SNMP Configuration and Management Manual

Abstract

This manual describes how to install, start, configure, and stop the HP Simple Network Management Protocol (SNMP) Agent and subagents. The SCF commands used by SNMP are described. This manual also discusses the objects in the Management Information Bases (MIBs) used by the agent and subagents. The SNMP Agent and its subagents comply with SNMP Requests for Comment (RFCs) so that SNMP-compliant management applications, known as SNMP managers, can manage various resources on an HP NonStop™ host.

Product Version

SNMP Agent (T9576G06), T9576H01)

Supported Release Version Updates (RVUs)

This manual supports the G06.24 RVU and all subsequent G-series RVUs and the H06.03 RVU and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

Part Number	Published
424777-006	July 2005

Document History

Part Number	Product Version	Published
424777-002	SNMP Agent (T9576G06)	May 2002
424777-003	SNMP Agent (T9576G06)	November 2002
424777-004	SNMP Agent (T9576G06)	December 2003
424777-005	SNMP Agent (T9576G06)	September 2004
424777-006	SNMP Agent (T9576G06, T9576H01)	July 2004

Subagent Product Versions

IPX/SPX Subagent (T8170D30)
TCP/IP Subagent (T7862G05)
Ethernet/Token Ring Subagent (T0326G06)
Trap Multiplexer (T1041G06)
NonStop NET/MASTER Trap Subagent (T8491D20)
EMS Trap Subagent (T9632D31)
Host Resources Subagent (T8496G04)
ONS Subagent (T8103D40)

SNMP Configuration and Management Manual

Glossary | Index | Examples | Figures | Tables

Part II. SCF for the SNMP Agent

4. Introduction to SCF for the SNMP Agent

4. Introduction to SCF for the SNMP Agent (continued)

5. SCF Commands for the SNMP Agent

8. TCP/IP Subagent (continued)

11. Host Resources Subagent

11. Host Resources Subagent (continued)

11. Host Resources Subagent (continued)

Figures (continued)

Tables

Tables (continued)

Tables (continued)

Tables (continued)

What’s New in This Manual

Manual Information

Abstract

This manual describes how to install, start, configure, and stop the HP Simple Network Management Protocol (SNMP) Agent and subagents. The SCF commands used by SNMP are described. This manual also discusses the objects in the Management Information Bases (MIBs) used by the agent and subagents. The SNMP Agent and its subagents comply with SNMP Requests for Comment (RFCs) so that SNMP-compliant management applications, known as SNMP managers, can manage various resources on an HP NonStop™ host.

Product Version

SNMP Agent (T9576G06), T9576H01)

Supported Release Version Updates (RVUs)

This manual supports the G06.24 RVU and all subsequent G-series RVUs and the H06.03 RVU and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

Part Number	Published
424777-006	July 2005

Document History

Part Number	Product Version	Published
424777-002	SNMP Agent (T9576G06)	May 2002
424777-003	SNMP Agent (T9576G06)	November 2002
424777-004	SNMP Agent (T9576G06)	December 2003
424777-005	SNMP Agent (T9576G06)	September 2004
424777-006	SNMP Agent (T9576G06, T9576H01)	July 2004

Subagent Product Versions

IPX/SPX Subagent (T8170D30)
TCP/IP Subagent (T7862G05)
Ethernet/Token Ring Subagent (T0326G06)
Trap Multiplexer (T1041G06)
NonStop NET/MASTER Trap Subagent (T8491D20)
EMS Trap Subagent (T9632D31)
Host Resources Subagent (T8496G04)
ONS Subagent (T8103D40)

New and Changed Information

This revision contains these changes and additions:

Section	Change
Section 1, The NonStop SNMP Environmentt	Notes concerning H-Series support have been included.
Section 2, Installing and Configuring the SNMP Agent	Notes concerning H-Series support have been included.
Section 4, Introduction to SCF for the SNMP Agent	H-series information has been added.
Section 7, Troubleshooting the SNMP Agent	H-series information has been added.
Section 8, TCP/IP Subagent	H-series information has been added.
Section 10.ONS Trap Subagent (D-series)	This section has been deleted.
Section 11.NonStop NET/MASTER Trap Subagent	Has become section Section 10, NonStop NET/MASTER Trap Subagent.
Section 12. Host Resources Subagent	Has become Section 11, Host Resources Subagent.
Section 13. Trap Multiplexer Subagent/Manager	Has become section Section 12, Trap Multiplexer Subagent/Manager.
Section 14. IPX/SPX Subagent	Has become section 13 with the title Section 13, IPX/SPX Subagent (G-Series).
Section 15. Ethernet Subagent	Has become Section 14, Ethernet Subagent.

About This Manual

The SNMP agent and subagents let customers manage HP NonStop systems by using SNMP managers. SNMP managers are SNMP-compliant applications that manage multiplatform networks.

This manual describes how to install, configure, and interpret information generated by the SNMP agent and its subagents. It also defines SCF commands for SNMP.

Your Comments Invited

After using this manual, please take a moment to send us your comments. You can do this by:

- Completing the online Contact NonStop Publications form if you have Internet access.
- Faxing or mailing the form, which is included as a separate file in Total Information Manager (TIM) collections and located at the back of printed manuals. Our fax number and mailing address are included on the form.
- Sending an e-mail message to the address included on the form. We will immediately acknowledge receipt of your message and send you a detailed response as soon as possible. Be sure to include your name, company name, address, and phone number in your message. If your comments are specific to a particular manual, also include the part number and title of the manual.

Many of the improvements you see in manuals are a result of suggestions from our customers. Please take this opportunity to help us improve future manuals.

Audience

This manual has two audiences: NonStop system personnel and users of the SNMP manager.

- NonStop system personnel include system managers, network managers, and operators who handle such host-based operations as SNMP agent and subagent installation, configuration, and troubleshooting. These personnel need a working familiarity with the HP NonStop Kernel operating system as well as such subsystems as TCP/IP, Subsystem Control Facility (SCF), and Event Management Service (EMS). These personnel are acquainted with basic SNMP principles.
- Users of the SNMP manager configure and use SNMP managers to send requests to and receive responses from the SNMP agent and subagents. Users of the SNMP manager also monitor traps forwarded by the SNMP agent. These users are familiar with their management applications and with the platforms those applications run on. SNMP manager users know the format of requests and responses exchanged with SNMP agents on systems provided by different vendors.

Purpose

For NonStop system personnel, this manual describes how to install and configure the SNMP agent and its subagents. It also documents the messages generated and explains how the SNMP agent and subagents fit into the overall NonStop subsystem architecture.

For users of the SNMP manager, this manual describes the MIBs supported in the NonStop SNMP environment. In addition, it documents how the SNMP agent and subagents comply with guidelines in RFCs and describes the traps generated.

Organization

This manual has five parts. Part I, Installing and Configuring SNMP, is an overview of SNMP and tells how to configure, start, stop, and manage an SNMP agent. .Part II, SCF for the SNMP Agent, tells how to manage SNMP using the Subsystem Control Facility (SCF). Part III, Troubleshooting describes diagnostics. Part IV, SNMP Subagents describes the SNMP subagents. Part V, Appendices, provides reference information. Following is a brief description of each section:

Part I, Installing and Configuring SNMP

- Section 1, The NonStop SNMP Environment, provides a high-level view of SNMP and how it is implemented in the NonStop environment.
- Section 2, Installing and Configuring the SNMP Agent, describes how to configure the SNMP agent using SCF, the RUN command, PARAM statements, and the SNMPCTL file. This section describes how to configure security, response/request connection, and trap- connection agent elements.
- Section 3, MIBs Supported by the SNMP Agent, describes the traps and EMS event messages that the SNMP agent generates.

Part II, SCF for the SNMP Agent

- Section 4, Introduction to SCF for the SNMP Agent, provides an overview of the subsystem control facility (SCF) and how it can be used to manage SNMP.
- Section 5, SCF Commands for the SNMP Agent, describes how the components of a NonStop agent configuration are organized and managed through the SCF interface

Part III, Troubleshooting

- Section 6, SNMP Agent PTrace Facility. describes the PTrace utility, which tracks and maintains records of communications between processes.
- Section 7, Troubleshooting the SNMP Agent, contains guidelines about what to do when you encounter problems starting the SNMP agent.

Part IV, SNMP Subagents

- Section 8, TCP/IP Subagent, describes the subagent that facilitates management of TCP/IP processes on SNMP systems.
- Section 9, EMS Trap Subagent, describes the subagent that translates EMS events into SNMP traps.
- Section 11, Host Resources Subagent, describes the subagent that translates Event Management Service (EMS) events routed to NonStop NET/MASTER MS into SNMP traps whose objects are defined in the EMS Trap MIB.
- Section 12, Trap Multiplexer Subagent/Manager describes the subagent that supports the management of hardware and software resources on the NonStop system where the subagent is installed.
- Section 13, IPX/SPX Subagent (G-Series), describes the product that, as a manager, receives traps from network devices and converts them into EMS events and, as a subagent, provides manager station access to an MIB that describes trap conversion activities and configuration attributes.
- Section 14, Ethernet Subagent, describes the Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) Subagent, which lets you monitor and manage NonStop IPX/SPX networks.

Part V, Appendices

- Appendix A, SCF Command Syntax Summary for the NonStop Agent, lists the syntax for the SCF commands supported by the NonStop agent.
- Appendix B, SCF Error Messages for the NonStop Agent, describes the SCF error messages specific to the NonStop SNMP agent.
- Appendix C, Unsolicited SNMP Agent Messages, describes the EMS event messages generated by the SNMP agent.

A Glossary provides definitions of SNMP terms and acronyms.

Related Reading

This manual has three companion manuals:

- The *SNMP Manager Programmer's Guide*explains how to use the Manager Services Toolkit product to generate SNMP managers that run as NonStop Kernel processes in Guardian or HP NonStop Kernel Open System Services (OSS) environments.
- The *TCP/IP (Parallel Library) Configuration and Management Manual* describes how to configure and manage the Parallel Library TCP/IP subsystem.
- The *TCP/IPv6 Configuration and Management Manual* describes how to configure and manage the NonStop TCP/IPv6 subsystem.
- The *TCP/IP (Parallel Library) Migration Guide* provides information about migrating your NonSTop TCP/IP applications to the Parallel Library TCP/IP environment.

Throughout this manual, you will find references to RFCs. RFCs are documents that describe the Internet suite of protocols and related experiments. The Internet Architecture Board requires that protocols be documented in RFCs so that standards and ideas are widely accessible.

You can also obtain information about SNMP at libraries and book stores. Many excellent and up-to-date books are available.

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

This requirement is described under SNMP Agent Process on page 2-16.

General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

file-name

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *`Italic computer type`* letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

`pathname`

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num ]
   [ -num]
   [ text]
```

```
K [ X | D ] address-1
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list may be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address-1 [ , new-value ]...
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE

   [ , attribute-spec ]...
```

!i and !o. In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT (  segment-id                    !i
                         , error        ) ;              !o
```

!i,o. In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ;                      !i,o
```

!i:i. In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ (  filename1:length           !i:i
                            , filename2:length ) ;       !i:i
```

!o:i. In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ (  filenum                          !i
                        , [ filename:maxlen ] ) ;          !o:i
```

Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text. Bold text in an example indicates user input entered at the terminal. For example:

```
ENTER RUN CODE

?123

CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

```
p-register

process-name
```

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list might be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
P=%p-register E=%e-register
```

Notation for Management Programming Interfaces

The following list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS. Uppercase letters indicate names from definition files; enter these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

lowercase letters. Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

!r. The !r notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME      token-type ZSPI-TYP-STRING.          !r
```

