VLOG interface/helper class that implements the VLOG CLI API. It forwards CLI requests to appropriate internal objects for execution.

vlog_client

VLOG client is an object that allows protocol modules to forward log messages to VLOG server.

The `vlog_client` object has these responsibilities:

- Provides a command-line communication interface between IMI and `VLOGD`
- Contains CLI handlers for, and extensions to the `VLOGD` module, for the `terminal monitor` command
- Extends to VLOG VR creation and deletion events
- Implements the client-side of the VLOG client-server protocol
- Establishes the client-server communication link

- Forwards log messages to VLOG server
- Monitors connection and reconnects in case of failure

vlog_server

The `vlog_server` receives connection requests from VLOG clients, maintains the connections, and distributes log messages received via these connections to the `vlog_vr` object. Distribution is based upon the VR ID contained in each message.

vlog_msg

The `vlog_msg` message contains the following elements:

- ID of the VR in context
- Message type
- VLOG client module identification
- Priority
- Data length
- Log array of 256 characters

vlog_term

A `vlog_term` represents a VLOG user terminal, and it is responsible for writing debug output to the physical terminal.

The `vlog_term` strips from the log message the VR name field, if debug output is destined for a VR home terminal. All `vlog_term` objects are stored in a `vlog_terms` container that is attached to the `vlog_glob` object.

vlog_vr

The `vlog_vr` object represents either a Privileged VR or a non-privileged VR. A `vlog_vr` object is attached to the `ipi_vr` structure. It is responsible for forwarding log output to the attached `vlog_term` object. Each `vlog_term` has its own container of attached `vlog_term` objects and a log file.

lib_globals

This object provides a location where a global VR context is maintained. The context must be set by a protocol module whenever processing of an event for a VR is started. This object also stores the log file `set`, `get` and `unset` callback functions.

Log file names are stored in the application (IMI and VLOGD), in the VR context, while the log-file-related CLI command handlers are still located in the lib/log.c file. A set of callbacks, `set`, `get` and `unset`, is used to communicate log files names between the log.c library and upper application layers.

Two modules utilize the log file names: IMI for the `show running config` command, and `VLOGD` to manipulate the log files.

log

The `log` object receives log messages from application codes, preprocesses them and then forwards them to the VLOG server or `syslogd`. It also stores log files names in an application and retrieves them to include in the `running config` content.

The following changes were made in `log.c`:

- `openzlog` starts the `vlog_client`
- `closezlog` closes the `vlog_client`
- `vzlog()` forwards messages to `vlog_client`

The `log.c` file still contains the `log file` CLI handlers. However, per-VR log file names are stored in the application. The `lib.c` file contains the APIs to set, unset and retrieve log file names.

thread

The thread simulates a multithreading environment. It sets the PVR default context at the time of dispatching a thread event.

pal_vlog_file

This object is a representation of a single log file. It is instantiated by `VLOGD` and shared between `VLOGD` main code and the PAL. It also:

- opens and closes the log file
- writes log messages to the log file
- detects when the log file becomes full, and rotates the log files by deleting the oldest, assigning a version number to the file that is being closed, and opening a new, empty file
- stores log files names applications and retrieves them to include in `running config` content

CHAPTER 5 Tunneling and Transitioning

Typically, tunneling is used to transmit private data over a public network, such as the Internet. Tunnels enable carrying of incompatible data over an existing network. For example, IPv6 data can be transmitted over IPv4 networks. Secure tunneling protocols (such as IPSec) can be used for transferring sensitive data over public networks.

Tunneling

Tunneling is achieved by encapsulating IP packets of private networks within IP packets of public networks. This allows packets destined for one IP address to be wrapped and redirected to another IP address. To encapsulate an IP packet, an outer IP header is inserted before the packet's existing header. The source and destination addresses in the inner IP header, specify the original sender and recipient of the packet.

The ZebOS-XP IPv4 Tunneling implementation supports Generic Routing Encapsulation (GRE) and IP-in-IP (IP-IP) Tunneling.

Transitioning

IPv6 transition is required for migrating from IPv4 to IPv6.

IPv6 Transition - 6to4

One method to connect to the global IPv6 network over an existing IPv4 network is called 6to4 automatic tunneling. With 6to4, you don't have to specify destination address of the tunnel endpoint, instead, destination IPv6 address itself contains destination IPv4 address to be used for the tunnel encapsulation. ZebOS-XP can create 6to4 global IPv6 address automatically which is delivered from the configured tunnel source address.

IPv6 Transition - ISATAP

The ISATAP, or Intra-Site Automatic Tunnel Address Protocol, is used for a site which does not have a fully-native IPv6 network. Each ISATAP router and ISATAP host can talk to each other through the ISATAP automatic tunnel over the existing IPv4 network. One big difference from a 6to4 tunnel is that an ISATAP tunnel interface treats the underlying IPv4 as an NBMA (Non-Broadcast Multi-Access) network so each ISATAP interface has same IPv6 prefix. Router Advertisement (RA) functionality can be used to assign the prefix automatically.

IPv6 Transition - GRE Tunnel

Configuring a GRE (Generic Routing Encapsulation) Tunnel for IPv6 transition is similar to configuring a Configured Tunnel.To configure a GRE Tunnel, a route is statically configured between two routers to enable forwarding of IPv6 packets over an IPv4 network.

Index

B

Basic Access 5

C

CLI API 8
  configuring APS 8
communication methods 11
configuration interfaces 8

E

enable command 13
end command 13
exit command 13

G

Generic Routing Encapsulation 25
generic routing encapsulation 25
GRE 25

I

IMI 5
IMI and central control 7
IMI and NSM configuration 9
IMI and NSM data flow 10
IMI architecture 8
IMI CLI 9
IMI master copy of common data 7
IMI persistent daemon 7
IMI-NSM interaction 9
IMISH 5
interface command 13
IP-IP tunneling 25
ipip tunneling 25
IPv6 transition 6to4 25
IPv6 transition GRE tunnel 25
IPv6 transition ISATAP 25

L

line and show command implementation 11
line method 12

M

Management Interface Modules 5
message format 13
multiple types of access 7

N

NSM and IMI data events 10
NSM IMI data flow 10
NSM interaction 9
NSM IPC 8
NSM-IMI interaction 9

P

ping 11

S

shell 11
show method 12
special commands 13
system commands 11
system startup 9

T

telnet 11
traceroute 11
transitioning 25
tunneling 25
tunneling and transitioning 5, 25

W

write file command 7
write memory command 7

Z

ZebOS IMI shell 11
ZebOS VLOG 15
  Architecture 16
  Data Flow 21
  Debug Support for Protocol Modules 15
  Display 19
  Features 15
  File System 17
  IMI Restart 20
  IMISH Builds 20
  Log Files 20
  Objects 22
  PM VR Debug Support 21
  Protocol Module Connectivity 20
  Restart 21
  Software Components 17
  Static Model 22

SYSLOG Daemon 17
System Domain 19
User Interfaced 17
Users 15
VR Builds 15
ZebOS VLOG Objects
lib_globals 23
log 23
pal_vlog_file 24
thread 24
vlog_api 22
vlog_client 22
vlog_msg 23
vlog_server 23
vlog_term 23
vlog_vr 23