!o. The !o notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER      token-type ZSPI-TYP-FNAME32.         !o
```

Change Bar Notation

Change bars are used to indicate substantive differences between this edition of the manual and the preceding edition. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

> The message types specified in the REPORT clause are different in the COBOL85 environment and the Common Run-Time Environment (CRE).
>
> The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

Part I. Installing and Configuring SNMP

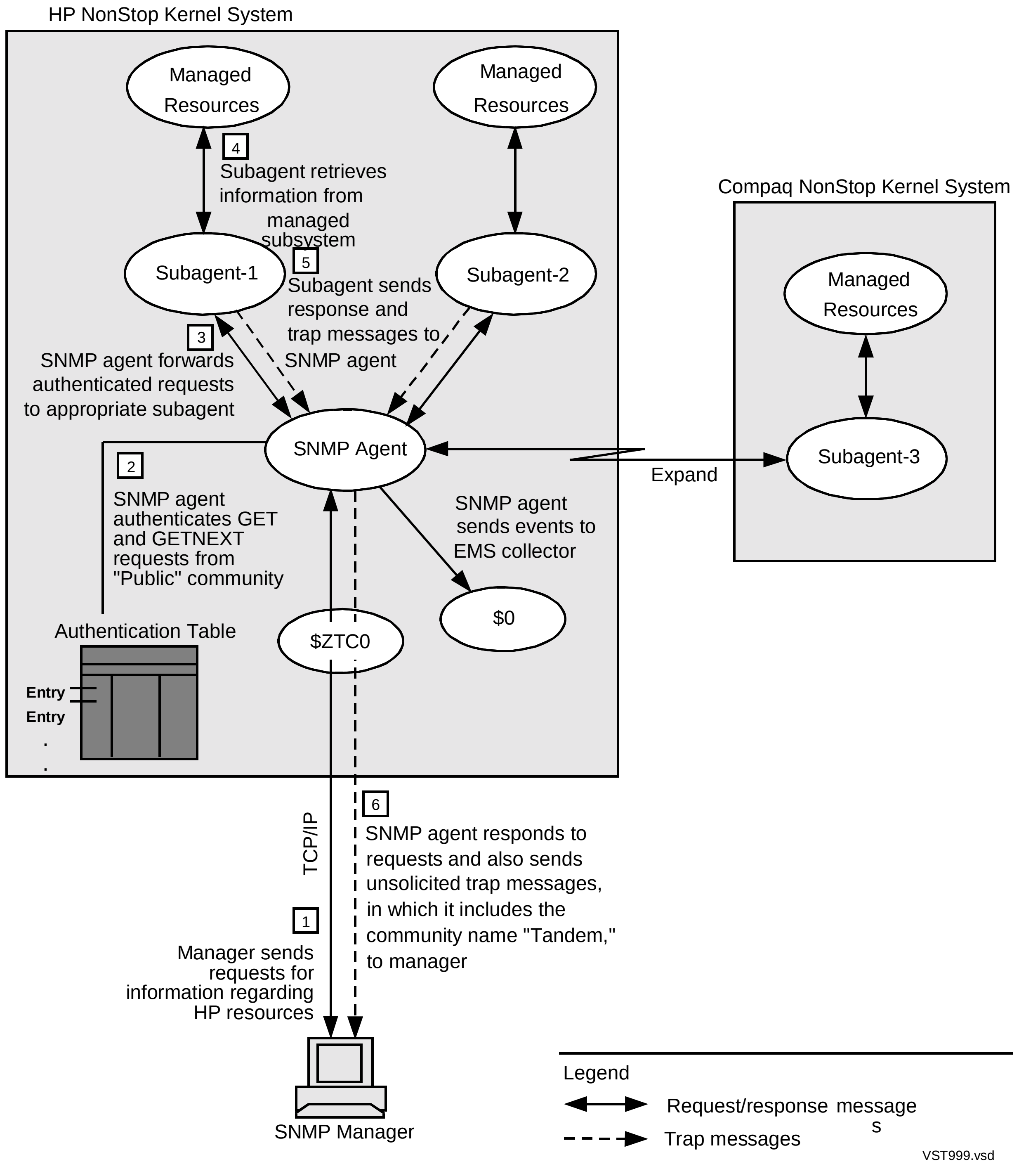
Part I consists of the following sections, which give an overview of the SNMP agent and provide installation and configuration information:

1 The NonStop SNMP Environment

This section explains how HP has implemented SNMP to facilitate management of its HP NonStop systems by using SNMP-compliant applications known as managers.

SNMP originated in the Internet community in the late 1980s as a means for managing TCP/IP and Ethernet networks. Because of its relative simplicity, SNMP has gained widespread acceptance as a protocol for managing devices from multiple vendors on a network. Today, many vendors offer SNMP-compliant applications that run on several workstation platforms. These applications manage devices attached to various kinds of networks when the devices are instrumented with SNMP-compliant software known as **SNMP agents**.

Figure 1-1. Default SNMP Agent/Manager Interaction Over TCP/IP


HP NonStop Kernel System
Managed Resources
Managed Resources
4
Subagent retrieves information from managed subsystem
Subagent-1
5
Subagent sends response and trap messages to SNMP agent
Subagent-2
3
SNMP agent forwards authenticated requests to appropriate subagent
SNMP Agent
2
SNMP agent authenticates GET and GETNEXT requests from "Public" community
SNMP agent sends events to EMS collector
Authentication Table
Entry
Entry
$ZTC0
$0
Compaq NonStop Kernel System
Managed Resources
Subagent-3
Expand
TCP/IP
6
SNMP agent responds to requests and also sends unsolicited trap messages, in which it includes the community name "Tandem," to manager
1
Manager sends requests for information regarding HP resources
SNMP Manager
Legend
Request/response messages
Trap messages
VST999.vsd

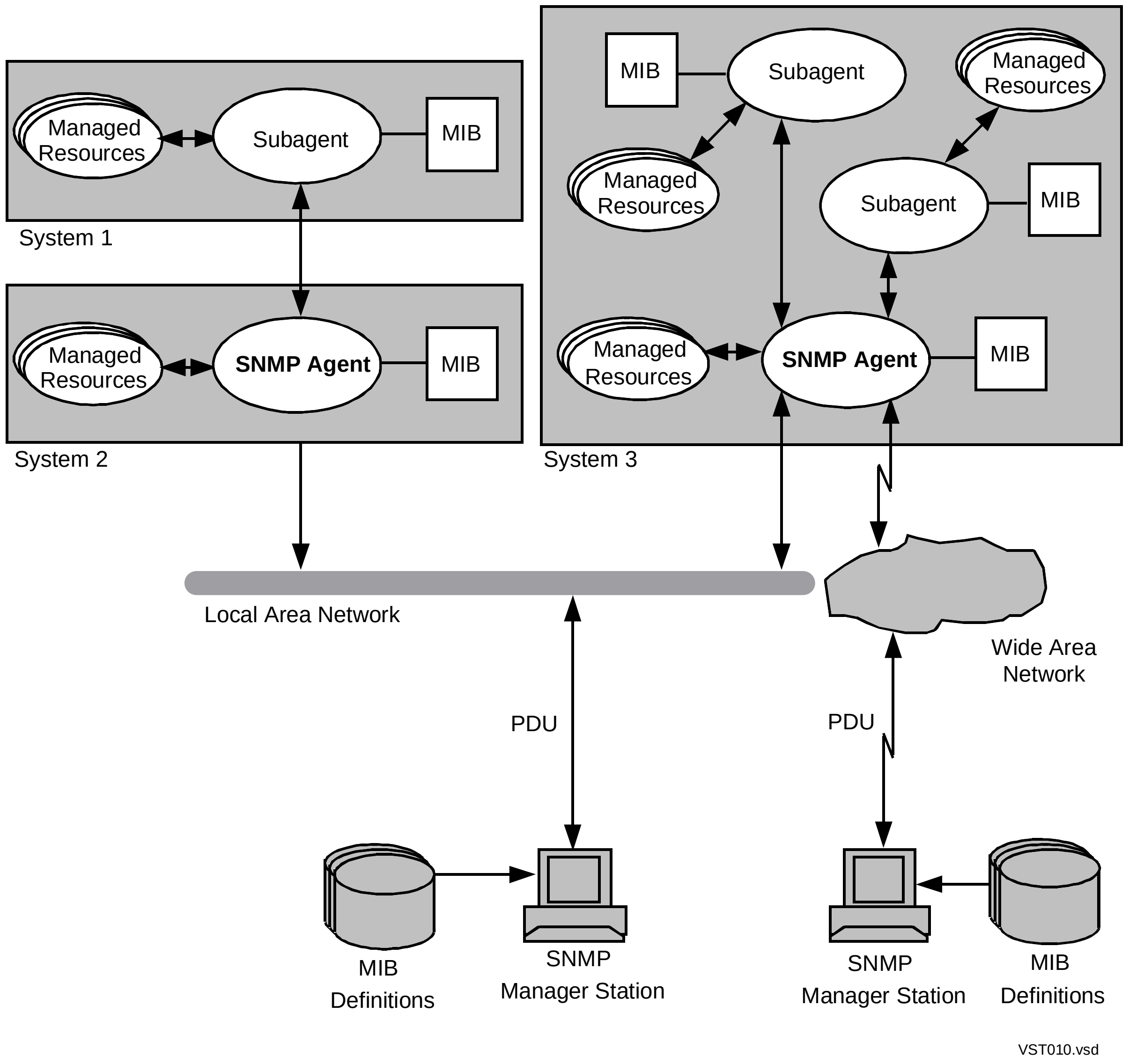
By default, the SNMP agent does the following:

- Receives and sends SNMP messages through NonStop Kernel IPC calls or through any available subnet associated with the TCP/IP process $ZTC0 on the local node
- Accepts any request received through TCP/IP that contains the community name “public,” and gives the “public” community READONLY access
- Accepts any request received through the IPC interface and grants the requesting manager READWRITE access to all MIB objects supported by the SNMP agent and subagents except the SNMP agent’s private zagInProfile group objects

  Note. Objects in the SNMP agent’s zagInProfile group (see Using SCF to Configure Components of the SNMP Agent Environment on page 2-16) are accessible only if the process accessor ID (PAID) of the manager process is compatible with that of the agent process.

- Sends traps through the TCP/IP process $ZTC0 to the Internet address associated with each SNMP manager from which a request is received; includes the community Tandem in the traps. $ZTC0 is on the local node. The SNMP manager must be using TCP/IP.
- Sends the event messages that the SNMP agent generates to the EMS collector process $0 on the local node.
- Forwards requests pertaining to managed resources other than those defined in the System and SNMP MIB-II groups to the appropriate subagent and returns responses back to the requesting manager. The SNMP agent processes the SNMP MIB-II groups.

Figure 1-2. Key Components in the SNMP Environment



Subsystem Components

Following are descriptions of the subsystem components.

SNMP Manager Station

A device on which an SNMP manager runs.

SNMP Manager

An application that automates the management of network elements (**managed resources**) under the control of one or more SNMP agents. HP provides an SNMP agent on its NonStop systems, and other vendors provide SNMP agents on their devices. An agent communicates with a manager using the message and information protocol defined in public SNMP documents known as **RFCs**.

The SNMP manager sends requests to and receives responses from agent processes. Communication between managers and the SNMP agent occurs through the NonStop TCP/IP subsystem, the Parallel Library TCP/IP subsystem or the NonStop TCP/IPv6 subsystem. The NonStop TCP/IP, Parallel Library TCP/IP, and Nonstop TCP/IPv6 subsystems support both local area networks (LANs) and wide area networks (WANs) or, through calls to NonStop Kernel interprocess communication (IPC) procedures, either locally or over *Expand*, when the manager resides on NonStop Kernel systems.

Note. H-series RVUs do not suport Parallel Library TCP/IP

PDU

Organization of messages exchanged between managers and agents. **Protocol data units** (**PDUs**), defined in RFCs, conduct specific operations, such as retrieving a value, changing a value, or sending unsolicited notifications, known as **traps**, to an SNMP manager. Message Protocol on page 1-6 provides more detail about the various kinds of PDUs.

MIB

Information exchanged between managers and agents. An MIB describes a collection of objects that can be managed. An example of an MIB object is the physical location of a node. MIBs are described in a language known as **Abstract Syntax Notation One** (**ASN.1**). Some MIBs are Internet MIBs defined in RFCs. Other MIBs are vendor defined.

SNMP agents access and modify values for MIB objects on behalf of SNMP managers. An SNMP manager can interpret MIB values when the SNMP manager has access to a compiled version of the ASN.1 MIB definition. For more about MIBs, see Information Protocol on page 1-9.

SNMP Agent

A process that intercepts PDUs from an SNMP manager and responds to them with PDUs. The agent accesses information described in MIBs, either directly or through SNMP subagents. The SNMP agent acts as a server for any SNMP network management requester, providing information about HP resources. The SNMP agent supports two MIB-II groups (System and SNMP) defined by RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*, and a private group (zagInternal) that is defined by HP. For more information about how the

SNMP agent is implemented, see Architectural Overview of NonStop SNMP on page 1-14.

SNMP Subagent

Entity used by HP (and other vendors), in their SNMP implementations. SNMP subagents handle a particular collection of resources. Some subagents are implemented as independent processes, and some are bound into other processes, such as the agent process. Some subagents, such as those shown in Figure 1-2, have their own MIBs. Other subagents might implement multiple MIBs, or even share a MIB with another subagent. Some subagents can reside on a system different from the agent's system. For details about each subagent HP currently offers, refer to Part II, SCF for the SNMP Agent of this manual.

Toolkits

HP offers a Subagent Toolkit that helps programmers generate subagents that interoperate with the SNMP agent. Subagents make customer-written NonStop applications manageable by SNMP managers. For more information about the Subagent Toolkit, refer to the *SNMP Subagent Programmer's Guide*.

HP also offers a Manager Services toolkit that allows C and C++ programmers to create SNMP managers that run as NonStop Kernel processes in either the Guardian or in the Open System Services (OSS) environment. For information about Manager Services, refer to the *SNMP Manager Programmer's Guide*.

Message Protocol

In an SNMP environment, agents and managers communicate through the exchange of three types of messages:

Request messages	Sent by a manager to initiate an action by an agent
Response messages	Sent by an agent to respond to a manager's request
Trap messages	Sent by an agent to notify a manager of the occurrence of significant events

Each SNMP message contains a unit of information called a **PDU**. Five types of PDUs are supported by any SNMP agent:

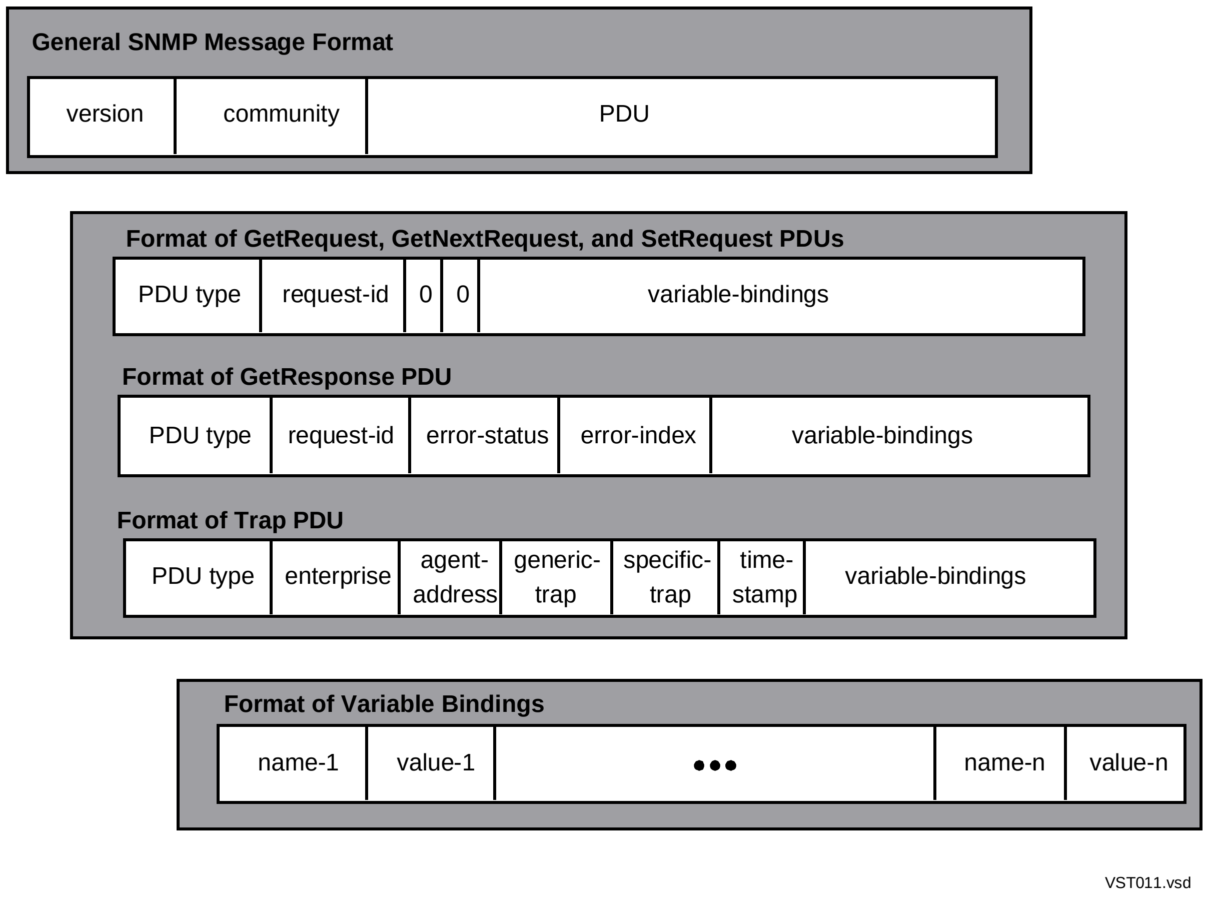
- GetRequest
- GetNextRequest
- SetRequest (request PDUs)
- GetResponse (response PDU)
- Trap PDU

The format of SNMP messages is summarized in Figure 1-3 and described in more detail in the following subsection.

General SNMP Message Format

Each PDU is embedded in a message, or packet, beginning with a version number and a community name. The **version number** identifies the version of SNMP being implemented; currently, this value is always 1 for NonStop SNMP messages. The **community name** identifies one or more SNMP managers; it is used for access control. Following the version and community identifiers is one of the five types of PDUs.

Figure 1-3. Format of SNMP Messages



Request PDUs

The three kinds of request PDUs have the same format as the response PDU, but the **error-status** and **error-index** fields are always set to 0. The **request-id** uniquely identifies individual requests. **Variable-bindings** are a list of variable names and corresponding values.

The three request PDUs are:

- **GetRequest PDU**. This PDU performs a **Get operation**. The Get operation retrieves a value of a MIB object.
- **GetNextRequest PDU**. This PDU performs a **GetNext operation**. The GetNext operation retrieves the value of the object instance that is next in **lexicographic order**.
- **SetRequest PDU**. This PDU performs a **Set operation**. The Set operation alters the value of a MIB object.

Response PDU

The function performed by a **GetResponse PDU** is referred to as an SNMP **Response operation**. The Response operation returns a response to the originator of a Get, GetNext, or Set operation. The response operation contains a **request-id** that associates it with a request PDU. The response PDU provides such information as error indications and values of MIB objects. Error-status values are described in Appendix C, Unsolicited SNMP Agent Messages.

Trap PDU

The function performed by a Trap PDU is referred to as an SNMP **Trap operation**. The Trap operation issues an unsolicited message to an SNMP manager to signal an important event. The **enterprise** field provides the SNMP agent object identifier or subagent object identifier; the object identifier for the SNMP agent is 1.3.6.1.4.1.169.3.155.1. (Object identifiers are described under Information Protocol on page 1-9 The **agent-address** field is the Internet address of the device on which the agent forwarding the trap is installed. The **generic-trap** and **specific-trap** fields characterize the trap as a particular kind of trap. Values for traps that originate from the SNMP agent are described in Appendix C, Unsolicited SNMP Agent Messages. Values for traps from subagents are described in Section 9, EMS Trap Subagent.

Transmitting and Receiving SNMP Packets

SNMP managers convert user requests into SNMP messages, or packets, as illustrated in Figure 1-3 on page 1-7. The manager then encodes these SNMP packets into a format that can be transported to and decoded by the target agent.

IPC-Encoded and BER-Encoded SNMP Packets

The SNMP agent can process SNMP packets encoded using the HP IPC format. The IPC format provides the best encoding and decoding performance on NonStop Kernel systems.

Also, all agents can interpret information encoded according to the basic encoding rules (BER) associated with ASN.1, which define how to encode an ASN.1 value as an octet string. BER-encoded packets can be transmitted over any transport protocol that the manager and target agent mutually support.

Transmission Protocols Supported by the SNMP Agent

IPC-encoded SNMP packets are transmitted and received through NonStop Kernel interprocess communication calls. The SNMP agent supports the interface, #MGR, through which an IPC communication path between an SNMP manager and a SNMP agent can be established.

The most widely supported transmission protocol for transmitting and receiving BER-encoded SNMP packets is TCP/IP's User Datagram Protocol (UDP). The SNMP agent supports the UDP protocol.

Differences in the way the SNMP agent handles requests received through the HP NonStop Kernel IPC and requests received through the UDP protocol are discussed Section 2, Installing and Configuring the SNMP Agent and in Section 3, MIBs Supported by the SNMP Agent.

Information Protocol

The foundation of any network management system is a collection of information about the elements to be managed. In SNMP environments, this information is described by a MIB (Management Information Base). Each item in a MIB is known as a **MIB object**.

- **Internet-standard MIBs**. These MIBs are approved by the Internet Architecture Board (IAB). MIB-I, MIB-II, and Remote Network Monitoring MIB. MIB-I is the original MIB for managing TCP/IP-based internets. MIB-II is an extended version of MIB-I and provides additional object definitions. Remote Network Monitoring MIB defines objects that manage remote network monitoring devices. Most MIB-II objects are supported by the TCP/IP Subagent; two MIB-II groups (System and SNMP) are supported by the SNMP agent. The Host Resources Subagent supports the standard Host Resources MIB, which is contained in the MIB-II subtree.
- **Experimental MIBs**. These MIBs are used in Internet experiments.
- **Enterprise MIBs**. These MIBs are vendor defined.

To promote interoperability, SNMP defines a standard scheme for naming all MIB objects.

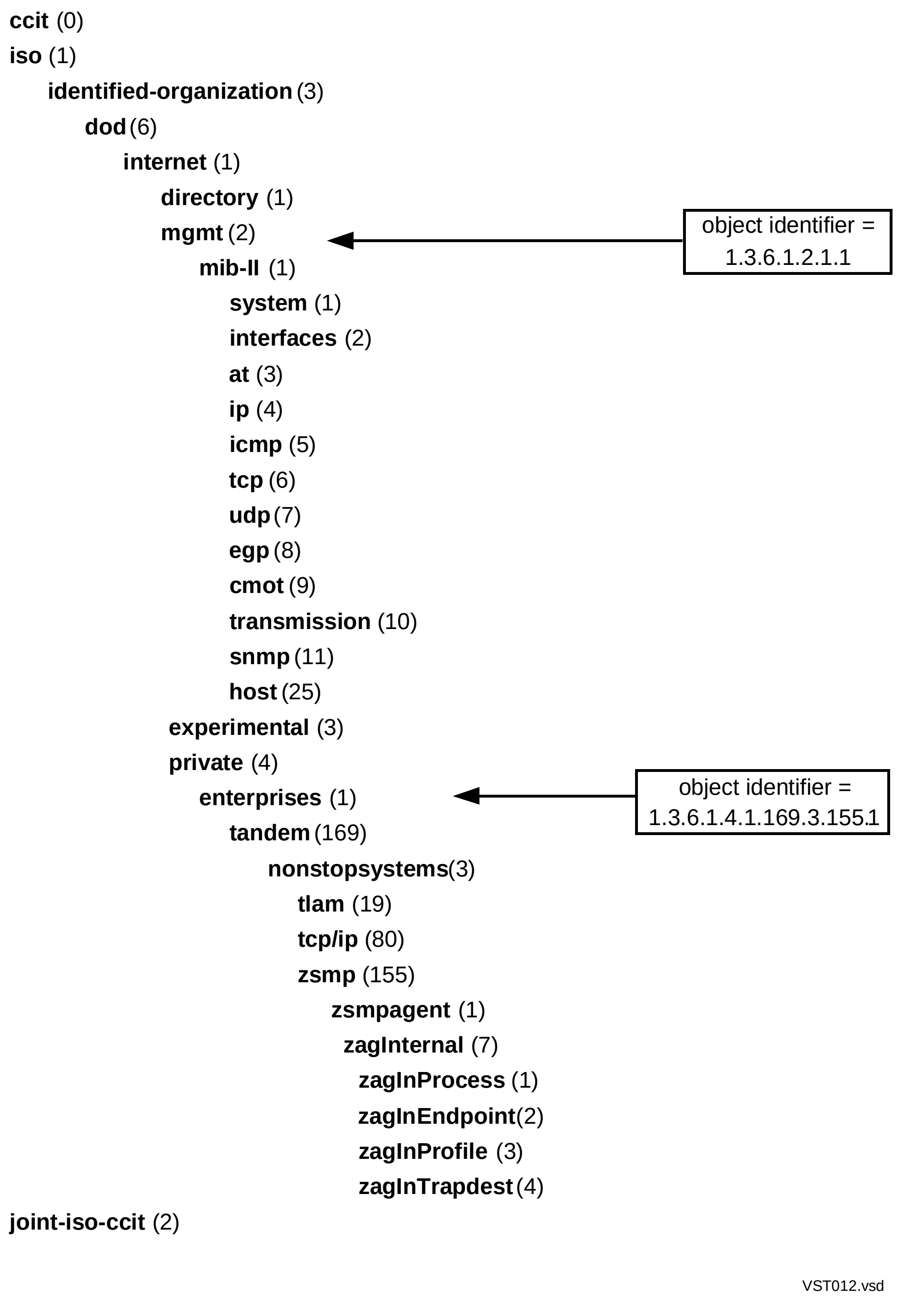
SNMP Naming Scheme

SNMP uses a hierarchical model to identify objects. Every object in an SNMP environment is identified by an **object descriptor** and an **object identifier**. Object descriptors are logical names. Object identifiers are expressed in numeric dot notation. For example, the System group of MIB-II is identified as:

```
system
1.3.6.1.2.1.1
```

As Figure 1-4 shows, an object identifier is a sequence of integers resulting from traversing a global tree. The object tree is composed of an unlabeled **root** connected to labeled **nodes**. Each node in turn can have subordinate nodes of its own. A node and its subordinate nodes make up a **subtree**. A node that has no subordinates is referred to as a **leaf** object.

Figure 1-4. Nodes in the SNMP Object Tree



As Figure 1-4 indicates, MIB-II is described in the **mgmt** subtree. **tandem** has been assigned the value 169 in the **enterprises** subtree. In the tandem subtree, **nonstopsystems** has been assigned the number 3. The nonstopsystems subtree contains MIBs and subsystems defined by HP.

In the **zsmp** subtree of the nonstopsystems node, the SNMP agent subtree, **zsmpagent**, has been assigned the value 1. The object identifier for the SNMP agent is:

```
1.3.6.1.4.1.169.3.155.1
```

In the zsmpagent subtree, the **zagInternal** group (value of 7), encompasses the four subgroups (**zagInProcess**, **zagInEndpoint**, **zagInProfile**, and **zagInTrapdest**) that constitute the private portion of the SNMP agent's MIB. The object identifier for this private MIB's zagInternal group is:

```
1.3.6.1.4.1.169.3.155.1.7
```

Object identifiers not only identify organizations and their elements but also identify MIB objects. For example, the **zagInTdHostAddr** object from the zagInTrapdest group of the SNMP agent's private MIB is:

```
1.3.6.1.4.1.169.3.155.1.7.4.2.1.4
```

The Management Information Base

Each MIB object is identified by an object descriptor and identifier. Sometimes MIB objects are divided into groups called **object groups**. Most items subordinate to MIB-II in Figure 1-4 are actually object groups. Examples are the UDP group, the Interfaces group, and the TCP group. Many of these groups are supported by the TCP/IP Subagent's MIB, as described in Section 8, TCP/IP Subagent.

Values for individual MIB objects can two forms:

- **Scalar**. An object is assigned a scalar value when the object can have only one value. The length of time since the SNMP subsystem was brought up (sysUpTime) is an example of such an object.
- **Tabular**. Some objects, such as TCP connections, can have multiple values, each describing a single instance of the object. These object values are organized into tables. Each row in the table represents information about a single object instance. A row in a table is referred to as an **entry**.

Each MIB object has an **access attribute**. This attribute determines the kind of access SNMP managers have to values of the object.

read-only	The value can be retrieved but not modified.
read-write	The value can be both retrieved and modified.
write-only	The value can be modified but not retrieved.
not-accessible	The value cannot be retrieved or modified.

In this manual, descriptions of MIB objects are normally provided in tables with four columns (in some instances, the third and fourth columns have been combined). This excerpt from the table describing the System group objects in the SNMP agent's MIB is followed by a description of each column.

Table 1-1. System Group Objects Supported by SNMP Agent's MIB

Object and Attributes	Definition	Format of Value	Derivation of Value
sysContact 1.3.6.1.2.1.1.4 read-write DisplayString (SIZE (0..255))	Information describing the contact person for this node. This value is stored in SNMPCTL.	The default value is unknown.	Initially, the default value. Can be set from an authorized SNMP manager.

Column 1	The first column identifies the MIB object and lists its attributes. This information appears for each object: its name (object descriptor), its numeric identifier (object identifier), its access attribute, and its ASN.1 syntax.
Column 2	The second column defines what the object's value means.
Column 3	The third column describes the format of the value.
Column 4	The fourth column indicates how the value was derived.

For a description of the objects in the SNMP agent's MIB, see Section 3, MIBs Supported by the SNMP Agent. For descriptions of the objects in each subagent's MIB, see Part IV, SNMP Subagents, of this manual. When objects are organized into tables, the sections provide information about how the tables are created and maintained.

Traps

Traps are unsolicited messages forwarded by an agent to SNMP managers when significant events occur. Some traps and their contents are defined by the SNMP standards. Other traps are proprietary. Traps that the SNMP agent supports are defined in Part 3, MIBs Supported by the SNMP Agent. Traps that subagents offered by HP generate are described in Part IV, SNMP Subagents.

The SNMP agent can be configured to send traps to one or more specific SNMP managers. You can also suppress traps altogether.

Architectural Overview of NonStop SNMP

Figure 1-5 illustrates how the SNMP agent interacts with NonStop system and network elements to provide SNMP support:

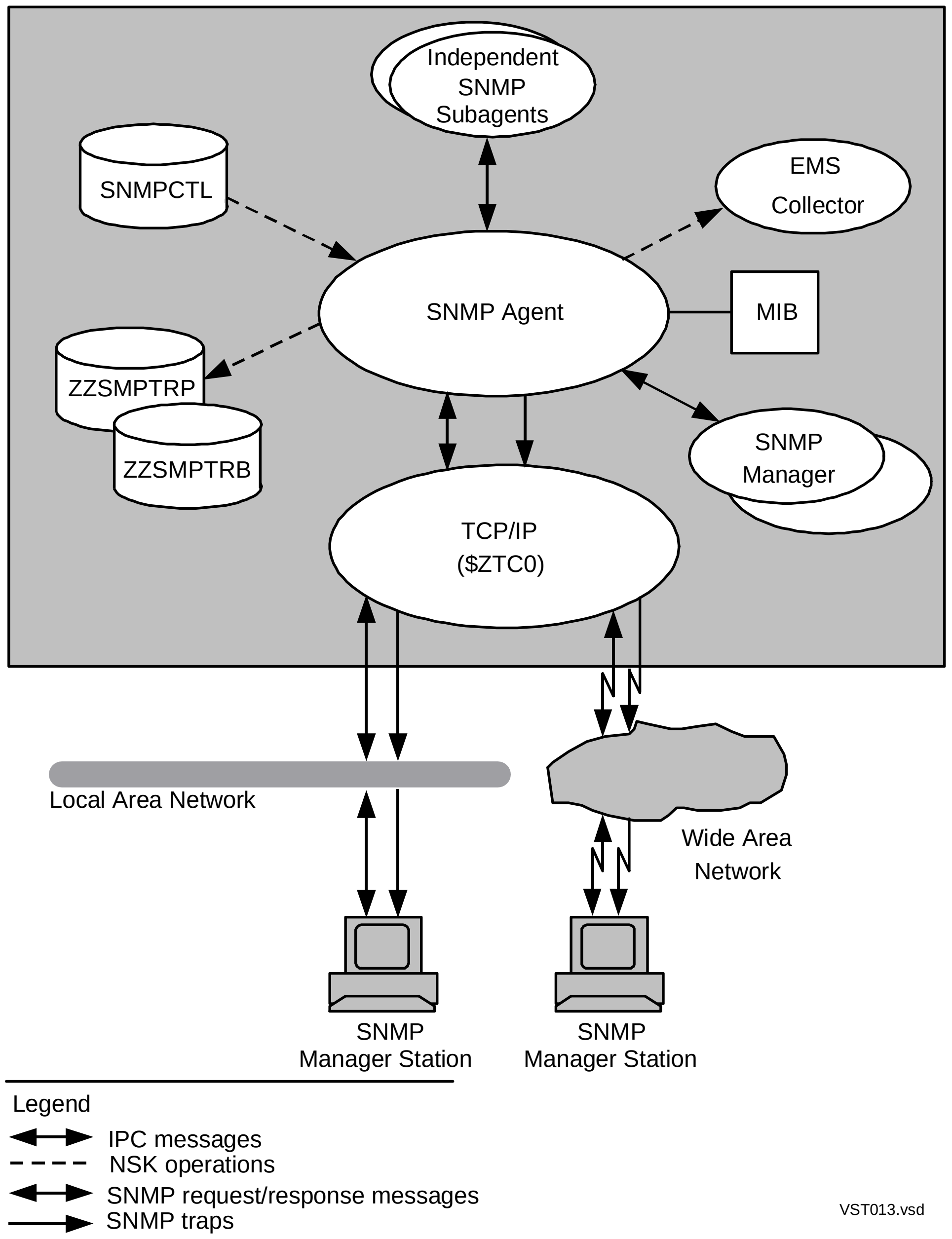
- The SNMP agent communicates with the remote SNMP managers through the TCP/IP subsystem and these underlying subsystems:
    - X.25 Access Method (X25AM)

      The X25AM subsystem supports WAN data communications over an X.25 packet-switching network and is supported on G-series and H-series RVUs of the NonStop Kernel.
    - ServerNet LAN Systems Access (SLSA) subsystem

      The SLSA subsystem supports parallel LAN I/O on systems that implement ServerNet.
- The SNMP agent authenticates requests as it receives them using the scheme described in Section 2, Installing and Configuring the SNMP Agent.
- The SNMP agent uses the Subsystem Programmatic Interface (SPI) protocol to send event messages to an Event Management Service (EMS) collector. Refer to Appendix C, Unsolicited SNMP Agent Messages, for a description of event messages originating from the SNMP agent and to Part IV, SNMP Subagents for events from subagents.
- Most of the MIB-II groups are supported by the TCP/IP Subagent (see Section 8, TCP/IP Subagent), which supports the management of TCP/IP resources. Two MIB-II groups (System and SNMP) are supported by the SNMP agent. (See Section 3, MIBs Supported by the SNMP Agent.)
- The SNMP agent also supports a private MIB (zagInternal group) that describes attributes of the SNMP agent process and defines the resources the SNMP agent
- uses. These private MIB objects are defined by HP and are used to control the configuration and operation of the SNMP agent.

- The SNMP agent uses Interprocess Communication (IPC) messages to communicate with independent subagents about the resources they manage. Subagents can reside on the same node as the SNMP agent or on different nodes. Each SNMP agent can support one instance of any particular subagent.
- The SNMP control file (SNMPCTL) contains configuration parameters for the SNMP agent.
- When the primary SNMP agent process is active, it writes trace records to the file ZZSMPTRP. When the backup process becomes active, it writes to the file ZZSMPTRB.

Figure 1-5. NonStop SNMP Architecture



RFC Compliance

The NonStop SNMP implementation complies with standards and guidelines published in these RFCs:

- RFC 1155, *Structure and Identification of Management Information for TCP/IP-Based Internets*. This document describes MIB naming conventions, syntax, and other MIB object characteristics.
- RFC 1157, *A Simple Network Management Protocol (SNMP)*. This document describes the SNMP architecture and message protocol.
- RFC 1212, *Concise MIB Definitions*. This document describes the ASN.1 conventions used to define SNMP MIBs.
- RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*.
- RFC 1215, *A Convention for Defining Traps for Use With the SNMP*. This document describes ASN.1 conventions for defining traps.

The degree to which subagents comply with MIB object definitions in RFCs is often documented in this manual in three-column tables. For example, the following excerpt from a table describes how the TCP/IP Subagent's MIB complies with IP group definitions in RFC 1213 and is followed by an explanation of each column in the table.

Table 1-2. Compliance With IP Group Definitions in RFC 1213

Object Descriptor	Compliance	Explanation
ipForwarding	Partial	Set operation not supported

Column 1	The first column lists the object descriptor.
Column 2	The second column contains one of three values indicating the degree to which implementation of the object complies with RFC guidelines. “Yes” indicates full compliance. “Partial” indicates some but not full implementation of the object. “No” indicates no support for the object.
Column 3	The third column provides comments that explain deviations and other special behavior relative to the object.

Related Documents

The RFCs listed earlier and in other sections of this manual are public-domain documents that you can obtain from InfoWay or from one of the following sources:

- DDN Network Information Center
  14200 Park Meadow Drive, Suite 200
  Chantilly, VA 22021
  USA
  phone: 1-800-365-3642
  mail: nic@nic.ddn.mil

- The Internet. If your site has IP-connectivity to the Internet community, you can use anonymous FTP to the host nic.ddn.mil (residing at 192.112.36.5) and retrieve files from the directory

  ```
  rfc/
  ```

- Electronic mail. Send a message to

  ```
  mail-server@nisc.sri.com
  ```

  In the subject field, indicate the RFC number:

  ```
  Subject:  SEND rfcs/rfc1130.txt
  ```

2 Installing and Configuring the SNMP Agent

The section tells you how to install, start, and stop the SNMP agent and how to configure the following:

- SNMPCTL file (The SNMPCTL File on page 2-6)
- Multiple SNMP agents per node Starting Multiple SNMP Agents on Each Node on page 2-23)
- EMS collector (Configuring the EMS Collector on page 2-25)
- Security (Configuring Security on page 2-26)
- TCP/IP request/response connections (Configuring TCP/IP Request/Response Connections on page 2-34)
- Trap destinations (Configuring Trap Destinations on page 2-38)

Installation

The SNMP agent is delivered ready to be installed and started without any custom configuration.

You install the SNMP agent using the Distributed Systems Management/Software Configuration Manager DSM/SCM product. The SNMP agent program file is installed on your system as:

```
$SYSTEM.SYSTEM.SNMPAGT
```

Note. For G-series or later releases of the NonStop Kernel, DSM/SCM replaces the Install program as the standard tool for installing and managing new software.

Refer to the *DSM/SCM User's Guide* for complete software installation information.

The device type for the SNMP agent is 31.

Before You Configure the SNMP Agent Operations Environment

To configure SNMP you must get the following information from the person responsible for the SNMP agent and from the person responsible for the SNMP manager with which the SNMP agent communicates. This information you gather gives you an idea of the configuration you’ll need.

- The type of access required by the SNMP managers that communicate through TCP/IP to resources managed by the SNMP agent and subagents
- The Internet address and community name of any SNMP manager that communicates through TCP/IP for which you want to refine the default security scheme
- The Internet address of any SNMP manager that expects to receive trap messages
- The community name, if any, that the SNMP manager expects to see in trap messages received from the SNMP agent
- The HP resources that the SNMP manager wants to manage

Note. The SNMP implementation supported by HP uses subagents to handle various collections of resources. The HP resources the SNMP manager wants to manage determine the subagents to be installed. The subagents to which the SNMP agent forwards requests are installed, started, and configured separately from the SNMP agent. Defining the connection between the subagent and the SNMP agent is part of the subagent’s configuration, not the SNMP agent’s configuration.

- The Internet addresses by which the SNMP agent can be reached
- The SNMP traps and MIB objects the SNMP agent and its subagents support and what they mean in the context of the NonStop environment
- The community name that must be included in requests to the SNMP agent
- The location of the NonStop system on which the SNMP agent resides

Initialization Tasks

Before you begin to configure the SNMP agent you should understand what the SNMP agent does when it is started to give you an idea of the state of the SNMP agent in which you'll be configuring. When you start the SNMP agent, it performs these initialization tasks:

- Ensures that the SNMP agent is running as a named process. If you did not specify a name, TACL assigns one.
- Ensures that the SNMP agent is running with super user group access privileges, if the PORT startup parameter is port 161 (the standard SNMP port for receiving manager requests) or a port number less than 1023. If the port number is greater than 1023, the SNMP agent generates an EMS event stating that the SNMP agent must run under a super group creator accessor ID and stops running. Refer to the description of the PORT startup parameter (page 2-10) for more information on this topic.
- Opens any TCP/IP subnets configured for communications with SNMP managers. If no TCP/IP subnets are available, the agent process starts but generates socket-error and state-change EMS event messages.
- Starts a backup process if you requested one.
- Processes configuration information. Creates a trap destination definition (if none already exists) for the Internet address associated with any manager from which the SNMP agent receives a request through TCP/IP.

The Default SNMP Agent

When started without any customization, the SNMP agent:

- Receives and sends SNMP messages through NonStop Kernel IPC calls or through any available subnet associated with the TCP/IP process $ZTC0 on the local node.
- Accepts any request received through TCP/IP that contains the community name “public” and gives the “public” community READONLY access.
- Accepts any request received through the IPC interface and grants the requesting manager READWRITE access to all MIB objects supported by the SNMP agent and subagents except the SNMP agent's private zagInProfile group objects.

Note. Objects in the SNMP agent's zagInProfile group (see SNMP Agent Private MIB Objects on page 3-31) are accessible only if the process accessor ID (PAID) of the manager process is compatible with that of the agent process. See Authentication Table Entries on page 3-31 for more details.

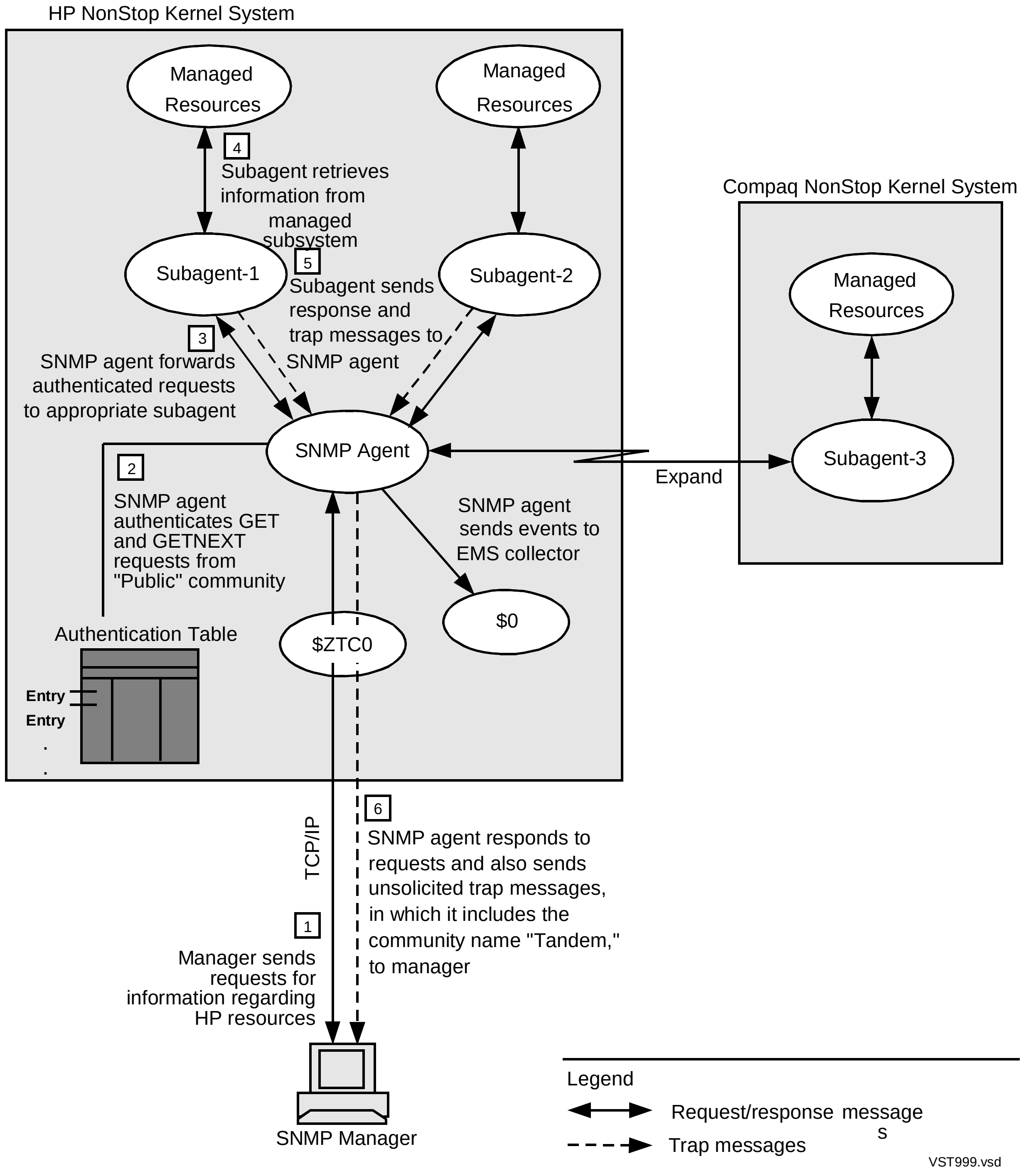
- Sends traps through the TCP/IP process $ZTC0 on the local node to the Internet address associated with each SNMP manager communicating using TCP/IP from which a request is received. Includes the community “Tandem” in the traps.

- Sends event messages it generates to the EMS collector process $0 on the local node.
- Forwards requests pertaining to managed resources other than those defined in the System and SNMP MIB-II groups (which the SNMP agent itself processes) to the appropriate subagent, and returns responses to the requesting manager.

The subagents to which the SNMP agent forwards requests are installed, started, and configured separately from the SNMP agent. Defining the connection between the subagent and the SNMP agent is part of the subagent's configuration, not the SNMP agent's configuration. Section IV, SNMP Subagents, describes installing, starting, and configuring the subagents that HP supports.

Figure 2-1 illustrates the default SNMP agent operating in an SNMP network management environment.

Figure 2-1. Default SNMP Agent/Manager Interaction Over TCP/IP


HP NonStop Kernel System
Managed Resources
Managed Resources
4
Subagent retrieves information from managed subsystem
Subagent-1
5
Subagent sends response and trap messages to SNMP agent
Subagent-2
3
SNMP agent forwards authenticated requests to appropriate subagent
SNMP Agent
2
SNMP agent authenticates GET and GETNEXT requests from "Public" community
SNMP agent sends events to EMS collector
Authentication Table
Entry
Entry
$ZTC0
$0
Compaq NonStop Kernel System
Managed Resources
Subagent-3
Expand
TCP/IP
6
SNMP agent responds to requests and also sends unsolicited trap messages, in which it includes the community name "Tandem," to manager
1
Manager sends requests for information regarding HP resources
SNMP Manager
Legend
Request/response messages
Trap messages
VST999.vsd

The SNMPCTL File

When started, the SNMP agent creates an SNMP control file (SNMPCTL) in the subvolume from which it is started. You control the SNMP agent by configuring the SNMPCTL file. If started using the TRACE startup parameter, the SNMP agent also creates two trace files (ZZSMPTRP and ZZSMPTRB) in the subvolume from which it is started.

Initially, the values shown in Table 2-1 are assigned to several objects in the SNMP agent's MIB. The SNMP agent's MIB is described in detail in Section 3, MIBs Supported by the SNMP Agent.

Table 2-1. Initial Values of Objects in the SNMP Agent's MIB

MIB Object	**Initial Value**
System group objects:	

sysDescr	`Tandem NonStop Kernel System`
	`Version: D`*`xx`*
	`Node: `*`node-name`*
	`Agent: `*`agent-process-name`*
	where D*xx* is the product version of the SNMP agent; for example, D23.

sysObjectID	1.3.6.1.4.1.169.3.155.1
sysContact	`"unknown"`
sysName	A name associated with the default TCP/IP process handling communication between the SNMP agent and SNMP managers. It is derived by a gethostname socket call against the TCP/IP process specified in the TCPIP^PROCESS^NAME startup parameter or, if none is specified, against $ZTC0 on the local node.
sysLocation	`"unknown"`
sysServices	79
SNMP group object:	
snmpEnableAuthenTraps	1 (yes)

Configure the SNMCTL File to Control the SNMP Agent

Configure the SNMPCTL file in one of four ways:

- RUN command parameters (RUN Command on page 2-7)
- PARAM statements (PARAM Statements' Custom Configuration for the RUN Command on page 2-16)

- The Subsystem Control Facility (SCF) (Using SCF to Configure Components of the SNMP Agent Environment on page 2-16)
- SNMP Set operations on SNMP agent private MIB objects (Configuring the SNMP Agent Through SNMP Requests on page 2-17)

Note. You also define some components of the SNMP agent configuration by using RUN line startup options, but these settings persist only as long as the agent process is running.

When the SNMP agent starts, it evaluates all startup parameters in this order:

1. A default value is assigned to every startup parameter.
2. PARAM statements are processed. Unrecognizable or erroneous parameters are ignored without comment.
3. Startup parameters in the RUN command line are processed. With the exception of the SWAPVOL startup parameter, unrecognized or erroneous parameters cause the SNMP agent process to terminate. If you define a swap volume as anything other than a local disk, the SWAPVOL startup parameter is ignored, and a warning message is issued.

RUN Command

To start the SNMP agent, use the RUN command. No additional configuration beyond that provided through the RUN command startup parameters is required for the SNMP agent to receive and reply to SNMP requests solely through NonStop Kernel IPC calls. However, even in this limited scope, additional configuration is required to allow the SNMP agent to forward traps to SNMP managers that communicate through NonStop Kernel IPC calls.

```
[RUN] [[$volume.]subvolume.]SNMPAGT
   / NAME [$agent-process] [,other-run-option]... /
   [startup-parameter [,startup-parameter]...]
```

volume

identifies the volume on which the SNMP agent program file SNMPGT resides. You can omit the SNMP agent program if SNMPAGT resides on your current subvolume.

By default, when you install the SNMP agent, SNMPAGT is placed in $SYSTEM.SYSTEM.

subvolume

identifies the subvolume on which SNMPAGT resides. You can omit the subvolume parameter if it is named in your TACL #PMSEARCHLIST.

agent-process

identifies the agent process. You can specify one through five alphanumeric characters, but the first character must be alphabetic. HP recommends using $ZSNMP to identify an SNMP agent. If you are running multiple SNMP agents on a single host (see Starting Multiple SNMP Agents on Each Node on page 2-23), HP recommends appending a digit: for example, $ZSNM0 and $ZSNM1.

If you don't supply a value for *agent-process* at /NAME TACL generates a name, because the SNMP agent must run as a named process. You need to know the name of the agent process to use SCF to configure the SNMP agent. You also need to know the name of the agent process to stop it. The SNMPCTL File and the WARM Startup Parameter on page 2-14 tells you how to find out the name of the agent process.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the NOWAIT option so you can resume TACL operations once the SNMP agent is started.

startup-parameter

is one of the following parameters that control attributes of the SNMP agent process. You can specify startup parameters either in the RUN command or through TACL PARAM statements issued before you issue the RUN command. For more information on TACL PARAM statements, refer to PARAM Statements' Custom Configuration for the RUN Command on page 2-16.

```
backup-cpu-number                                 |
COLD                                              |
WARM                                              |
COLLECTOR [$alternate-collector-process]          |
DATAPAGES pages [E[XTENSIBLE]]                    |
MAXOPENERS number-of-subagents                    |
PORT request-port-number                          |
SUBAGENT^TIMEOUT timeout-seconds                  |
SWAPVOL #disk-name                                |
TCPIP^PROCESS^NAME [$node.]$tcp/ip-process        |
TRACE                                              |
TRAPPORT trap-port-number                        |
```

backup-cpu-number

specifies a backup process processor for a persistent agent process pair. The backup process monitors the primary process and takes over if the primary process fails. HP strongly recommends that you use this parameter.

This parameter should be an integer that identifies one of the processors on your system. By default, this parameter is omitted.

`COLD | WARM`

tells the SNMP agent what to do with the SNMP control file (SNMPCTL), if one exists, and where to get the SNMP configuration information. Every SNMP agent process stores configuration information and persistent MIB information in a file named SNMPCTL in the run subvolume.

`COLD`

instructs the SNMP agent to purge any SNMPCTL file in the run subvolume and create a new, empty SNMP file. To enable the SNMP agent to receive requests through TCP/IP, configure at least one ENDPOINT object. You can then complete your configuration either through SCF or through SNMP, as described in Part II, SCF for the SNMP Agent and in Configuring the SNMP Agent Through SNMP Requests on page 2-17.

Note. No additional configuration beyond that provided through the RUN command startup parameters is required for the SNMP agent to receive SNMP requests solely through NonStop Kernel IPC calls.

`WARM`

instructs the SNMP agent to use the SNMPCTL file in the run subvolume. If no SNMP file exists, the SNMP agent creates one and fills it with the default configuration values. WARM is the default value for this parameter.

`COLLECTOR [$`*`alternate-collector-process`*`]`

identifies the EMS collector to which the SNMP agent and its backup process send event messages. The default is $0 on the local node.

To suppress event messages, include this parameter with no value:

```
COLLECTOR
```

To send event messages to an alternate collector, identify the name of the collector process:

`COLLECTOR $`*`alternate-collector-process`*

After the SNMP agent is running, you can specify an alternate collector by using SCF. For more information, refer to Part II, SCF for the SNMP Agent.

If you specify a nonexistent alternate collector process name, event messages are suppressed.

`DATAPAGES` *`pages`* `[E[XTENSIBLE]]`

specifies the amount of storage to allocate for general dynamic memory.

Specify a value for *`pages`* that ranges from 0 to 65280, representing the number of 2048-byte pages to be allocated.

Including the keyword EXTENSIBLE makes the segment extensible. An extensible data segment is one for which swap file extents are not allocated until needed. If you specify 0 pages, the SNMP agent allocates data pages dynamically as needed until one of the following conditions exists:

- Approximately 4 megabytes are allocated.
- The virtual memory limit is reached.
- The underlying disk file space is unavailable.

The default value for this parameter is 2040 EXTENSIBLE. The default SNMP agent configuration uses about 1 megabyte of memory.

`MAXOPENERS` *`number-of-openers`*

specifies the total number of openers (including subagents, managers communicating through NonStop Kernel IPC calls, and SCF) that one SNMP agent process can simultaneously support. The default value is 20.

`PORT` *`request-port-number`*

specifies the port number to monitor for requests from managers communicating over TCP/IP. The default value is 161, the SNMP standard port for receiving requests. If a port number greater than 1023 is used, the agent process need not be associated with the super user group (255, *n*).

You can associate the SNMP agent process with the super user group in two says. From a TACL prompt:

- Log on using a super group user ID before starting the SNMP agent.
- Use the File Utility Program (FUP) to give ownership of the SNMP agent program file (SNMPAGT) to a super group user ID. Then secure SNMPAGT so that a user who does not have a super group user ID can execute SNMPAGT, and set the PROGID so that the owner ID of SNMPAGT is used as the creator accessor ID when the program is run. Refer to the *File Utility Program (FUP) Reference Manual* for more information.

`SUBAGENT^TIMEOUT` *`timeout-seconds`*

is the number of seconds the SNMP agent should wait before canceling a message to a subagent. The default is 3 seconds.

`SWAPVOL #`*`disk-name`*

identifies the disk to use for swapping extended memory.

Specify *`disk-name`* as an unqualified device name. Any qualification is ignored.

The default disk is the volume used for swapping the virtual data SNMP agent processes, as specified by the optional SWAP run option. If you omit both the SWAPVOL startup parameter and the SWAP run option, the volume on which SNMPAGT resides is used for both swapping and extended memory swapping, a situation that can result in insufficient available space.

`TCPIP^PROCESS^NAME [$`*`node`*`.]$`*`tcpip-process`*

specifies the default TCP/IP process to be used when request/response connection points and trap destinations are defined for the SNMP agent. Start the agent using the TCPSAM or TCP6SAM process name as input to this startup parameter only when the product is going to be run on Parallel Library TCP/IP or Nonstop TCP/IPv6.

The TCP/IP process is used as:

- The default NETWORK attribute value for ENDPOINT objects created through SCF and the default zagInEpNetwork object value for zagInEndpointTable rows created through SNMP
- The default NETWORK attribute for TRAPDEST objects created through SCF and the default zagInTdNetwork object value for zagInTrapdestTable rows created through SNMP

If you do not include the TCPIP^PROCESS^NAME startup parameter, the default TCP/IP process is $ZTC0 on the local node.

`TRACE`

enables tracing of the agent process if the backup process takes over. By default, tracing is not enabled.

When you enable tracing through TRACE, the SNMP agent creates a primary trace file (ZZSMPTRP) and a backup trace file (ZZSMPTRB) in the subvolume from which the primary trace file was started. To specify different trace file names, use a TACL DEFINE statement. For example:

```
ADD DEFINE =PRIMARY-TRACE-FILE, CLASS MAP, FILE $SYSTEM.SNMP.SNMPTRCP
ADD DEFINE =BACKUP-TRACE-FILE, CLASS MAP, FILE $SYSTEM.SNMP.SNMPTRCB
```

For information about the DEFINE statement, refer to the *TACL Reference Manual*.

Use the TRACE startup parameter to enable tracing in the backup process. You can initiate tracing of the primary agent process by using the SCF TRACE command (as discussed in Section 7, Troubleshooting the SNMP Agent), but tracing stops when the backup process takes over. The SCF trace file is closed and preserved when the backup process takes over.

`TRAPPORT` *`trap-port-number`*

specifies the trap port to send traps from the agent. If you do not specify this option, the default value for the trap port will remain 162.

Special Considerations for Users of Logical Network Partitions (LNPs)

Like NonStop TCP/IP, NonStop TCP/IPv6 allows you to restrict the connectivity of applications to particular subnets. However, TCP/IPv6 allows this through the creation of Logical Network Partitions (LNPs). The *TCP/IPv6 Configuration and Management Manual* gives a detailed explanation of how to create and use LNPs.

Each LNP is associated with a separate TCP6SAM process. Thus, each LNP on which you want to provide SNMP services requires that you run a separate instance of the SNMP Agent.

To provide SNMP services on one or more LNPs, you must issue a RUN command from a different subvolume for each of those LNPs.

For example, from within one subvolume, specify:

```
RUN $SYSTEM.SYSTEM.SNMPAGT / NAME $znm0 / &
TCP^IP^PROCESS^NAME $zb03A
```

From within another, specify:

```
RUN $SYSTEM.SYSTEM.SNMPAGT / NAME $znm1 / &
TCP^IP^PROCESS^NAME $zb01a
```

Note. If your RUN command does not specify a TCP6SAM process associated with a particular LNP, an SMNP Agent has access only to the subnets comprising the Default Partition.

Startup Parameter Summary

Table 2-2. Summary of Startup Parameters (page 1 of 2)

Startup Parameter	**What the Parameter Defines**	**Default Behavior**
backup-cpu-number	The backup process CPU for a persistent agent process pair.	No parameter value is assigned, and no backup process is created.
COLD | WARM	Which SNMPCTL file to use while running.	If SNMPCTL exists in the run subvolume, that file is used. Otherwise, the SNMP agent creates an SNMPCTL file using the default configuration values.
COLLECTOR [$*alternate-collector-process*]	The EMS collector to which to send event messages.	$0 on the local node.
DATAPAGES *pages* [E[XTENSIBLE]]	The amount of storage to allocate for use as general dynamic memory.	240 2048-byte pages, extensible.
MAXOPENERS *number-of-openers*	The maximum number of openers the agent can support simultaneously.	20.
PORT *request-port-number*	The port number to monitor for manager requests. If a port number greater than 1023 is used, the SNMP agent need not run as a process associated with the super user group (255, *n*).	161, the SNMP standard port for receiving requests.
SUBAGENT^TIMEOUT *timeout-seconds*	The number of seconds the agent should wait before canceling a message to a subagent.	3 seconds.
SWAPVOL $*disk-name*	The disk to use for swapping extended memory.	The volume used for swapping virtual data.

Table 2-2. Summary of Startup Parameters (page 2 of 2)

Startup Parameter	What the Parameter Defines	Default Behavior
TCPIP^PROCESS^NAME [*/node*.]$*tcpip-process*	The default TCP/IP process to be used when request/response connections and trap destinations are defined for the SNMP agent. For Parallel Library TCP/IP, this is the TCPSAM process. See the *TCP/IP (Parallel Library) Configuration and Management Manual.* For NonStop TCP/IPv6, this is the TCP6SAM process. See the *TCP/IPV6 Configuration and Management Manual*.	$ZTC0 on the local node.
TRACE	Enables tracing of the backup agent process.	No parameter value is assigned, and no backup process tracing is enabled.
TRAPPORT *trap-port-number*	The trap port number to send traps to manager.	162, the SNMP standard port for sending traps.

WARM | COLD Custom Configuration Parameters for the RUN Command

The startup parameters COLD and WARM allow you to customize how the SNMP agent functions. For details about COLD and WARM parameters see page 2-9.

If you start the SNMP agent using the WARM startup parameter, a default operations environment (from RUN command parameters) is automatically configured.

If you start the SNMP agent using the COLD startup parameter, you must explicitly configure all aspects of the operations environment.

The SNMPCTL File and the WARM Startup Parameter

When you start the SNMP agent using the WARM startup parameter (the default) and no SNMPCTL file exists in the run subvolume (the first time you start an SNMP agent process), the SNMP agent creates an SNMPCTL file and fills it with default RUN command configuration values. You can then modify the SNMP file using RUN command parameters, SCF, PARAM statements, or SNMP requests.

WARM is the default value.

If an SNMPCTL file already exists in the run subvolume (when you stop and restart an SNMP agent process), the SNMP agent uses the existing configuration values.

Note. To stop and restart an SNMP agent using the default configuration, you must stop the SNMP agent, purge the existing SNMPCTL file, and then restart the SNMP agent process using the WARM option.

The SNMPCTL File and the COLD Startup Parameter

When you start the SNMP agent using the COLD startup parameter and no SNMPCTL file exists in the run subvolume, the SNMP agent creates an empty SNMPCTL file. You must explicitly set each configuration value to be stored in the SNMPCTL file using the RUN command parameters.

When you start the SNMP agent specifying the COLD startup parameter and an SNMPCTL file already exists in the run subvolume, the SNMP agent purges the existing SNMPCTL file and creates an empty SNMPCTL file. You must explicitly set each configuration value to be stored in the empty SNMPCTL file using the RUN command. Values set when the COLD parameter is specified persist only as long as the agent process is running.

Here is an example of the RUN command in which the COLD parameter is specified:

```
RUN SNMPAGT /NAME $ZSNMP, NOWAIT/ COLD
```

Use SCF or SNMP to configure all components of the SNMP agent. (See Part II, SCF for the SNMP Agent, for more information on SCF for SNMP.)

COLD and WARM Configuration Scenarios.

COLD/WARM Startup Parameter	First Time SNMP Agent Started?	How Is the SNMP Agent Configured?
COLD	Yes/No	The SNMP agent creates an SNMPCTL file after purging any existing file by that name in the run subvolume. The SNMPCTL file must then be explicitly configured.
WARM	Yes	No SNMPCTL file exists in the run subvolume. The SNMP agent creates an SNMPCTL file in the run subvolume and fills it with default values.
WARM	No	An SNMPCTL file exists in the run subvolume. The SNMP agent opens the SNMPCTL file and is configured with the values specified in the SNMCTL file.

The SNMPCTL File and TRAPPORT

The SNMPCTL file created using earlier versions of the SNMP agent is not compatible with the T9576G06 version. A workaround is to purge the old SNMPCTL file.

PARAM Statements' Custom Configuration for the RUN Command

You can use TACL PARAM statements to set startup parameters prior to issuing the RUN command. Set the parameters one at a time, using the syntax described in RUN command startup-parameter on page 2-8, and precede the parameters with the keyword PARAM.

For example, these PARAM statements

```
PARAM DATAPAGES 4000 E
PARAM TRACE
PARAM TRAPPORT 2000
SNMPAGT /NAME $ZSNMP, NOWAIT, CPU 4/ 6
```

are equivalent to the following RUN command:

```
SNMPAGT /NAME $ZSNMP, NOWAIT, CPU 4/ 6, &
DATAPAGES 4000 E, TRACE, TRAPPORT 2000
```

Using SCF to Configure Components of the SNMP Agent Environment

A third way to configure and manage the SNMP agent environment is with SCF. The following five components of the SNMP agent can be configured using SCF.

Refer to Part II, SCF for the SNMP Agent for complete information on SCF commands for SNMP.

SNMP Agent Process

The SCF PROCESS object represents the SNMP agent process. The only attribute you can manage through the SCF interface is the EMS collector to which the SNMP agent process sends event messages it generates.

Security

The SCF PROFILE object defines an entry in an authentication table that the SNMP agent consults to determine whether to accept or reject incoming requests from SNMP managers that communicate through TCP/IP.

Each SCF PROFILE object corresponds to a row in the SNMP agent's private zagInProfileTable.

Request/Response Connections

The SCF ENDPOINT object identifies an Internet address and TCP/IP process to be used for the receiving and sending of SNMP request/response messages through TCP/IP.

Each SCF ENDPOINT object corresponds to a row in the SNMP agent’s private zagInEndpointTable.

Trap Connections

The SCF TRAPDEST object identifies an Internet address and TCP/IP process to be used for the sending of trap messages.

Each SCF TRAPDEST object corresponds to a row in the SNMP agent’s private zagInTrapdestTable.

Trap Port

The object trap port can't be configured or managed through SCF.

Configuring the SNMP Agent Through SNMP Requests

You can configure and manage the following components of the SNMP agent environment through SNMP requests.

Request/Response Connections

The SNMP agent’s zagInEndpoint group objects define the Internet addresses and TCP/IP processes to be used for the receiving and sending of request/response messages.

Trap Connections

The SNMP agent’s zagInTrapdest group objects identify the Internet addresses and TCP/IP processes to be used for the sending of trap messages. Trap ports can also be used.

Authentication Table Entries

The SNMP agent’s zagInProfile group objects define the authentication table entries to be used for authenticating requests received from SNMP managers that communicate with the SNMP agent through TCP/IP.

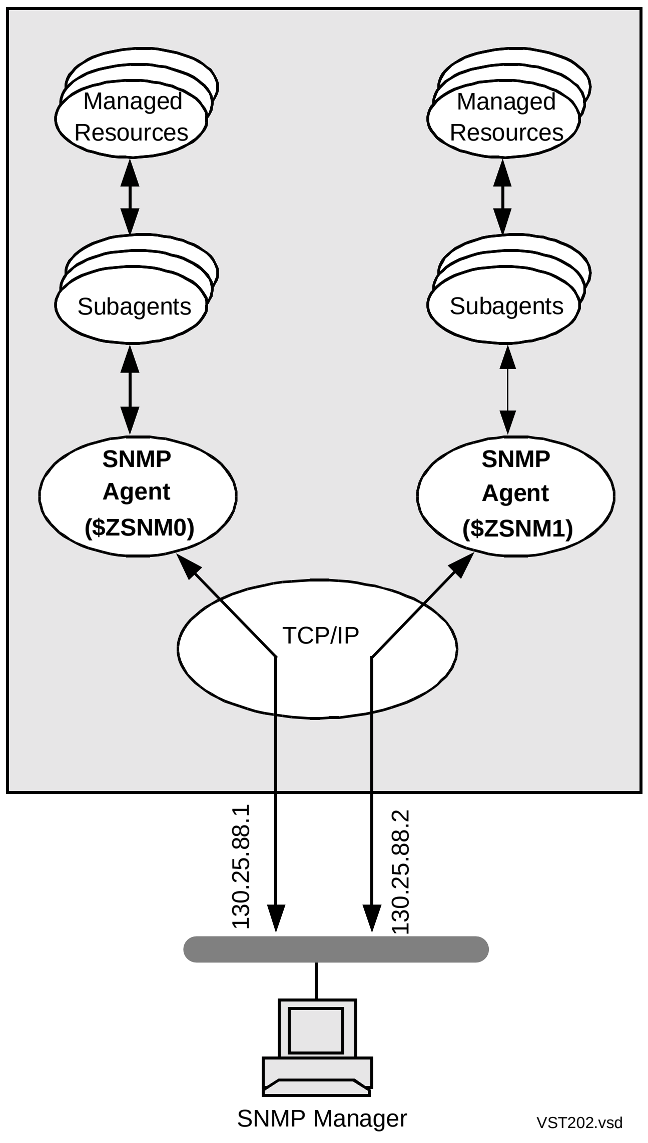
Managing Configuration Definitions Through SNMP

You can manage the SNMP agent configuration by manipulating rows in the SNMP agent’s private MIB tables, as follows:

- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent’s private MIB zagInEndpointTable to manage the TCP/IP connection points that the SNMP agent uses for request/response messages. Each row in the zagInEndpointTable corresponds to an SCF ENDPOINT object.
- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent’s private MIB zagInTrapdestTable to manage the destinations to which the SNMP agent sends traps. Each row in the zagInTrapdestTable corresponds to an SCF TRAPDEST object. However, the object zagInTdPort cannot be viewed or managed through SCF, only from the manager.
- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent’s private MIB zagInProfileTable to manage the authentication table entries that the SNMP agent uses for accepting or rejecting request messages received through TCP/IP. Each row in the zagInProfileTable corresponds to an SCF PROFILE object.

Single-Node and Multiple-Node Scenarios

Figure 2-2. Running Multiple SNMP Agents on a Single Node



In Figure 2-2, two SNMP agents are running on the same node. To run multiple agents on a single node, issue the following two commands. Issue the first command from one subvolume and the second command from a different subvolume.

- From one subvolume:

  ```
  RUN $SYSTEM.SYSTEM.SNMPAGT /NAME $ZSNM0, NOWAIT/
  ```

- From a different subvolume:

  ```
  RUN $SYSTEM.SYSTEM.SNMPAGT /NAME $ZSNM1, NOWAIT/
  ```

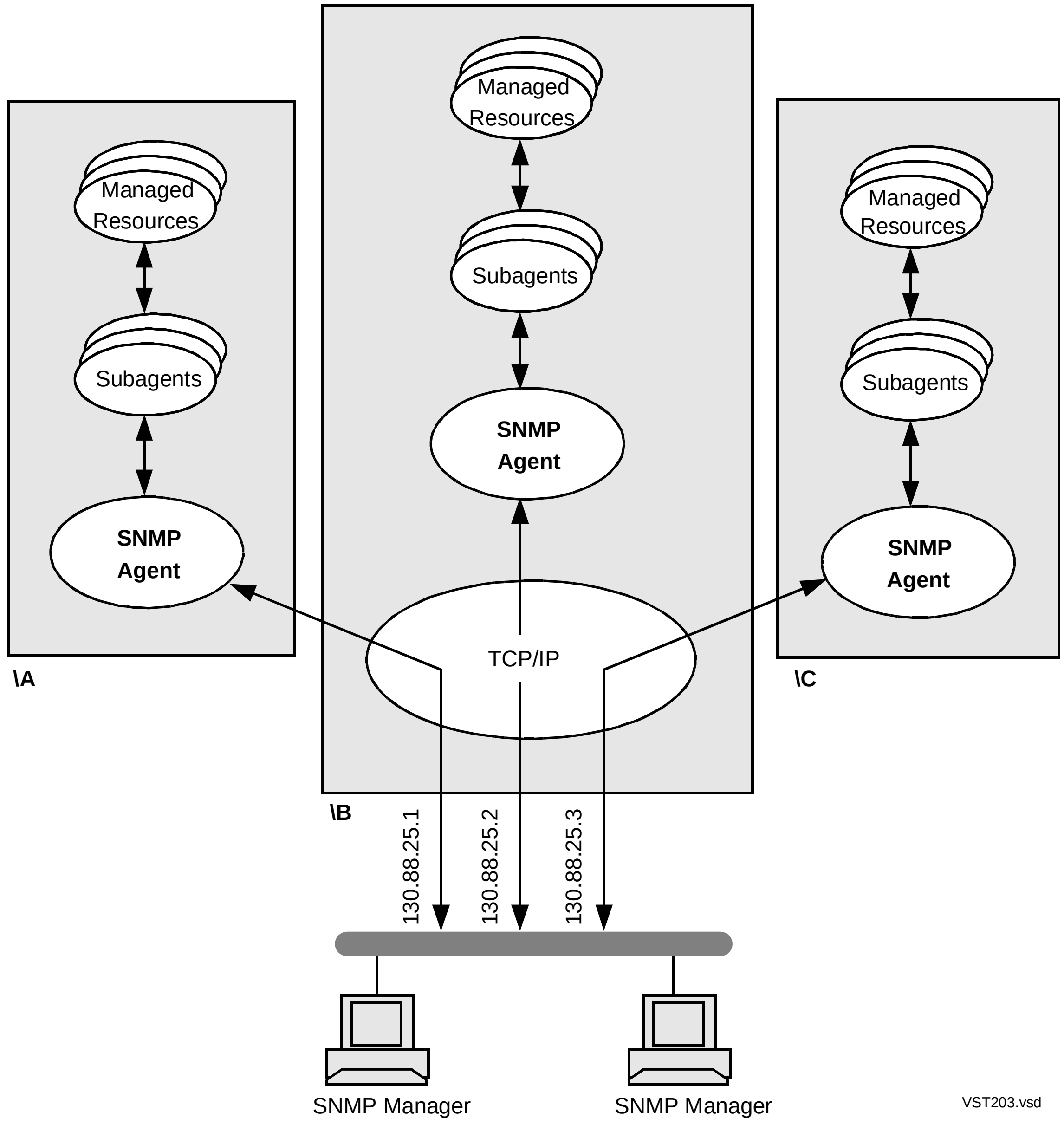
△ **Caution.** Each SNMP agent needs exclusive access to an SNMPCTL file. The SNMPCTL file used is the file that resides in the subvolume from which the SNMP agent is started. Therefore, each SNMP agent running on the same node must be started from the subvolume in which the SNMPCTL file you want the agent to use resides.

If the standard SNMP port for receiving requests is used (port 161), only one agent process can use the TCP/IP subnet at a time. Distinct subnets ensure that each message is routed to the appropriate target SNMP agent. Therefore, the type of configuration illustrated in Figure 2-2 is possible only if at least one TCP/IP subnet is available for each SNMP agent running. Each SNMP agent must be reached through a separate Internet address. You can configure both NonStop TCP/IP and NonStop TCP/IPv6 to have distinct subnets. (To configure distinct subnets for NonStopTCP/IP/v6, use the Logical Network Partition feature described in the *TCP/IPv6 Configuration and Management Manua*l.)

In the Parallel Library TCP/IP environment, in order to bind each agent process to a subnet, you must alter the ENDPOINT object and change the HOSTADDR attribute to the IP address you want the agent bound to. See Remote Connections on page 2-37.

Note. H-series RVUs do not provide support for Parallel Library TCP/IP.

Figure 2-3. Running SNMP Agents on Multiple Nodes

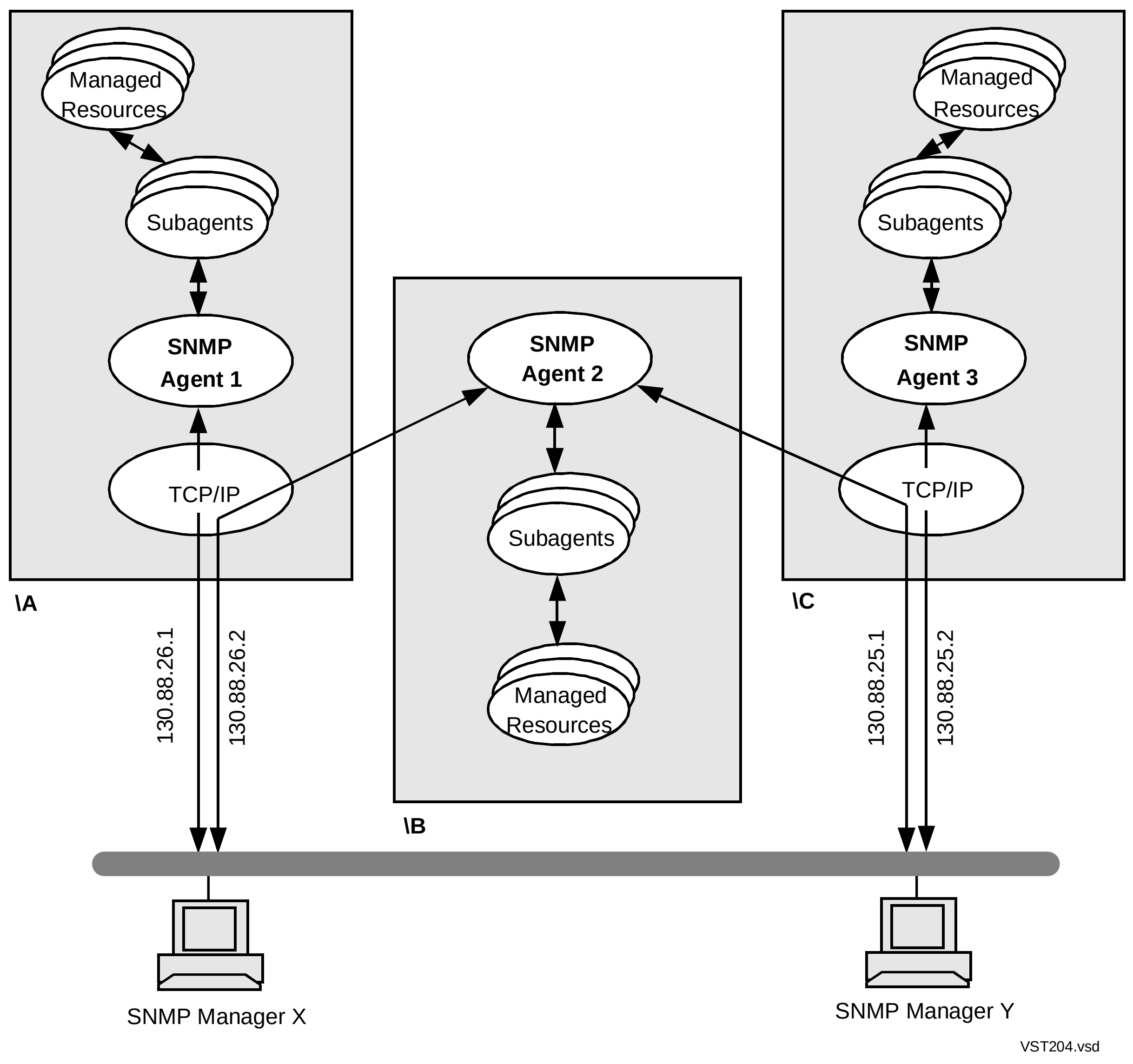


In Figure 2-3, SNMP agents on three different nodes process requests through the same TCP/IP process. This type of configuration is possible only if there is at least one TCP/IP subnet available for each SNMP agent running because a TCP/IP subnet can be used by only one agent process at a time. Each SNMP agent must be reached through a separate Internet address.

Note. YParallel Library TCP/IP and NonStop TCP/IPv6 do not support this configuration.

A disadvantage of this scenario is that \A and \C cannot be managed if \B fails.

Figure 2-4. Managing Overlapping Sets of Resources



In Figure 2-4, two TCP/IP SNMP managers manage an overlapping set of resources. SNMP manager X monitors resources through SNMP agents 1 and 2. SNMP manager Y monitors resources through SNMP agents 3 and 2. SNMP agent 2 processes requests from both managers. The advantage of this scenario is that if \A or \C fails, resources accessed through SNMP agent 2 and its subagents can still be monitored.

Starting Multiple SNMP Agents on Each Node

At least one TCP/IP subnet is required to handle communication between a NonStop agent process and an SNMP manager that is using TCP/IP. To find out whether at least one subnet is available, issue the SCF INFO and STATUS commands against the TCP/IP subsystem SUBNET object, for example:

```
INFO SUBNET $ZTC*.*

TCPIP Info SUBNET \EAST.$ZTC0.*

Name              Devicename        *IPADDRESS        TYPE      *SUBNETMASK

#LOOP0            \NOSYS.$NOIOP      127.0.0.1        LOOP-BACK  %HFF000000
#EN1              \EAST.LAN1         130.252.60.234   ETHERNET   %HFFFFF000

STATUS SUBNET $ZTC0.#EN1

TCPIP Status SUBNET \EAST.$ZTC0.#EN1

Name              Status

#EN1              STARTED
```

The NonStop agent can use a TCP/IP process on any node to communicate with SNMP managers.

When running more than one SNMP agent on the same node, and when the agent/manager transport protocol is not restricted to NonStop Kernel IPC calls, follow these guidelines:

- Start each SNMP agent from a different subvolume. Each SNMP agent uses the SNMPCTL file on the subvolume from which the SNMP agent was run. Running each SNMP agent from a different subvolume ensures that each SNMP agent maintains its own distinct configuration.
- Start only one SNMP agent process using the built-in defaults. Start additional SNMP agents in one of two ways:
    - Use the COLD startup parameter. Then configure at least one ENDPOINT object through SCF and complete the configuration through SCF or SNMP to avoid configuration conflicts.
    - Use the TCPIP^PROCESS^NAME startup parameter to specify a different TCP/IP process for the default response/request connection point. Note that:

      For Parallel Library TCP/IP, the TCPIP^PROCESS^NAME is the name of a TCPSAM process.

      Note. H-series RVUs do not provide support for Parallel Library TCP/IP.

      See the *TCP/IP (Parallel Library) Configuration and Management Manual* for information about how to find the name of a TCPSAM process.

      For NonStop TCP/IPv6,the TCPIP^PROCESS^NAME is the name of a TCP6SAM process.

See the *TCP/IPv6 Configuration and Management Manual* for information about how to find the name of a TCP6SAM process.

See also The TCP/IP Subagent and Its Managed Resources on page 8-4.

The following example starts two SNMP agents on the same HP node. For Parallel Library TCP/IP and NonStop TCP/IPv6, the TCP/IP process name is the TCPSAM/TCP6SAM name.

```
$SYSTEM TCON 5> RUN $SYSTEM.SYSTEM.SNMPAGT/NAME $ZSNM0, NOWAIT/
$SYSTEM TCON 6> VOLUME $SYSTEM.TCON1
$SYSTEM TCON1 7> RUN $SYSTEM.SYSTEM.SNMPAGT/NAME $ZSNM1, NOWAIT/&
$SYSTEM TCON1 7> TCPIP^PROCESS^NAME $ZTC1
$SYSTEM TCON1 12> SCF
SCF - T9082D20 - (01JUN93) (18MAY93) - 01/09/96 12:38:06 System \EAST
Copyright Tandem Computers Incorporated 1986 - 1996

1-> INFO PROCESS $ZSNM0, SUB ALL

SNMP Info PROCESS
Name            *EMS Collector
$ZSNM0          \EAST.$0

SNMP Info PROFILE

Name             *Access         *Hostaddr                *Community
$ZSNM0.#DEFAULT  READONLY        0.0.0.0                  public

SNMP Info ENDPOINT

Name             *Network        *Hostaddr
$ZSNM0.#DEFAULT  \EAST.$ZTC0     0.0.0.0
2-> INFO PROCESS $ZSNM1, SUB ALL

SNMP Info PROCESS

Name            *EMS Collector
$ZSNM1          \EAST.$0

SNMP Info PROFILE

Name             *Access         *Hostaddr                *Community
$ZSNM1.#DEFAULT  READONLY        0.0.0.0                  public

SNMP Info ENDPOINT

Name             *Network        *Hostaddr
$ZSNM1.#DEFAULT  \EAST.$ZTC1     0.0.0.0
```

Stopping the SNMP Agent

To stop an SNMP agent process, issue the STOP command from a TACL prompt, naming the agent process you want to stop:

```
STOP $ZSNMP
```

Get the name of the SNMP agent process using either the SCF STATUS PROCESS $ZSNMP command or the TACL STATUS command.

```
STATUS PROCESS $ZSNMP

SNMP Status PROCESS $ZSNMP

Name                Status          Trace       Trace File

$ZSNMP              STARTED         OFF         N/A
```

```
status *, prog $system.system.snmpagt

Process         Pri PFR %WT Userid  Program file
Hometerm

$DMA      5,85  144     011 255,209 $SYSTEM.SYSTEM.SNMPAGT
$T304.#TRM6
```

Configuring the EMS Collector

Configuring the EMS collector requires the following tasks:

- Direct the SNMP agent process to send its events to a specific EMS collector
- Configure and manage the authentication mechanism by which the SNMP agent accepts or rejects requests from SNMP managers through TCP/IP
- Configure and manage TCP/IP request/response connections between the SNMP agent and SNMP managers
- Configure and manage the definitions of destinations to which the SNMP agent sends traps

When you start the SNMP agent using the default startup parameters, the SNMP agent sends its events to the EMS collector $0. To use an EMS collector other than $0:

- Include the COLLECTOR startup parameter when you start the SNMP agent.
- Use the SCF ALTER command to change the EMSCOLL attribute of the PROCESS object.

Note. No MIB object exists in the SNMP agent’s private zagInProcess group for specifying the EMS collector.

Collector process information is not stored in the SNMPCTL file. When you stop and restart an SNMP agent process, you must respecify the nondefault collector process to which you want the SNMP agent to send its events.

Using the COLLECTOR Startup Parameter

To cause both primary and backup agent processes to send events to an EMS collector other than $0, include the COLLECTOR startup parameter when you start the SNMP agent. For example:

```
RUN SNMPAGT/NAME $ZSNMP, NOWAIT/ COLLECTOR $ACOL
```

Although HP does not recommend doing so, you can suppress output of SNMP agent events altogether by including the COLLECTOR startup parameter with no argument:

```
RUN SNMPAGT/NAME $ZSNMP, NOWAIT/ COLLECTOR
```

This approach is the only way to suppress event logging. You cannot suppress event logging through SCF.

For more information on the COLLECTOR startup parameter, refer to COLLECTOR [$alternate-collector-process] on page 2-9.

Using the SCF ALTER PROCESS Command

You can also specify that the agent process send its events to an alternate collector by issuing an SCF ALTER command against the PROCESS object and specifying a new value for the EMSCOLL attribute. For example:

```
ALTER PROCESS $ZSNMP, EMSCOLL $ACOL
```

Configuring Security

Before the SNMP agent processes an SNMP request, the SNMP agent authenticates the request to determine whether the SNMP manager is qualified to perform the operation in the request.   The request authentication rules implemented by the SNMP agent depend on whether the SNMP manager is communicating through NonStop Kernel IPC calls or through the TCP/IP UDP transport protocol.

Authenticating Requests Received Over TCP/IP

To configure the basis on which the SNMP agent accepts or rejects requests from SNMP managers that communicate through TCP/IP, build an authentication table. Each entry in the authentication table is one instance of a PROFILE object, or one row in the SNMP agent’s private zagInProfileTable. You can configure, request information about, and manage authentication table entries in two ways:

- By issuing SNMP Set requests against the SNMP agent’s zagInProfile group objects from an SNMP manager that uses NonStop Kernel IPC calls to communicate with the SNMP agent.
- By issuing SCF commands against PROFILE objects.

To avoid extraneous authenticationFailure traps, configure security before you configure request/response connection points. Request/response connection points are represented by SCF ENDPOINT objects or by rows in the SNMP agent private zagInEndpointTable. For more information, refer to Configuring TCP/IP Request/Response Connections on page 2-34.

The Authentication Mechanism

Authentication is the procedure by which the agent process accepts or rejects requests from an SNMP manager that communicates through TCP/IP.

The SNMP agent looks at three elements of an incoming SNMP message to determine whether to process (or forward) a request:

- The **community name** portion of the community string included in the message
- The **Internet address** from which the request originated
- The **SNMP operation** that the SNMP manager wants the agent process to run or forward for running on the manager's behalf

Community Strings

Each request that an SNMP agent receives from an SNMP manager includes a community string composed of one or two discrete sections delimited by two colons (`::`) as follows:

`*community-name*[::*subagent-password*]`

The community string included in incoming requests is part of the SNMP manager's configuration and is set as described in the documentation provided by the vendor of the SNMP manager.

The SNMP agent parses the *community-name* portion of the community string. Community names that an SNMP agent recognizes are configured through SCF by the system administrator responsible for the SNMP agent.

Note. The *subagent-password* is passed to the subagent by the SNMP agent and is used by the subagent to employ additional security. For more information, see After a Request Has Been Authenticated on page 2-31.

When an SNMP agent process receives a request, the SNMP agent searches for a matching community name in the authentication table. If the SNMP agent does not find a matching community name entry in the authentication table, the request is dropped. If the snmpEnableAuthenTraps object in the SNMP group supported by the SNMP agent is set to 1, the SNMP agent also sends an authenticationFailure trap to all broadcast type trap destinations. Trap destinations are represented by SCF TRAPDEST objects or by rows in the SNMP agent's private zagInTrapdestTable. (For more information, see Configuring Trap Destinations on page 2-38.)

Internet Addresses

Each entry in the authentication table contains a single Internet address or a full wild-card address (0.0.0.0.) specification. Internet addresses are discussed in detail in the *TCP/IP Configuration and Management Manual*.

When the SNMP agent finds a community name match, the SNMP agent looks at the address from which the request originated. If the address contained in the incoming request matches the address in the authentication table entry, the SNMP agent accepts the request.

If the SNMP agent does not find a matching Internet address entry in the authentication table, the request is dropped. If configured to do so, the SNMP agent sends an authenticationFailure trap to all trap destinations.

SNMP Operations

The authentication table also describes the SNMP operations authorized for each community. Allowable SNMP operations are described by one of two access modes, specified as ACCESS attributes in SCF: READONLY and READWRITE.

- If the access mode is **READONLY**, the SNMP agent accepts GetRequest and GetNextRequest PDUs from the SNMP manager. SetRequest PDUs are dropped.
- If the access mode is **READWRITE**, the SNMP agent accepts SetRequest, GetRequest, and GetNextRequest PDUs from the SNMP manager.

If the request is for an unauthorized operation, the request is dropped. If configured to do so, the SNMP agent sends an authenticationFailure trap to all trap destinations.

If the request passes all three tests (community name, Internet address, and access mode), the SNMP agent is said to have authenticated the request.

Configuring the Authentication Table

An authentication table entry has the following attributes:

SCF		**SNMP Agent Private MIB**	
Object	**Attribute**	**Table**	**Object Within Table Row**
PROFILE	*#profile-name*	zagInProfileTable	zagInPfName
	COMMUNITY		zagInPfCommunity
	HOSTADDR		zagInPfHostAddr
	ACCESS		zagInPfAccess

- You name the object when you define it.
- COMMUNITY or zagInPfCommunity is a community name, unique among authentication table entries.
- HOSTADDR or zagInPfHostAddr is an SNMP manager Internet address.

- ACCESS or zagInPfAccess specifies the SNMP operations authorized for each community: READONLY or READWRITE.

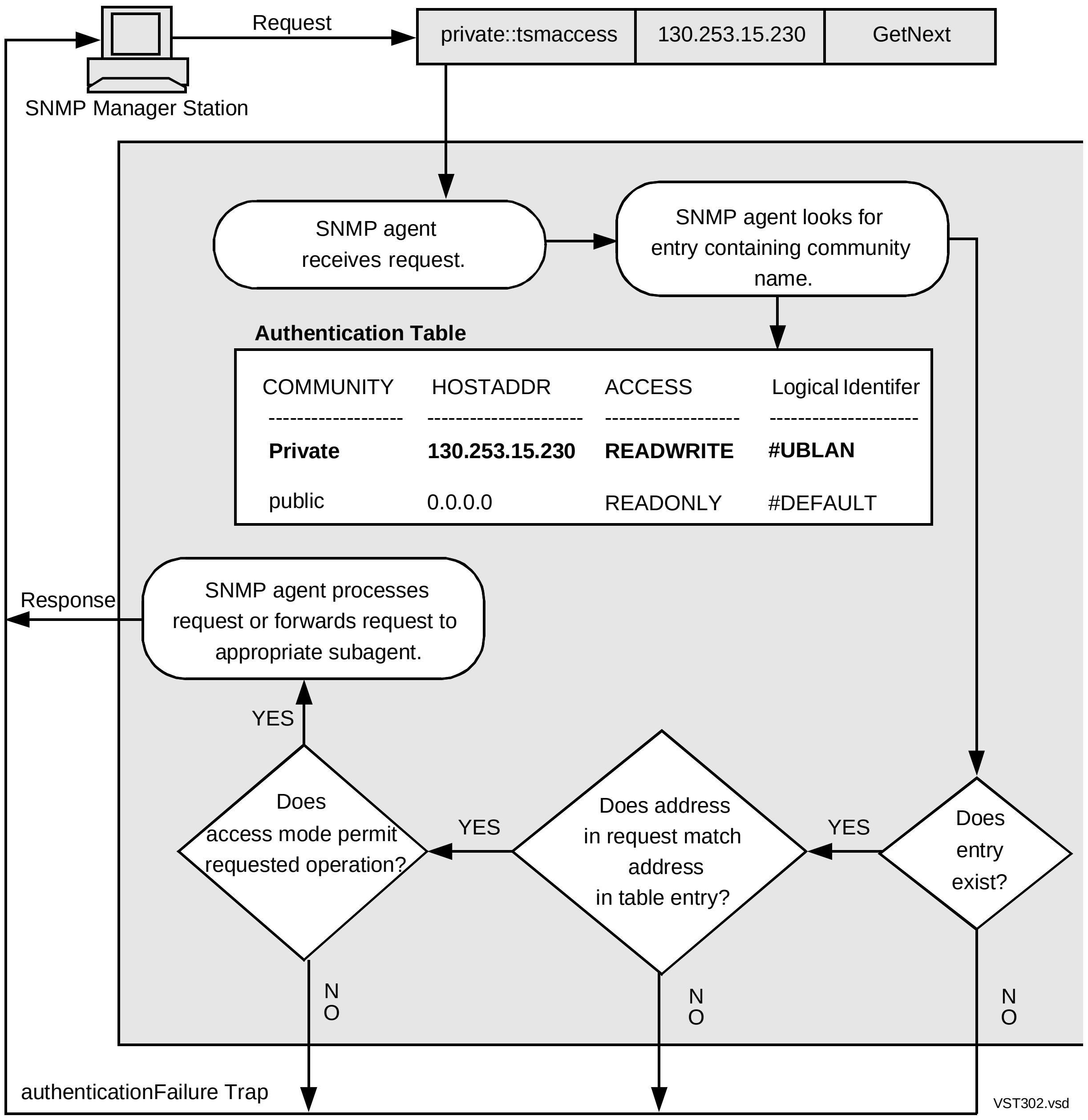
The default authentication table entry is named $*agent-process*.#DEFAULT and has the following attribute values associated with it (the SNMP agent process is $ZSNMP):

```
PROFILE $ZSNMP.#DEFAULT, COMMUNITY public
   HOSTADDR 0.0.0.0, ACCESS READONLY
```

You must add an authentication table entry for each community (other than “public”) from which the SNMP agent will be receiving SNMP requests.

Unless you stop or alter the default authentication table entry, the SNMP agent continues to accept incoming Get and GetNext requests from any SNMP manager belonging to the “public” community, regardless of other table entries you define to tighten security.

Figure 2-5. Authenticating Requests Received Through TCP/IP


Request
private::tsmaccess
130.253.15.230
GetNext
SNMP Manager Station
SNMP agent receives request.
SNMP agent looks for entry containing community name.
Authentication Table
COMMUNITY
HOSTADDR
ACCESS
Logical Identifer
Private
130.253.15.230
READWRITE
#UBLAN
public
0.0.0.0
READONLY
#DEFAULT
Response
SNMP agent processes request or forwards request to appropriate subagent.
YES
Does access mode permit requested operation?
YES
Does address in request match address in table entry?
YES
Does entry exist?
NO
NO
NO
authenticationFailure Trap
VST302.vsd

In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the “public” community, as well as Set requests sent under the “Private” community from the SNMP manager whose address is 130.252.15.230. The SCF commands that defined this nondefault PROFILE object are:

```
ASSUME PROCESS $ZSNMP
ADD PROFILE #UBLAN, COMMUNITY Private,&
ACCESS READWRITE, HOSTADDR 130.253.15.230
START PROFILE #UBLAN
```

This request would be authenticated.

Note that the trap messages shown in Figure 2-5 are generated only when:

- A message is received that fails authentication.
- The snmpEnableAuthenTraps object in the SNMP group of the SNMP agent’s MIB is set to 1.

Refer to Configuring Trap Destinations on page 2-38 for more information on traps.

Authenticating Requests Received Using the IPC Interface

Requests received using the IPC interface are authenticated as follows:

- The community name and Internet address in the request are not examined. (However, they are passed to the subagent to which the request is forwarded; the subagent can then employ additional security based on the community name and Internet address.)
- All MIB objects under the control of the SNMP agent are accessible. Requests for operations on objects in the zagInProfile group are performed only if the PAID of the SNMP manager process is compatible with the PAID of the agent process. If the agent process is associated with the super user group (user ID 255,n), the SNMP manager process must also be associated with the super user group. If the agent process is not associated with the super user group, the PAID of the SNMP manager process is irrelevant.
- Requests for MIB objects under the control of subagent are passed to subagents for processing, as described in the following subsection.

After a Request Has Been Authenticated

Authentication allows the SNMP manager access to the following SNMP agent services:

- Access to the MIB-II SNMP and System group objects, as well as the private MIB objects supported by the SNMP agent. These requests are processed by the SNMP agent.
- The forwarding of SNMP requests concerning MIB objects supported by subagents to the appropriate subagent process.

By default, authenticated manager requests forwarded to subagents are processed in accordance with the access attributes of individual MIB objects. SNMP managers can perform Get and GetNext operations on read-only MIB objects and Get, GetNext, and Set operations on read-write objects.

When the SNMP agent forwards a request to a subagent, the forwarded request includes the Internet address and community string. The subagent can use the Internet address and the *`subagent-password`* portion of the community string to employ additional security, responding to a request to access a MIB object, independent of the subagent's access attributes, only if the request contains a particular password and originates from a specific Internet address. This option, known as subagent request authentication, requires that the password appear as follows in the manager station's community string:

```
agent-community-string::subagent-password
```

Security Scenarios for SNMP Managers Using TCP/IP

Table 2-3. Security Scenarios (page 1 of 2)

Scenario	Tasks	Method (SCF Commands)
Allow an SNMP manager to retrieve information only.	Ask the SNMP administrator to send requests to the SNMP agent under the "public" community. By default, the SNMP agent accepts Get and GetNext requests from all members of the "public" community.	Ask the SNMP administrator.
Allow an SNMP manager to set values and retrieve information.	Configure an authentication table entry for the host address of the SNMP manager, give the authentication table a unique community name, and assign the table READWRITE access. Then activate the entry and inform the SNMP administrator of the community name that must be present in requests sent to the SNMP agent. The SNMP agent accepts Get, GetNext, and Set requests that contain the new community name.	ADD PROFILE START PROFILE
Allow an SNMP manager to retrieve information only.	Ask the SNMP administrator to send requests to the SNMP agent under the "public" community. By default, the SNMP agent accepts Get and GetNext requests from all members of the "public" community.	Ask the SNMP administrator.

Table 2-3. Security Scenarios (page 2 of 2)

Scenario	Tasks	Method (SCF Commands)
	Alternatively, you can simply alter the default PROFILE entry and assign READWRITE access to the “public” community. In this case, you should also alter the default host address to make it more restrictive. (Otherwise, all SNMP managers sending requests under the “public” community can alter variables.) Then activate the PROFILE entry.	STOP PROFILE ALTER PROFILE START PROFILE
Revoke Set privileges from an SNMP manager, but allow it to retrieve information.	Deactivate the authentication table entry for the community under which the SNMP manager has been sending requests. Alter the SNMP manager’s access mode to READONLY. Then reactivate the authentication table entry.	STOP PROFILE ALTER PROFILE START PROFILE
	The READONLY access will apply to all managers using the community name. If READONLY access is not acceptable, ask the other SNMP administrators to reconfigure their managers to send requests under a different community name.	
Revoke all access privileges from an SNMP manager.	Deactivate the authentication table entry for the community under which the SNMP manager has been sending requests.	STOP PROFILE
	The SNMP agent accepts no requests from the SNMP manager.	
	If more than one SNMP manager sends requests under that community, alter the host address to exclude the particular manager. Then activate the entry.	STOP PROFILE ALTER PROFILE START PROFILE
	If you cannot alter the host address to exclude one manager, ask the other SNMP administrators to reconfigure their managers to send requests under a different community name. Then add an authentication table entry for the new community and activate the entry.	STOP PROFILE ADD PROFILE START PROFILE

Configuring TCP/IP Request/Response Connections

The TCP/IP process handling the communication between the SNMP agent process and SNMP managers is represented by a request/response connection definition. You specify and request information about the request/response connections by issuing:

- SCF commands against ENDPOINT objects.
- SNMP Set requests against the SNMP agent's zagInEndpointTable entry objects, as described in Section 3, MIBs Supported by the SNMP Agent.

A TCP/IP request/response connection definition has the following attributes:

SCF		**SNMP Agent Private MIB**	
Object	**Attribute**	**Table**	**Object Within Table Row**
ENDPOINT	`#endpoint-name`	zagInEndpointTable	zagInEpName
	HOSTADDR		zagInEpHostAddr
	NETWORK		zagInEpNetwork

- You name the object when you define it.
- HOSTADDR or zagInEpHostAddr is the Internet address by which the SNMP agent can be addressed. The Internet address is the TCP/IP subnet address through which the SNMP agent receives requests from SNMP managers.
- NETWORK or zagInEpNetwork is the TCP/IP process that handles request/response messages.

The default TCP/IP request/response connection definition specifies that the SNMP agent receives SNMP requests and returns responses through any available subnet associated with TCP/IP process $ZTC0 on the node on which the agent process is running.

You can specify that a different TCP/IP process be used in the default configuration by including a TCPIP^PROCESS^NAME startup parameter in the RUN command when you start the SNMP agent process.

Using the default request/response connection definition or passing a TCP/IP process name at startup works for an SNMP agent process that is not sharing the same TCP/IP subsystem with other SNMP agent processes. However, when multiple SNMP agent processes use the same TCP/IP subsystem to communicate with SNMP managers, you must ensure that each SNMP agent process has a distinct subnet available for its own use or that each SNMP agent process has been configured at startup to use a unique port. Configuring distinct subnets ensures that each message is routed to the appropriate target SNMP agent.

Single-Agent Connections

When only one SNMP agent uses a TCP/IP process, defining the request/response connection is straightforward. Use the default host address value and ensure that at least one subnet is available to handle communication with the SNMP managers. You can issue SCF INFO and STATUS commands against the TCP/IP subsystem SUBNET object to find out about the availability of subnets. For example:

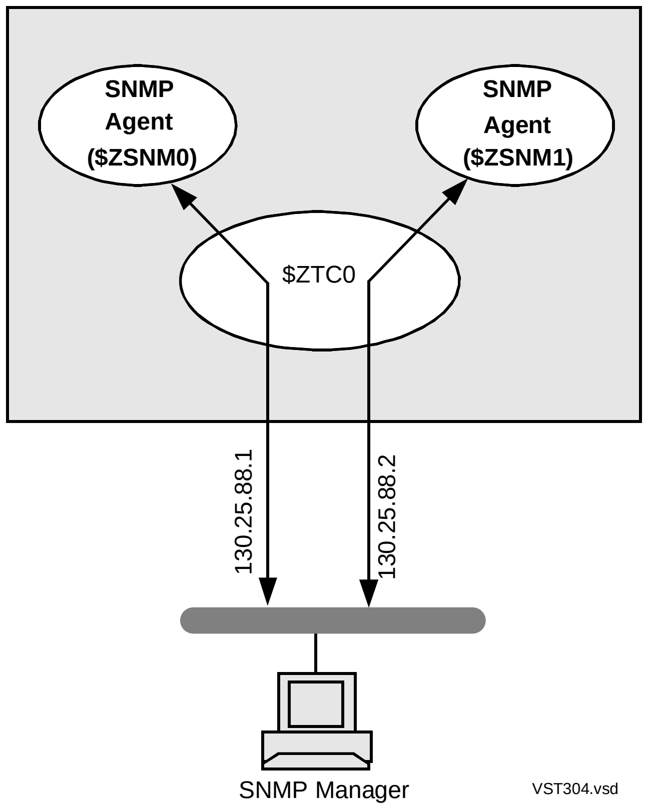
```
info subnet $ztc0.*

TCPIP Info SUBNET \WEST.$ZTC0.*

Name           Devicename     *IPADDRESS        TYPE     *SUBNETMASK

#LOOP0         \NOSYS.$NOIOP   127.0.0.1        LOOP-BACK %HFF000000
#SUBNET1       \WEST.LAN1      130.252.88.1     ETHERNET  %HFFFF0000
#SUBNET2       \WEST.LAN2      130.252.87.1     ETHERNET  %HFFFF0000

status subnet $ztc0.#subnet2

TCPIP Status SUBNET \WEST.$ZTC0.#SUBNET2

Name               Status

#SUBNET2           STARTED
```

Multiple-Agent Connections

If the same TCP/IP subsystem is handling SNMP agent/manager communication for more than one SNMP agent process, each SNMP agent must either be reached through a separate Internet address or must be configured at startup to use a unique port. You must configure nonoverlapping host address values in request/response connection definitions.

If a request/response connection has been defined to use the full wild-card specification (0.0.0.0) to refer to all Internet addresses associated with a particular TCP/IP process, and you want to start another SNMP agent process to be reached through the same TCP/IP process, you must configure nonoverlapping host address values in the two SNMP agents' request/response connection definitions.

Figure 2-6. Multiple Local SNMP Agents, Single Host Request/Response Connections



In Figure 2-6 a successful configuration involving two SNMP agents uses the same TCP/IP process for request/response communications. In this configuration, the following is assumed:

- $ZSNM0 and $ZSNM1 were started from different subvolumes with the built-in defaults.
- The default request/response connection definition associated using each SNMP agent was altered as illustrated in the following SCF commands:

```
STOP ENDPOINT $ZSNM1.#DEFAULT
STOP ENDPOINT $ZSNM0.#DEFAULT
ASSUME PROCESS $ZSNM1
ALTER ENDPOINT #DEFAULT, HOSTADDR 130.25.88.2
START ENDPOINT #DEFAULT

ASSUME PROCESS $ZSNM0
ALTER ENDPOINT #DEFAULT, HOSTADDR 130.25.88.1
START ENDPOINT #DEFAULT
```

Each SNMP agent now has a distinct subnet to handle communication with SNMP managers.

Remote Connections

The TCP/IP process used for communicating with SNMP managers does not have to be on the same node as the agent process. As in the scenario featuring multiple SNMP agents sharing a TCP/IP process on one host (Figure 2-6), you need to ensure that every local and remote agent process sharing the TCP/IP process uses a unique subnet address. In Parallel Library TCP/IP, the agents are not associated with TCP/IP processes but rather just with the IP addresses. This procedure of altering the ENDPOINT is the method by which you bind the agent to unique IP addresses in the Parallel Library TCP/IP process.

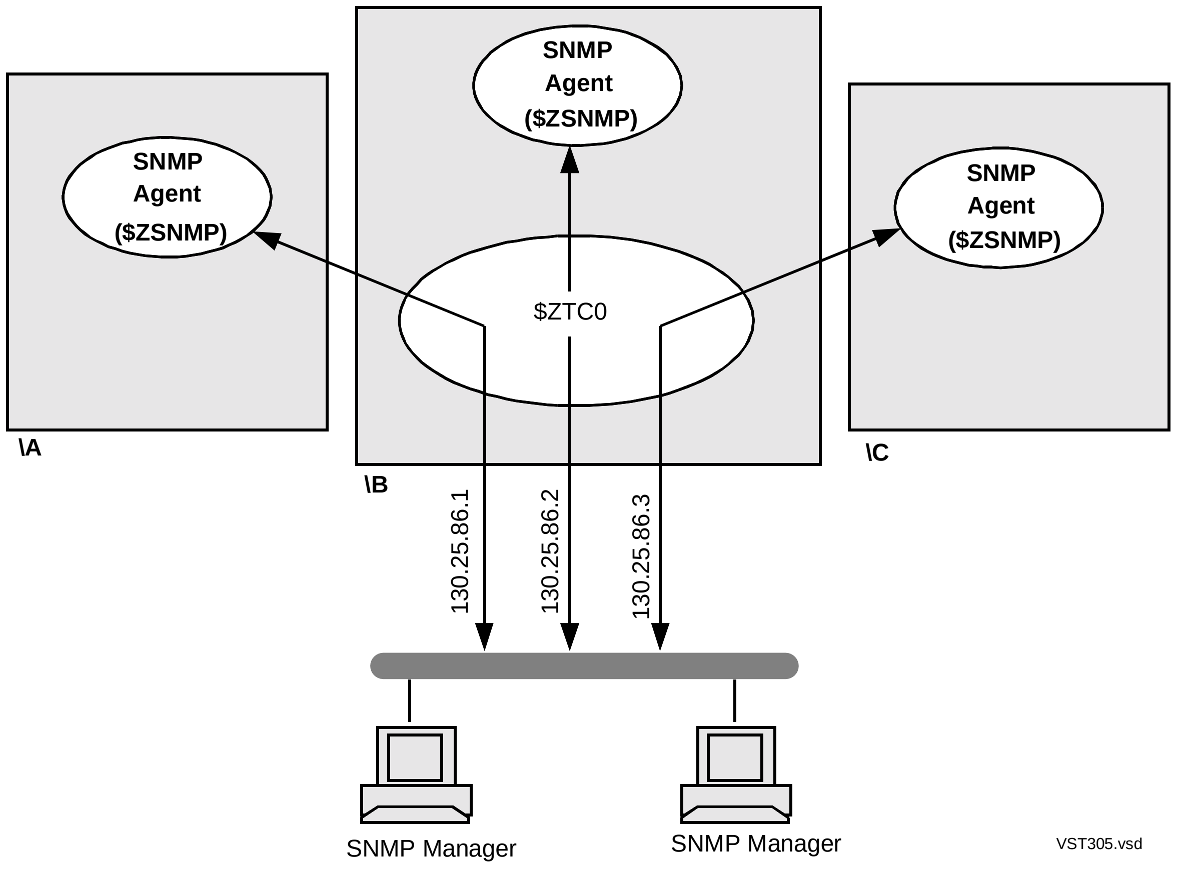
Note. H-series RVUs do not provide support for Parallel Library TCP/IP.

In Figure 2-7, agent processes on three nodes are started using the built-in defaults. Later, the default request/response connection definitions are altered for all three SNMP agents. The following example illustrates altering connection definitions using SCF:

```
SYSTEM \B
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, HOSTADDR 130.25.86.2
START #DEFAULT

SYSTEM \A
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.1
START #DEFAULT

SYSTEM \C
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.3
START #DEFAULT
```

Note that the ENDPOINT definition for the SNMP agent on node \B uses the default NETWORK attribute value, so this attribute need not be included in the ALTER command.

Figure 2-7. Multiple Remote SNMP Agents, Single Host Request/Response Connections



Configuring Trap Destinations

Each SNMP manager to which the SNMP agent process sends trap messages is represented by a trap destination definition. You specify and request information about trap destinations by issuing:

- SCF commands against TRAPDEST objects.
- SNMP Set requests against the SNMP agent's zagInTrapdestTable entry objects, as described in Section 3, MIBs Supported by the SNMP Agent.

A trap destination definition has the following attributes:

SCF		SNMP Agent Private MIB	
Object	**Attribute**	**Table**	**Object Within Table Row**
TRAPDEST	*#trapdest-name*	zagInTrapdestTable	zagInTdName
	COMMUNITY		zagInTdCommunity
	HOSTADDR		zagInTdHostAddr
	NETWORK		zagInTdNetwork
			zagInTdType
			zagInTdPort

- You name the object when you define it.
- COMMUNITY or zagInTdCommunity specifies the name included in trap messages. This attribute is assigned the value “Tandem” by default.
- HOSTADDR or zagInTdHostAddr specifies the Internet address of an SNMP manager to which traps are to be sent. You cannot use the wild-card Internet address designation “0.0.0.0” as a value for this attribute; each value must be a unique Internet address. No default value exists for this attribute.
- NETWORK or zagInTdNetwork specifies the TCP/IP process that handles the sending of trap messages. The TCP/IP process determines which subnet to use for the actual data transfer. The default value is the TCP/IP process specified in the TCPIP^PROCESS^NAME startup parameter or, if none was specified, $ZTC0 on the local node.
- The zagInTdType object specifies the type of the trap destination: directed or broadcast. Directed trap destinations receive only trap messages specifically directed to the trap destinations by the agent or subagent from which the trap originates. Broadcast trap destinations receive all trap messages except directed ones.
- The zagInTdPort object specifies the port to which traps should be sent. The object zagInTdPort cannot be viewed or managed through SCF, only from the SNMP manager.

Note. SCF does not provide for defining directed type trap destinations. All trap destination definitions created through SCF are of type broadcast. (However, trap designation definitions can be modified through SNMP to type directed.)

You can configure as many trap destination definitions as you like. SCF issues a warning if you add a definition that points to a currently defined destination. In this case, duplicate traps are sent to the same address.

Dynamically Generated Trap Destinations

If no trap destinations are defined, then for every SNMP manager communicating through TCP/IP from which the SNMP manager receives a request, the SNMP agent creates a trap destination definition. The trap port value is 162, the default value. Each dynamically generated TRAPDEST object contains the default community, network, and trap type attribute values. The host address attribute of each dynamically generated TRAPDEST object is the Internet address from which the request was received. These TRAPDEST objects are named “#DYNA0,” “#DYNA1,” and so on, and are stored in the SNMPCTL file.

For example, if the SNMP agent receives a request through TCP/IP from the manager at Internet address 130.252.86.10, the SNMP agent creates and starts the following TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA0, COMMUNITY Tandem,
  NETWORK $ZTC0, HOSTADDR 130.252.86.10
```

Then, if the SNMP agent receives a request from the SNMP manager at Internet address 155.186.130.123, the SNMP agent creates and starts another TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA1, COMMUNITY Tandem,
  NETWORK $ZTC0, HOSTADDR 155.186.130.123
```

Once you explicitly create a TRAPDEST object using the ADD command, dynamic TRAPDEST generation stops. The default trap port would be 162, however, the port cannot be seen in SCF, but can be retrieved and set to a different port using the SNMP manager.

The following is a summary of dynamic TRAPDEST object generation:

- Dynamic TRAPDEST generation is enabled when the SNMP agent is started with the WARM startup parameter and there are no TRAPDEST objects in the SNMPCTL file.
- Dynamic TRAPDEST generation is disabled when you do one of the following:
    - Use the ADD command to add a TRAPDEST object (or create one through SNMP).
    - Stop and restart the SNMP agent with the WARM option after at least one default TRAPDEST object has been generated.

      If the SNMP agent is stopped, and then restarted, using the WARM option, even if you have never explicitly added a TRAPDEST object, no new TRAPDEST objects are dynamically generated since the SNMPCTL file contains the #DYNA*n* objects created when the SNMP agent was previously running.

Forwarding Traps to Managers Communicating Through IPC

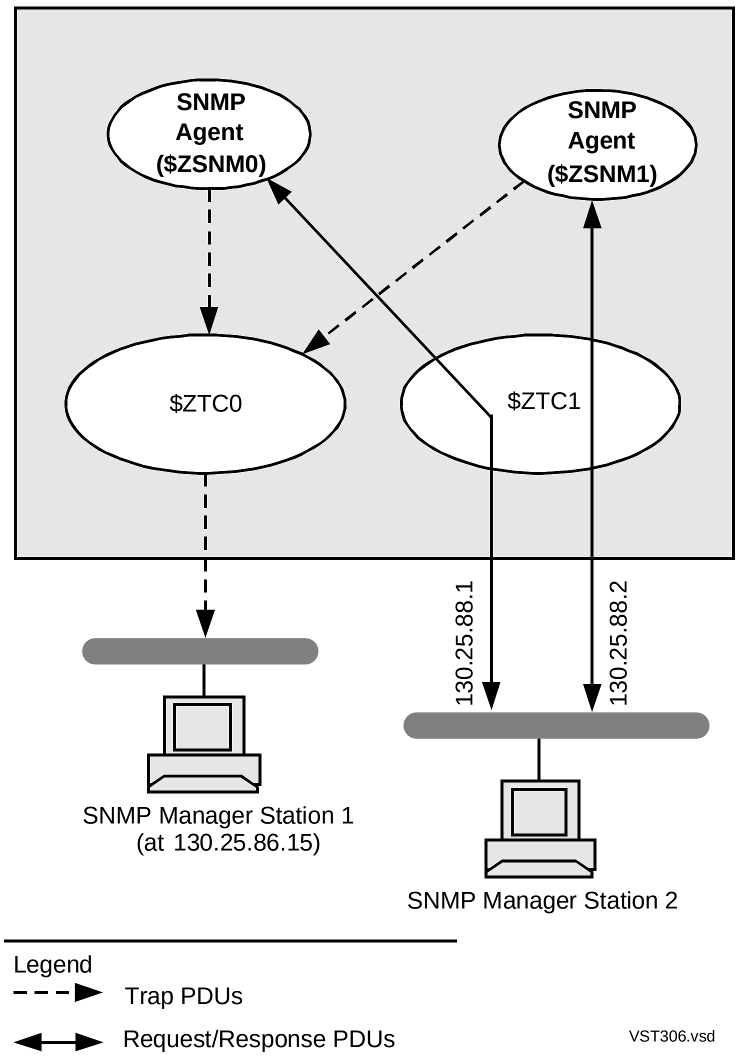
No mechanism exists for the SNMP agent to automatically forward traps through the IPC interface. To forward traps to an SNMP manager that communicates with the SNMP agent through NonStop Kernel IPC calls, configure a trap destination using a host address of 127.0.0.1 (the local loopback address, a TCP/IP convention that refers to “this” host).

Disabling the Sending of Traps

Any SNMP manager using the proper access authority can disable the sending of authenticationFailure traps to all configured trap destinations by setting the snmpEnableAuthenTraps MIB object to 2. Refer to Appendix C, Unsolicited SNMP Agent Messages, for more information.

Single-Host Connections

Figure 2-8. Multiple SNMP Agents, Single Host Connections



In Figure 2-8, two SNMP agents receive requests through one TCP/IP process and send trap messages to the same destination through a different TCP/IP process. In this configuration:

- $ZSNM0 was started using the built-in defaults.
- $ZSNM1 was started (from a different subvolume) using a value of $ZTC1 for the TCPIP^PROCESS^NAME startup parameter.

The SCF commands used to configure the ENDPOINT and TRAPDEST objects in Figure 2-8 are:

```
ASSUME PROCESS $ZSNM0
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, NETWORK $ZTC1, &
HOSTADDR 130.25.88.1
START ENDPOINT #DEFAULT
ADD TRAPDEST #STA1, HOSTADDR 130.25.86.15
START TRAPDEST #STA1
```

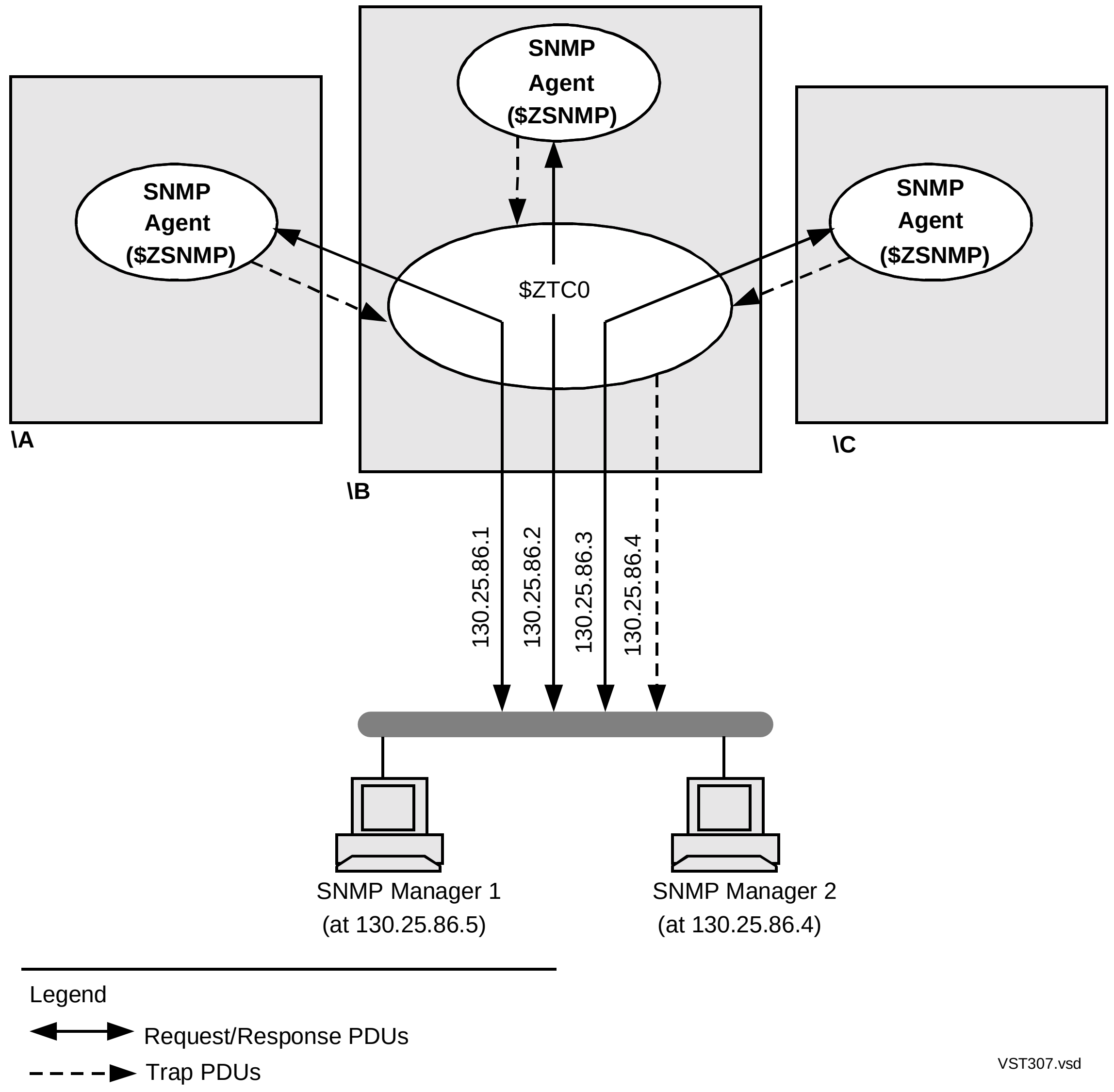
```
ASSUME PROCESS $ZSNM1
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, HOSTADDR 130.25.88.2
START ENDPOINT #DEFAULT
ADD TRAPDEST #STA1, NETWORK $ZTC0, HOSTADDR 130.25.86.15
START TRAPDEST #STA1
```

Multiple-Host Connections

The TCP/IP process handling the sending of traps does not have to be on the same node as the agent process.

Note. Parallel Library TCP/IP and Nonstop TCP/IPv6 do not support Multi-Host connections.

Figure 2-9. Multiple SNMP Agents, Multiple Host Connections



In Figure 2-9, the SNMP agents on nodes \A, \B, and \C accept requests from either SNMP manager, but only send traps to the manager whose internet address is 130.25.86.4. Following are the SCF commands used to configure connections for this scenario. Note that the SNMP agent on node \B uses the default NETWORK attribute value in its TRAPDEST object definition:

```
SYSTEM \B
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, HOSTADDR 130.25.86.2
START #DEFAULT
ADD TRAPDEST #STA2, HOSTADDR 130.25.86.4
START TRAPDEST #STA2
```

```
SYSTEM \A
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.1
START #DEFAULT
ADD TRAPDEST #STA2, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.4
START TRAPDEST #STA2
```

```
SYSTEM \C
ASSUME PROCESS $ZSNMP
STOP ENDPOINT #DEFAULT
ALTER ENDPOINT #DEFAULT, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.3
START #DEFAULT
ADD TRAPDEST #STA2, NETWORK \B.$ZTC0, &
HOSTADDR 130.25.86.4
START TRAPDEST #STA2
```

3 MIBs Supported by the SNMP Agent

The SNMP agent supports two MIB-II groups and one private MIB group. The two MIB-II groups provide information about agent and node configuration and SNMP message traffic to SNMP managers. The private MIB group is defined by HP and lets you monitor—and in some cases modify—SNMP agent configuration attributes from an SNMP manager.

Definitions for objects within the SNMP agent's private MIB are contained in the file **ZSMPMIB**. This MIB file is installed in the SNMP agent's installation subvolume (ISV) and must be incorporated into the network management application's MIB. Objects from the two groups from MIB-II are not included in the ISV. They belong to the System and SNMP groups from MIB-II known to all SNMP managers.

SNMP Agent MIB Groups

The SNMP agent supports the objects in these three MIB groups:

- **System group**. This MIB-II group provides configuration information about the SNMP agent and the node on which it resides.
- **SNMP group**. This MIB-II group provides statistical information about all the SNMP message traffic that the SNMP agent handles.
- **zagInternal group**. The zagInternal group constitutes the SNMP agent's private MIB. It contains subgroups and objects that are defined by HP to control the configuration and operation of the SNMP agent. The subgroups are **zagInProcess**, **zagInEndpoint**, **zagInProfile**, and **zagInTrapdest**.

The System and SNMP groups are identified by a check mark in the following list and are members of MIB-II:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              system (1) √
              snmp (11) √
```

Subgroups of the zagInternal group are identified by a check mark in the following list and are defined by HP:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                zsmp (155)
                                    zsmpagent (1)
                                        zagInternal (7)
                                            zagInProcess (1)√
                                            zagInEndpoint (2)√
                                            zagInProfile (3)√
                                            zagInTrapdest (4)√
```

In this section, System group objects are defined in the subsection System Group. SNMP group objects are defined in the subsection SNMP Group. The zagInternal group objects are defined in the subsection zagInternal Group.

RFC Compliance

The SNMP agent supports the MIB-II System and SNMP group objects in strict compliance with the standards specified in RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*.

Installation

The SNMP agent's MIB is implemented within the SNMP agent, so it is installed when the SNMP agent is installed.

System Group

The System group is a collection of scalar objects that provides general information about the host on which the SNMP agent is installed:

```
iso (1)
      identified-organization (3)
         dod (6)
            internet (1)
               mgmt (2)
                  mib-II (1)
                     system (1) √
                        sysDescr (1) √
                        sysObjectID (2) √
                        sysUpTime (3) √
                        sysContact (4) √
                        sysName (5) √
                        sysLocation (6) √
                        sysServices (7) √
```

MIB Objects

Table 3-1 describes the MIB-II System group objects that are supported by the SNMP agent.

Table 3-1. System Group Objects Supported by SNMP Agent (page 1 of 2)

Object and Attributes	**Definition**	**Format and Derivation of Value**
sysDescr 1.3.6.1.2.1.1.1 read-only DisplayString (SIZE (0..255))	A description of the node on which the SNMP agent is installed.	NonStop Kernel system version: D*xx* or G*xx* Node: *node-name* Agent: *agent-process-name* Returned by Guardian procedure calls.
sysObjectID 1.3.6.1.2.1.1.2 read-only OBJECT IDENTIFIER	The enterprise identifier of the SNMP agent.	The constant value for this object is `1.3.6.1.4.1.169.3.155.1`. Assigned by the Internet.
sysUpTime 1.3.6.1.2.1.1.3 read-only TimeTicks	The difference, in hundredths of a second, between the GMT Julian timestamp when the SNMP agent was brought up and the current GMT Julian timestamp.	Refer to the documentation for your SNMP manager. Returned by Guardian procedure call. The value of sysUpTime is not reset as subagents come and go.
sysContact 1.3.6.1.2.1.1.4 read-write DisplayString (SIZE (0..255))	Information describing the contact person for this node. This value is stored in SNMPCTL.	Initially derived from the default value `unknown`. Object value can be set by an authorized SNMP manager.

Table 3-1. System Group Objects Supported by SNMP Agent (page 2 of 2)

Object and Attributes	Definition	Format and Derivation of Value
sysName 1.3.6.1.2.1.1.5 read-write DisplayString (SIZE (0..255))	A name associated with an Internet address by which the SNMP agent can be reached.	A character string of 255 or fewer characters. If a TCP/IP process has been specified in the TCPIP^PROCESS^NAME parameter, the value is derived from the name received from a gethostname call on that process. Otherwise, it is derived from the name received from a gethostname call on the local $ZTC0 process.
sysLocation 1.3.6.1.2.1.1.6 read-write DisplayString (SIZE (0..255))	The physical location of the node on which the SNMP agent is installed. This value is stored in SNMPCTL.	Initially derived from the default value of `unknown`. Can be set by an authorized SNMP manager.
sysServices 1.3.6.1.2.1.1.7 read-only INTEGER (0..127)	A numeric indication of the set of services supported, derived as defined in RFC 1213.	The constant value for this object is `79`, indicating that the following services are supported: physical (1) datalink/subnetwork (2) internet (3) end-to-end (4) applications (7)

RFC Compliance

All of the objects described in Table 3-1 comply with the definitions for System group objects as set forth in RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*.

SNMP Group

The SNMP group contains statistical information about all SNMP message activity handled by the SNMP agent.

The SNMP group consists of a collection of scalar objects. The SNMP agent provides values for the objects identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              snmp (11) √
                snmpInPkts (1)  √
                snmpOutPkts (2) √
                snmpInBadVersions (3) √
                snmpInBadCommunityNames (4) √
                snmpInBadCommunityUses (5) √
                snmpInASNParseErrs (6) √
                snmpInTooBigs (8) √
                snmpInNoSuchNames (9) √
                snmpInBadValues (10) √
                snmpInReadOnlys (11) √
                snmpInGenErrs (12) √
                snmpInTotalReqVars (13) √
                snmpInTotalSetVars (14) √
                snmpInGetRequests (15) √
                snmpInGetNexts (16) √
                snmpInSetRequests (17) √
                snmpInGetResponses (18) √
                snmpInTraps (19) √
                snmpOutTooBigs (20) √
                snmpOutNoSuchNames (21) √
                snmpOutBadValues (22) √
                snmpOutGenErrs (24) √
                snmpOutGetRequests (25) √
                snmpOutGetNexts (26) √
                snmpOutSetRequests (27) √
                snmpOutGetResponses (28) √
                snmpOutTraps (29) √
                snmpEnableAuthenTraps (30) √
```

MIB Objects

Table 3-2 describes the MIB-II SNMP group objects that are supported by the SNMP agent.

Table 3-2. SNMP Group Objects Supported by SNMP Agent (page 1 of 3)

Object and Attributes	Definition	Format and Derivation of Value
snmpInPkts 1.3.6.1.2.1.11.1 read-only Counter	The total number of messages received.	An integer set by the SNMP agent counter.
snmpOutPkts 1.3.6.1.2.1.11.2 read-only Counter	The total number of SNMP messages sent.	An integer set by the SNMP agent counter.
snmpInBadVersions 1.3.6.1.2.1.11.3 read-only Counter	The total number of SNMP messages received that were for an unsupported SNMP version.	An integer set by the SNMP agent counter.
snmpInBadCommunity-Names 1.3.6.1.2.1.11.4 read-only Counter	The total number of SNMP messages received that used an unknown SNMP community name.	An integer set by the SNMP agent counter.
snmpInBadCommunity-Uses 1.3.6.1.2.1.11.5 read-only Counter	The total number of SNMP messages received that represented an SNMP operation not allowed by the community named in the message.	An integer set by the SNMP agent counter.
snmpInASNParseErrs 1.3.6.1.2.1.11.6 read-only Counter	The total number of ASN.1 or BER errors encountered when decoding received SNMP messages.	An integer set by the SNMP agent counter.
snmpInTooBigs 1.3.6.1.2.1.11.8 read-only Counter	The total number of SNMP PDUs received for which the value of the error-status field was 'tooBig.'	The constant value for this object is 0.
snmpInNoSuchNames 1.3.6.1.2.1.11.9 read-only Counter	The total number of SNMP PDUs received for which the value of the error-status field was 'noSuchName.'	The constant value for this object is 0.
snmpInBadValues 1.3.6.1.2.1.11.10 read-only Counter	The total number of SNMP PDUs received for which the value of the error-status field was 'badValue.'	The constant value for this object is 0.

Table 3-2. SNMP Group Objects Supported by SNMP Agent (page 2 of 3)

Object and Attributes	Definition	Format and Derivation of Value
snmpInReadOnlys 1.3.6.1.2.1.11.11 read-only Counter	The total number of valid PDUs received for which the value of the error-status field was ‘readOnly.’	The constant value for this object is 0.
snmpInGenErrs 1.3.6.1.2.1.11.12 read-only Counter	The total number of PDUs received for which the value of the error-status field was ‘genErr.’	The constant value for this object is 0.
snmpInTotalReqVars 1.3.6.1.2.1.11.13 read-only Counter	The total number of MIB objects retrieved successfully as the result of receiving valid GetRequest and GetNextRequest PDUs.	An integer set by the SNMP agent counter.
snmpInTotalSetVars 1.3.6.1.2.1.11.14 read-only Counter	The total number of MIB objects altered successfully as the result of receiving valid SetRequest PDUs.	An integer set by the SNMP agent counter.
snmpInGetRequests 1.3.6.1.2.1.11.15 read-only Counter	The total number of GetRequest PDUs accepted and processed.	An integer set by the SNMP agent counter.
snmpInGetNexts 1.3.6.1.2.1.11.16 read-only Counter	The total number of GetNextRequest PDUs accepted and processed.	An integer set by the SNMP agent counter.
snmpInSetRequests 1.3.6.1.2.1.11.17 read-only Counter	The total number of SetRequest PDUs accepted and processed.	An integer set by the SNMP agent counter.
snmpInGetResponses 1.3.6.1.2.1.11.18 read-only Counter	The total number of GetResponse PDUs accepted and processed.	The constant value for this object is 0.
snmpInTraps 1.3.6.1.2.1.11.19 read-only Counter	The total number of Trap PDUs accepted and processed.	The constant value for this object is 0.
snmpOutTooBigs 1.3.6.1.2.1.11.20 read-only Counter	The total number of PDUs generated for which the value of the error-status field was ‘tooBig.’	An integer set by the SNMP agent counter.

Table 3-2. SNMP Group Objects Supported by SNMP Agent (page 3 of 3)

Object and Attributes	Definition	Format and Derivation of Value
snmpOutNoSuchNames 1.3.6.1.2.1.11.21 read-only Counter	The total number of PDUs generated for which the value of the error-status field was 'noSuchName.'	An integer set by the SNMP agent counter.
snmpOutBadValues 1.3.6.1.2.1.11.22 read-only Counter	The total number of PDUs generated for which the value of the error-status field was 'badValue.'	An integer set by the SNMP agent counter.
snmpOutGenErrs 1.3.6.1.2.1.11.24 read-only Counter	The total number of PDUs generated for which the value of the error-status field was 'genErr.'	An integer set by the SNMP agent counter.
snmpOutGetRequests 1.3.6.1.2.1.11.25 read-only Counter	The total number of GetRequest PDUs generated.	The constant value for this object is 0.
snmpOutGetNexts 1.3.6.1.2.1.11.26 read-only Counter	The total number of GetNextRequest PDUs generated.	The constant value for this object is 0.
snmpOutSetRequests 1.3.6.1.2.1.11.27 read-only Counter	The total number of SetRequest PDUs generated.	The constant value for this object is 0.
snmpOutGetResponses 1.3.6.1.2.1.11.28 read-only Counter	The total number of GetResponse PDUs generated.	An integer set by the SNMP agent counter.
snmpOutTraps 1.3.6.1.2.1.11.29 read-only Counter	The total number of trap PDUs generated.	An integer set by the SNMP agent counter.
snmpEnableAuthenTrap s 1.3.6.1.2.1.11.30 read-write INTEGER	An indication of whether the SNMP agent is permitted to generate authenticationFailure traps. This value is stored in SNMPCTL.	Valid values are: 1 (enabled) 2 (disabled) The default value is 1. At initial startup, the value is set to 1, but it can be reset by an authorized SNMP manager.

RFC Compliance

All of the objects described in Table 3-2 comply with the definitions for SNMP group objects as set forth in RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*.

zagInternal Group

The zagInternal group of the SNMP agent's private MIB is defined by HP and provides information about the SNMP agent's internal characteristics. The zagInternal group consists of objects in these four subgroups:

- zagInProcess objects, which describe the attributes of the SNMP agent's process
- zagInEndpoint objects, which identify the Internet address by which the SNMP agent can be addressed and the TCP/IP process that handles request/response messages between managers and the SNMP agent
- zagInProfile objects, which define the criteria the SNMP agent uses to determine whether to accept or reject incoming requests from SNMP managers that communicate with the SNMP agent through TCP/IP
- zagInTrapdest objects, which define SNMP manager Internet addresses and TCP/IP processes used by the SNMP agent when routing traps

The SNMP agent provides values for these objects, which are identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                zsmp (155)
                                    zsmpagent (1)
                                        zagInternal (7)
                                            zagInProcess (1)
                                                zagInProcCurrTime (1) √
                                                zagInProcVersion (2) √
                                                zagInProcName (3)  √
                                                zagInProcPAID (4)  √
                                                zagInProcPrimPID (5)  √
                                                zagInProcBkupPID (6)  √
                                                zagInProcCreatTime (7) √
                                                zagInProcCpuTime (8) √
                                                zagInProcPri (9) √
                                                zagInProcHomeTerm (10) √
        zagInProcMaxOpeners (11) √
        zagInProcCurrentOpeners (12) √
    zagInEndpoint (2)
        zagInEndpointTable (1) √
            zagInEndpointEntry (1) √
                zagInEpRowStatus (1) √
                zagInEpName (2) √
                zagInEpState (3) √
                zagInEpHostAddr (4) √
                zagInEpNetwork (5)  √
    zagInProfile (3)
        zagInProfileTable (1) √
            zagInProfileEntry (1) √
                zagInPfRowStatus (1) √
                zagInPfName (2) √
                zagInPfState (3) √
                zagInPfHostAddr (4) √
                zagInPfCommunity (5) √
                zagInPfAccess (6) √
    zagInTrapdest (4)
        zagInDirectedTrapdestName (1) √
        zagInTrapdestTable (2) √
            zagInTrapdestEntry (1) √
                zagInTdRowStatus (1) √
                zagInTdName (2) √
                zagInTdState (3) √
                zagInTdHostAddr (4) √
                zagInTdCommunity (5) √
                zagInTdNetwork (6) √
                zagInTdType (7) √
                zagInTdPort (8) √
```

MIB Objects

Table 3-3 describes the zagInProcess, zagInEndpoint, zagInProfile, and zagInTrapdest objects of the zagInternal group.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 1 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInProcess Group Objects:	Describe the attributes of the SNMP agent's process.	
zagInProcCurrTime 1.3.6.1.4.1.169.3.155.1.7.1.1 read-only DisplayString	A string that shows the current time.	*month DD, YYYY HH:MM:SS:centisecond* value, which is derived from SNMP agent calls to the Guardian procedure JULIANTIMESTAMP.
zagInProcVersion 1.3.6.1.4.1.169.3.155.1.7.1.2 read-only DisplayString	A string that shows the current version of the SNMP agent.	Numeric value of the form D*xx* that is set by the SNMP agent (for example, D23).
zagInProcName 1.3.6.1.4.1.169.3.155.1.7.1.3 read-only DisplayString	A string that shows the current process name for the SNMP agent.	The $*processName* value, which is assigned by the RUN command NAME option when the SNMP agent is started. Derived from SNMP agent calls to the Guardian procedure PROCESS_GETINFO_.
zagInProcPAID 1.3.6.1.4.1.169.3.155.1.7.1.4 read-only DisplayString	A string that shows the access ID (*group*, *user*) of the agent process.	The *group.user* value, which is determined by the user ID under which the SNMP agent was started. Derived from SNMP agent calls to the Guardian procedure PROCESS_GETINFO_.
zagInProcPrimPID 1.3.6.1.4.1.169.3.155.1.7.1.5 read-only DisplayString	A string that shows the primary process PID (*cpu,pin*) for the agent.	The *cpu,pin* value, which is assigned by the RUN command CPU option or by $CMON. Derived from SNMP agent calls to the Guardian procedure PROCESS_ GETPAIRINFO_.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 2 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInProcBkupPID 1.3.6.1.4.1.169.3.155.1.7.1.6 read-only DisplayString	A string that shows the backup process PID (`cpu,pin`) for the agent.	The `cpu`,`pin` value is assigned by the RUN command PRI option or by $CMON. Derived from SNMP agent calls to the Guardian procedure PROCESS_ GETPAIRINFO_.
zagInProcCreatTime 1.3.6.1.4.1.169.3.155.1.7.1.7 read-only DisplayString	A string that shows the agent process creation time.	*month DD*, *YYYY*, *HH*:*MM*:*SS*.*centisecond* value, which is derived from SNMP agent calls to the Guardian procedure PROCESS_ GETINFOLIST_.
zagInProcCpuTime 1.3.6.1.4.1.169.3.155.1.7.1.8 read-only DisplayString	A string that shows the CPU time for the SNMP agent process.	*HH*:*MM*:*SS*:*centisecond* value, which is derived from SNMP agent calls to the Guardian procedure PROCESS_ GETINFOLIST_.
zagInProcPri 1.3.6.1.4.1.169.3.155.1.7.1.9 read-only INTEGER (1..199)	The current process priority.	Numeric value with range of 1 through 199, which is assigned by the RUN command PRI option or by $CMON. Derived from SNMP agent calls to the Guardian procedure PROCESS_ GETINFOLIST_.
zagInProcHomeTerm 1.3.6.1.4.1.169.3.155.1.7.1.10 read-only DisplayString	A string that shows the process home terminal name.	The $*termName* value, which is assigned by the RUN command TERM option or defaults to the terminal from which the SNMP agent was started. Derived from SNMP agent calls to the Guardian procedure PROCESS_GETINFO_.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 3 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInProcMaxOpeners 1.3.6.1.4.1.169.3.155.1.7.1.11 read-only INTEGER	The maximum number of subagent and SCF (Subsystem Control Facility) sessions that the SNMP agent can support at one time.	A numeric value assigned by the RUN command MAXOPENERS startup parameter option. The default value is `20`.
zagInProcCurrentOpeners 1.3.6.1.4.1.169.3.155.1.7.1.12 read-only INTEGER	The current number of subagents.	A numeric value that is set by the SNMP agent.
zagInEndpoint Group Objects:	Identify the Internet address by which the SNMP agent can be addressed and the TCP/IP process that handles request/response messages between managers and the SNMP agent.	
zagInEndpointTable 1.3.6.1.4.1.169.3.155.1.7.2.1 not-accessible	A list of Endpoint object entries.	
zagInEndpointEntry 1.3.6.1.4.1.169.3.155.1.7.2.1.1 not-accessible INTEGER (1..65535)	An Endpoint entry containing objects that define a socket on which SNMP manager messages are received and to which replies are sent.	
zagInEpRowStatus 1.3.6.1.4.1.169.3.155.1.7.2.1.1.1 read-write INTEGER (1..6)	Used to manage the creation, alteration, and deletion of table rows. The row must be in the notReady or the notInService state for any row object to be modified.	Numeric value of 1 through 6. See Table 3-6 for RowStatus values and their descriptions. Can be set by the SNMP agent or by an authorized SNMP manager.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 4 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInEpName 1.3.6.1.4.1.169.3.155.1.7.2.1.1.2 read-only DisplayString (SIZE (2..8))	The name assigned to this endpoint. This object is created when the SNMP agent extracts the index value from the initial row creation varbind.	$*agentName.* *#endpointName* The SNMP agent prepends *$agentName* to *#endpointName*. ENDPOINT names are set from SCF and must: 1. Be uppercase 2. Start with a # 3. Use only the valid characters A through Z and 0 through 9
zagInEpState 1.3.6.1.4.1.169.3.155.1.7.2.1.1.3 read-only INTEGER	The current operational state of the Endpoint.	One of these four numeric state values, which can only be read: DEFINED (1) STARTED (3) STARTING (4) STOPPED (5) Default value is `DEFINED`. Set by the SNMP agent.
zagInEpHostAddr 1.3.6.1.4.1.169.3.155.1.7.2.1.1.4 read-write DisplayString	The local host IP address that the agent is to monitor for incoming messages.	Format of #.#.#.# For IP addresses, this value must either be the default value `0.0.0.0` or a fully specified address (for example, 103.252.12.3). Can be set by an authorized SNMP manager or from SCF.
zagInEpNetwork 1.3.6.1.4.1.169.3.155.1.7.2.1.1.5 read-write DisplayString	The name of the TCP/IP process the SNMP agent must use to establish the socket session.	$*TCP/IP process-name* The default process name is `$ZTC0`. Can be set by an SNMP manager, from SCF, or by using the TCPIP^PROCESS^NAME run startup parameter.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 5 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInProfile Group Objects:	Define criteria that the SNMP agent uses to determine whether to accept or reject incoming requests from SNMP managers that communicate with the SNMP agent through TCP/IP. In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the “public.”	
zagInProfileTable 1.3.6.1.4.1.169.3.155.1.7.3.1 not-accessible Sequence of ZagInProfileEntry	A list of profile entries.	
zagInProfileEntry 1.3.6.1.4.1.169.3.155.1.7.3.1.1 not-accessible zagInProfileEntry	A profile entry that contains objects used for authenticating a request that arrives over TCP/IP.	
zagInPfRowStatus 1.3.6.1.4.1.169.3.155.1.7.3.1.1.1 not-accessible through TCP/IP read-write access through IPC interface INTEGER (1..6)	Used to manage the creation, alternation, and deletion of table rows. The row must be in the notReady or notInService state for any row object to be modified.	Numeric value of 1 through 6. See Table 3-6 for RowStatus values and their descriptions. Set by the SNMP agent or through authenticated SNMP requests received through the IPC interface.
zagInPfName 1.3.6.1.4.1.169.3.155.1.7.3.1.1.2 not-accessible through TCP/IP read-only access through IPC interface DisplayString	The name assigned to this profile. This object is created by the SNMP agent by extracting the index value for the row creation varbind.	`#`*`profile-name`* The default value is `#DEFAULT`. Nondefault PROFILE table entry objects can be configured from SCF or through authenticated SNMP requests received through the IPC interface.

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 6 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInPfState 1.3.6.1.4.1.169.3.155.1.7.3.1.1.3 not-accessible through TCP/IP read-only access through IPC interface INTEGER	The current operational state of the profile.	Possible states are: DEFINED (1) STARTED (3) STARTING (4) STOPPED (5) Default value is DEFINED. Set by the SNMP agent.
zagInPfHostAddr 1.3.6.1.4.1.169.3.155.1.7.3.1.1.4 not-accessible through TCP/IP read-write access through IPC interface DisplayString	The address of the host sending the message. In conjunction with the community name and access attributes, this address provides the authentication attributes for validating received messages.	The value is set from SCF or through authenticated SNMP requests received through the IPC interface.
zagInPfCommunity 1.3.6.1.4.1.169.3.155.1.7.3.1.1.5 not-accessible through TCP/IP read-write access through IPC interface DisplayString	A data string compared with the agent section of the community string of received messages to authenticate the message. The SNMP agent uses the agent community string for request authentication and passes the entire community string to subagents.	The agent section of the community string can be 0 to 50 bytes. The default value is `public`. The value can be set from SCF or through authenticated SNMP requests received through the IPC interface.
zagInPfAccess 1.3.6.1.4.1.169.3.155.1.7.3.1.1.6 not-accessible through TCP/IP read-write access through IPC interface INTEGER {readOnly (1), readWrite (2)}	The authority of the community members to retrieve and alter network management information.	The default value is `1` (read-only). Can be set from SCF or through authenticated SNMP requests received through the IPC interface.
zagInTrapdest Group Objects:	Define SNMP manager Internet addresses and TCP/IP processes used by the SNMP agent when routing traps.	

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 7 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInDirectedTrapdestName 1.3.6.1.4.1.169.3.155.1.7.4.1 read-only DisplayString (SIZE (2..8))	This object is used by the subagent to specify a trap destination name (zagInTdName) in the agent's trap destination table to which a directed trap is sent. To issue a directed trap, a subagent must include this object as the first variable binding in the trap PDU.	The varbind value for this object must be the name of a configured trap destination.
zagInTrapdestTable 1.3.6.1.4.1.169.3.155.1.7.4.2 not-accessible	A list of trap destination entries.	
zagInTrapdestEntry 1.3.6.1.4.1.169.3.155.1.7.4.2.1 not-accessible	An entry containing objects that define the attributes of a trap destination.	
zagInTdRowStatus 1.3.6.1.4.1.169.3.155.1.7.4.2.1.1 read-write INTEGER (SIZE (1..6))	Used to manage the creation, alternation, and deletion of table rows. The row must be in the 'notReady' or the 'notInService' state for any row object to be modified.	Numeric value of 1 through 6. See Table 3-6 for RowStatus values and their descriptions. Set by the SNMP agent or by an authorized SNMP manager.

zagInTdName
 	
1.3.6.1.4.1.169.3.155.1.7.4.2.1.2
read-only
DisplayString (SIZE (2..8))
	The name assigned to this trap destination. This object is created when the SNMP agent extracts the index value from the initial row creation varbind.	`$agentName.`
		`#trapdestName`
		The SNMP agent prepends *`$agentName`* to *`#trapdestName`*.
		Nondefault trap destination names can be set from SCF and must:
		1. Be uppercase
		2. Start with a #
		3. Use only the valid characters A through Z and 0 through 9

Table 3-3. zagInternal Group Objects Supported by SNMP Agent (page 8 of 8)

Object and Attributes	Definition	Format and Derivation of Value
zagInTdState 1.3.6.1.4.1.169.3.155.1.7.4.2.1.3 read-only INTEGER	The current operational state of the trap destination.	One of these four numeric state values, which can only be read: DEFINED (1) STARTED (3) STARTING (4) STOPPED (5) Default value is `DEFINED`. Set by the SNMP agent.
zagInTdHostAddr 1.3.6.1.4.1.169.3.155.1.7.4.2.1.4 read-write DisplayString	The address of the host which is to receive trap messages.	Format of #.#.#.#. No default value is provided. Can be set by an authorized SNMP manager or through SCF.
zagInTdCommunity 1.3.6.1.4.1.169.3.155.1.7.4.2.1.5 read-write DisplayString (SIZE (0..50))	The name that the SNMP agent returns in traps sent to the trap destination.	Display string with a default value of “Tandem.” Can be set by an SNMP manager or from SCF.
zagInTdNetwork 1.3.6.1.4.1.169.3.155.1.7.4.2.1.6 read-write DisplayString	The name of the TCP/IP process that the agent uses to send traps.	$*`TCP/IP process name`* The default value is “$ZTC0.” Can be set by an authorized SNMP manager, assigned from SCF, or set by the run startup parameter TCPIP^PROCESS^NAME.
zagInTdType 1.3.6.1.4.1.169.3.155.1.7.4.2.1.7 read-write INTEGER (1..2)	Broadcast trap destinations are sent all trap messages except directed ones. Directed trap destinations receive only directed traps.	One of these two integer values is permitted: Broadcast (1) signifies nondirected traps. Directed (2) signifies directed traps. The default value is `Broadcast` (1). Can only be set by an authorized SNMP manager.
zagInTdPort 1.3.6.1.4.1.169.3.155.1.7.4.2.1.8 read-write INTEGER	An entry containing the value of the trap port.	The default value is 162. The SNMP agent is not enhanced to configure the trap port through SCF. You cannot view the port value using SCF.

SNMP Manager Access to Private MIB Objects

Table 3-4 identifies the objects in the SNMP agent's private MIB that allow access (read-only or read-write) by SNMP management applications through the SNMP interface. For a detailed definition of all of the SNMP agent's private MIB objects, see Table 3-3.

Table 3-4. Access Status of SNMP Agent's Private MIB Objects (page 1 of 3)

Object Name	Object Syntax	Manager Access	Description
zagInProcess Objects			
zagInProcCurrTime	DisplayString	read-only	The current time on the SNMP agent's host.
zagInProcVersion	DisplayString	read-only	The SNMP agent's version.
zagInProcName	DisplayString	read-only	The SNMP agent's process name.
zagInProcPAID	DisplayString	read-only	The SNMP agent's process access ID.
zagInProcPrimPID	DisplayString	read-only	The SNMP agent's primary process ID.
zagInProcBkupPID	DisplayString	read-only	The SNMP agent's backup process ID.
zagInProcCreatTime	DisplayString	read-only	The SNMP agent's instantiation timestamp.
zagInProcCpuTime	DisplayString	read-only	The SNMP agent's CPU usage time.
zagInProcPri	Integer	read-only	The SNMP agent's priority.
zagInProcHomeTerm	DisplayString	read-only	The SNMP agent's home terminal name.
zagInProcMaxOpeners	Integer	read-only	The maximum number of subagents the SNMP agent can simultaneously support.
zagInProcCurrentOpeners	Integer	read-only	The current number of subagents communicating with the SNMP agent.
zagInEndpointTable Row Objects			
zagInEpRowStatus	Integer	read-write	The object used to create or delete zagInEndpointTable entries.
zagInEpName	Integer	read-only	The name of the endpoint.

* Table row objects for zagInProfileTable are accessible to managers that communicate with the SNMP agent through the IPC interface. These objects are defined as "not-accessible" for SNMP requests received through TCP/IP's UDP transport protocol.

Table 3-4. Access Status of SNMP Agent's Private MIB Objects (page 2 of 3)

Object Name	Object Syntax	Manager Access	Description
zagInEpState	Integer	read-only	The operational state of the endpoint.
zagInEpHostAddr	DisplayString	read-write	The local host IP address the agent is to monitor for incoming messages.
zagInEpNetwork	DisplayString	read-write	The name of the TCP/IP process the agent must use to establish the socket session.
zagInProfileTable Row Objects*			
zagInPfRowStatus	Integer	read-write	The object used to create or delete zagInProfileTable entries.
zagInPfName	DisplayString	read-only	The name of the authentication table entry.
zagInPfState	Integer	read-only	The operational state of the authentication table entry.
zagInPfHostAddr	DisplayString	read-write	The Internet address of the manager whose security is being defined in the authentication table entry.
zagInPfCommunity	DisplayString	read-write	A data string compared with the community string of requests received through TCP/IP for the purpose of accepting or rejecting the request.
zagInPfAccess	Integer	read-write	The authority of the community members to retrieve and alter network management information.
zagInTrapdestTable Row Objects			
zagInTdRowStatus	Integer	read-write	The object used to create or delete zagInTrapdestTable entries.
zagInTdName	DisplayString	read-only	The name of the trap destination.
zagInTdState	Integer	read-only	The operational state of the trap destination.
zagInTdHostAddr	DisplayString	read-write	The address of the host which is to receive trap messages.

* Table row objects for zagInProfileTable are accessible to managers that communicate with the SNMP agent through the IPC interface. These objects are defined as "not-accessible" for SNMP requests received through TCP/IP's UDP transport protocol.

Table 3-4. Access Status of SNMP Agent's Private MIB Objects (page 3 of 3)

Object Name	Object Syntax	Manager Access	Description
zagInTdCommunity	DisplayString	read-write	The name that the SNMP agent returns in traps sent to the trap destination.
zagInTdNetwork	DisplayString	read-write	The name of the TCP/IP process that the agent uses to send traps.
zagInTdType	Integer	read-write	The trap type: broadcast or directed.
zagInTdPort	Integer	read-write	An entry containing the value of the trap port. The default vale is 162.

* Table row objects for zagInProfileTable are accessible to managers that communicate with the SNMP agent through the IPC interface. These objects are defined as "not-accessible" for SNMP requests received through TCP/IP's UDP transport protocol.

Table Row Management Overview

Two approaches can be used to add or delete entries from tables in the SNMP agent's private MIB:

- You can issue Set operations from SNMP management applications that add, modify, or delete a table row in the SNMP agent's private MIB. The instance values for columns in the table row can be created by a series of Set commands issued from a management application. Row objects that have no default values do not need to be created by a single Set command, as they do in the case of SCF. The subsection Managing Table Rows From SNMP Managers describes this approach.
- You can issue SCF commands (ABORT, ADD, ALTER, DELETE, and STOP) against ENDPOINT, PROFILE, or TRAPDEST objects to manage the SNMP agent's configuration. Configuration changes that you make through SCF are immediately reflected in the SNMP agent's private MIB. More specifically, when you use SCF commands to modify ENDPOINT, PROFILE, or TRAPDEST object attributes, a **RowStatus object** is automatically created for that object.

In Table 3-5, an "X" identifies table row objects for zagInEndpointTable, zagInProfileTable, and zagInTrapdestTable that can be accessed by SCF commands and those that can be accessed by Set operations issued from SNMP managers. The RowStatus and State object values determine the operations you can perform on a table row entry, either from an SNMP manager or SCF.

Table 3-5. SCF and SNMP Manager Access to Table Row Objects

Table Row Object	Accessible From SCF	Accessible to SNMP Managers Through TCP/IP's UDP Transport Protocol	Accessible to SNMP Managers Through the IPC Interface
zagInEndpointTable			
zagInEpRowStatus		X	X
zagInEpName	X	X	X
zagInEpState	X	X	X
zagInEpHostAddr	X	X	X
zagInEpNetwork	X	X	X
zagInProfileTable			
zagInPfRowStatus			X
zagInPfName	X		X
zagInPfState	X		X
zagInPfHostAddr	X		X
zagInPfCommunity	X		X
zagInPfAccess	X		X
zagInTrapdestTable			
zagInTdRowStatus		X	X
zagInTdName	X	X	X
zagInTdState	X	X	X
zagInTdHostAddr	X	X	X
zagInTdCommunity	X	X	X
zagInTdNetwork	X	X	X
zagInTdType		X	X
zagInTdPort			

Managing Table Rows From SNMP Managers

The zagInEndpointTable, zagInTrapdestTable, and zagInProfileTable in the SNMP agent's private MIB contain entries that can be accessed by SNMP managers. The presence of a RowStatus object (zagInEpRowStatus, zagInTdRowStatus, and zagInPfRowStatus) with read-write access signifies that managers can add or delete entries in these tables.

Note. The zagInProfileTable entries can only be accessed through SNMP requests received though NonStop Kernel IPC calls if the Processor/Accessor ID (PAID) of the manager process is compatible with that of the agent process. In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the "public" community, as well as Set requests sent under the "Private" community from the SNMP manager whose address is 130.252.15.230. The zagInProfileTable objects are not accessible through TCP/IP's UDP transport protocol.

The table row algorithm described in this subsection is based on RFC 1443, *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)*, but has been modified for SNMP version 1. Refer to RFC 1443 for a detailed description of the SNMPv2 table row algorithm.

The RowStatus object is expressed as an integer having one of the six values described in Table 3-6. The RowStatus value supplied by a manager dictates the table row operation. The two classes of **RowStatus values** are action and state. Action values are set by a manager to create and delete entries. State values describe the availability of an entry for use by the SNMP agent and managers.

Table 3-6. RowStatus Value Definitions (page 1 of 2)

RowStatus Value	Value Type and Access	Effect of RowStatus Value
active (1)	A state value that can be read or written.	Makes the row available for use or indicates that the row is already available for use by the SNMP agent.
notInService (2)	A state value that can be read or written.	Makes the row unavailable for use or indicates that the row exists and is complete, but is administratively unavailable to the SNMP agent.
notReady (3)	A state value that can only be read.	Indicates that the row exists but is incomplete; therefore, the row is unavailable to the SNMP agent.

Table 3-6. RowStatus Value Definitions (page 2 of 2)

RowStatus Value	Value Type and Access	Effect of RowStatus Value
createAndGo (4)	An action value that can only be written.	Is supplied in a Set command by the management application that wants to create a new row and have it available for use by the SNMP agent.
createAndWait (5)	An action value that can only be written.	Is supplied in a Set command by the management application that wants to create a new row and have it administratively unavailable to the SNMP agent.
destroy (6)	An action value that can only be written.	Is supplied in a Set command by a management application that wants to delete an entire table row.

When you use an SNMP manager to submit a request to operate on a table entry, the SNMP agent returns one of the following responses:

- noError if your operation was successful
- noSuchName if your operation failed

Table 3-7 identifies the RowStatus value that you supply in an SNMP manager's Set operation to implement a specific table row management operation. Table 3-7 also identifies the value to which the table row's State object (zagInEpState, zagInTdState, and zagInPfState) are subsequently set by the SNMP agent. The State object value identifies the current operational state of the table row (defined, started, starting, or stopped).

Table 3-7. RowStatus Values and Table Management Operations (page 1 of 2)

To do this...	Set one of the following row status objects (denoted in the next three columns) to this value...	zagInEpRowStatus Object	zagInTdRowStatus Object	zagInPfRowStatus Object[1]	The row state object[2] will be set to...	The equivalent SCF command issued against an ENDPOINT, TRAPDEST, or PROFILE object is...
Create a table row. (Add a definition to the SNMPCTL file.)	createAndWait	X	X	X	defined	ADD
Create a table row and make it available for use by the SNMP agent. (Add and activate a definition in the SNMPCTL file.)	createAndGo	X	X	X	starting, then started	ADD, then START

Table 3-7. RowStatus Values and Table Management Operations (page 2 of 2)

To do this...	Set one of the following row status objects (denoted in the next three columns) to this value...	zagInEpRowStatus Object	zagInTdRowStatus Object	zagInPfRowStatus Object[1]	The row state object[2] will be set to...	The equivalent SCF command issued against an ENDPOINT, TRAPDEST, or PROFILE object is...
Remove a table row. (Remove a definition from the SNMPCTL file.)	destroy	X	X	X	none (row no longer exists)	DELETE
Make a table row unavailable for use by the SNMP agent. (Inactivate a definition in the SNMPCTL file.)	notInService	X	X	X	defined if object has never been started; otherwise stopped	ABORT or STOP
Make a table row available for use by the SNMP agent. (Activate a definition in the SNMPCTL file.)	active	X	X	X	starting, then started	START

Creating a Table Row

To create a table row, you need to supply the index value that uniquely identifies the new row. You can then build a Set operation message that contains:

- The object identifier (OID) for the RowStatus object of the row being created, appended by the unique index value (name) for the new row
- A RowStatus value that specifies either “createAndGo” or “createAndWait”
- A value for each table row object for which no default exists

Table 3-8 identifies which objects have default values. Any table row object that has a default value is assigned that value if you provide none in your Set operation.

To create a new row in zagInTrapdestTable with the name and index value of “#TRAP,” you would provide an OID and a RowStatus value for inclusion in the Set PDU:

```
name: 1.3.6.1.4.1.169.3.155.1.7.4.2.1.1.35.84.82.65.80
objectSyntax: 1      (simpleSyntax)
simpleSyntax: 1      (number)
value: 4             (createAndGo)
```

The integer sequence in the `name` line contains the OID for zagInTdRowStatus (1.3.6.1.4.1.169.3.155.1.7.4.2.1.1) appended with the OID for “#TRAP” (35.84.82.65.80). In this example, “#TRAP” is the name and index value for the new row. Its OID is derived from the ASCII character set. In the `value` line of this example, a RowStatus value of 4 (createAndGo) has been specified.

Note. Row naming conventions must comply with those imposed for SCF. The row name must begin with a “#” character, followed by an alphabetic character and then one to six alphanumeric characters. All alphabetic characters in the name must be uppercase.

After receiving the Set operation message, the SNMP agent creates the RowStatus object and all other **columnar objects** that have default values. (See Table 3-8.) The row Name object (zagInEpName, zagInTdName, zagInPfName) represents a special case. That object is read-only; therefore, the SNMP agent creates it by extracting the row’s index value from the initial row creation varbind.

If your Set operation is successful, the SNMP agent assigns one of the following RowStatus values to your entry:

- active (1) is assigned if you specified createAndGo (4) and all entry objects have valid values assigned.
- notInService (2) is assigned if you specified createAndWait (5) and all entry objects have valid values assigned.
- notReady (3) is assigned if an object that has no default value has not yet been assigned a value.

Once a new row has been created from a management application Set operation, that table row is immediately accessible from SCF.

Default Values for Table Row Objects

Table 3-8 lists the default values for zagInEndpointTable and zagInTrapdestTable row objects.

Table 3-8. ENDPOINT and TRAPDEST Default Values

Table Row Object Attribute	zagInEndpointTable Default Value	zagInTrapdestTable Default Value	zagInProfileTable Default Value
NAME	none; a name must be supplied by an authorized SNMP management application.	none; a name must be supplied by an authorized SNMP management application.	none; a name must be supplied by an authorized SNMP management application.
STATE[1]	DEFINED	DEFINED	DEFINED
HostAddr	0.0.0.0	none provided	0.0.0.0
Network	$ZTC0	$ZTC0	N/A
Community	N/A	Tandem	public
Access	N/A	N/A	READ-ONLY
Type	N/A	BROADCAST	N/A
Port	N/A	162	N/A
RowStatus	See Table 3-9.	See Table 3-9.	See Table 3-9.

[1] For row creation only. For other operations, there is no default value.

Table 3-9 identifies the final values for zagInEpRowStatus, zagInTdRowStatus, and zagInPfRowStatus if you initially set them to createAndGo or createAndWait when:

- All row objects are defined.
- Some row objects are undefined.

Table 3-9. RowStatus Values With Defined and Undefined Row Objects

RowStatus Value	All Row Objects Defined	Some Row Objects Undefined
createAndGo (4)	active (1)	Operation rejected
createAndWait (5)	notInService (2)	notReady (3)

Activating Table Row Entries

You can activate an entry when it has one of the following RowStatus values:

- notInService (2). Set the value of the RowStatus object to active (1).
- notReady (3). Provide a value for any table row objects that have none and then set the value of the RowStatus object to active (1).

Deactivating Table Row Entries

To deactivate an entry, simply assign its RowStatus object a value of NotInService (2).

Modifying Table Row Entries

You can modify the value of an table row object only when its RowStatus is notInService (2) or notReady (3). If the RowStatus value is active (1), assign it the value notInService (2) and then make your modifications. To reactivate the entry, assign its RowStatus object the value active (1).

Deleting a Table Row

To delete a row, assign its RowStatus object the value destroy (6). An entry that has a value of notInService (2) or notReady (3) can be deleted.

If the row deletion succeeds, all object instances associated with the row are immediately removed from the SNMP agent's private MIB, and a "noError" response is returned by the SNMP agent.

Configuring the SNMP Agent Through SNMP Requests

You can also configure and manage the following components of the SNMP agent environment through SNMP requests:

Request/Response Connections

The SNMP agent's zagInEndpoint group objects define the Internet addresses and TCP/IP processes to be used for the receiving and sending of request/response messages.

Within the zagInEndpoint group, each row in the zagInEndpointTable represents one TCP/IP request/response connection definition and corresponds to an SCF ENDPOINT object.

Trap Connections

The SNMP agent's zagInTrapdest group objects identify the Internet addresses and TCP/IP processes to be used for the sending of trap messages. Trap ports can also be used.

Within the zagInTrapdest group, each row in the zagInTrapdestTable represents one trap destination definition and corresponds to an SCF TRAPDEST object.

Authentication Table Entries

The SNMP agent's zagInProfile group objects define the authentication table entries to be used for authenticating requests received from managers that communicate with the SNMP agent through TCP/IP. (For information on how requests received through NonStop Kernel IPC calls are authenticated. In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the "public" community, as well as Set requests sent under the "Private" community from the SNMP manager whose address is 130.252.15.230.

Within the zagInProfile group, each row in the zagInProfileTable represents one authentication table entry and corresponds to an SCF PROFILE object.

Because of security issues, zagInProfile group objects can be accessed only through SNMP requests received though NonStop Kernel IPC calls. Requests for operations on objects in the zagInProfile group are performed only if the process accessor ID (PAID) of the manager process is compatible with that of the agent process. In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the "public" community, as well as Set requests sent under the "Private" community from the SNMP manager whose address is 130.252.15.230. zagInProfile group objects are not accessible through TCP/IP's UDP transport protocol.

SNMP Agent Private MIB Objects

Table 3-11 lists the SNMP agent's private MIB objects that define the underlying resources comprising the SNMP agent's operations environment. SNMP agent private MIB objects that describe the SNMP agent process itself (the zagInProcess group objects, all of which are read-only) are not listed..

Note. The zagInProfile group objects can only be accessed through SNMP requests received though NonStop Kernel IPC calls. Requests for operations on objects in the zagInProfile group are performed only if the PAID of the manager process is compatible with that of the agent process. In Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the "public" community, as well as Set requests sent under the "Private" community from the SNMP manager whose address is 130.252.15.230. The zagInProfile group objects are not accessible through TCP/IP's UDP transport protocol.

Table 3-10 gives the initial values of objects in the SNMP Agent's MIB.

Table 3-10. Initial Values of Objects in the SNMP Agent's MIB

MIB Object	Initial Value
System group objects:	

sysDescr	`NonStop Kernel System`
	`Version: D`*`xx`*
	`Node:` *`node-name`*
	`Agent:` *`agent-process-name`*
	where D*xx* is the product version of the SNMP agent; for example, D23.

sysObjectID	1.3.6.1.4.1.169.3.155.1
sysContact	`"unknown"`
sysName	A name associated with the default TCP/IP process handling communication between the SNMP agent and SNMP managers. It is derived by a gethostname socket call against the TCP/IP process specified in the TCPIP^PROCESS^NAME startup parameter or, if none is specified, against $ZTC0 on the local node.
sysLocation	`"unknown"`
sysServices	79
SNMP group object:	
snmpEnableAuthenTraps	1 (yes)

Table 3-11 gives the SNMP agent's private MIB objects that describe the SNMP operations environment.

Table 3-11. SNMP Agent Private MIB Objects That Describe the SNMP Operations Environment (page 1 of 4)

SNMP Agent Private MIB Object	What the Object Defines and Its Default Value	Equivalent SCF Construct
zagInEndpointTable	Contains all of the TCP/IP request/response connection point definitions.	All of the ENDPOINT objects currently defined.
zagInEndpointEntry	Each row in the zagInEndpointTable represents one request/response connection definition.	ENDPOINT object.
zagInEpRowStatus	The value of this object is the means by which request/response connection definitions are created and removed, and also the means by which their availability for use by the SNMP agent is controlled.	See Table 3-4 on page 3-20.

Table 3-11. SNMP Agent Private MIB Objects That Describe the SNMP Operations Environment (page 2 of 4)

SNMP Agent Private MIB Object	What the Object Defines and Its Default Value	Equivalent SCF Construct
zagInEpName	A name that serves as an index into the table, uniquely identifying the table row containing the request/response connection definition.	*#endpoint-name*
zagInEpState	The operational state of the request/response connection definition.	Reflects result of issuing SCF ADD, START, STOP/ABORT commands as returned by a STATUS command. See Table 3-4 on page 3-20.
zagInEpHostAddr	The local host address the agent is to monitor for incoming messages. The default is “0.0.0.0”.	HOSTADDR attribute.
zagInEpNetwork	The TCP/IP process that handles request/response messages. The default is the value specified in the TCPIP^PROCESS^NAME startup parameter or, if none is specified, $ZTC0 on the local node.	NETWORK attribute.
zagInTrapdestTable	Contains all of the trap destination definitions.	All of the TRAPDEST objects currently defined.
zagInTrapdestEntry	Each row in the zagInTrapdestTable represents a trap destination definition.	TRAPDEST object.
zagInTdRowStatus	The value of this object is the means by which trap destination definitions are created and removed, and also the means by which their availability for use by the SNMP agent is controlled.	See Table 3-4 on page 3-20.
zagInTdName	A name that serves as an index into the table, uniquely identifying the table row defining the trap destination.	*#trapdest-name*
zagInTdState	The operational state of the trap destination definition.	Reflects result of issuing SCF ADD, START, STOP/ABORT commands as returned by a STATUS command. See Table 3-4 on page 3-20.

Table 3-11. SNMP Agent Private MIB Objects That Describe the SNMP Operations Environment (page 3 of 4)

SNMP Agent Private MIB Object	What the Object Defines and Its Default Value	Equivalent SCF Construct
zagInTdHostAddr	The Internet address of an SNMP manager to receive traps. By default, if no trap destinations are defined, the SNMP agent creates a trap destination definition for each Internet address from which a request message is received.	HOSTADDR attribute.
zagInTdCommunity	The name included in trap messages sent by the SNMP agent. “Tandem” is the default.	COMMUNITY attribute.
zagInTdNetwork	The TCP/IP process that handles the sending of trap messages. The default is the value specified in the TCPIP^PROCESS^NAME startup parameter or, if none specified, $ZTC0 on the local node.	NETWORK attribute.
zagInTdType	The type of the trap destination: broadcast or directed. “Broadcast” is the default.	Although there is no equivalent SCF construct, all trap destinations created through SCF have a type value of “broadcast.”
zagInTdPort	An entry containing the value of the trap port. The default vale is 162.	This object can not be viewed/modified using the SCF commands. This can only be set by the manager when the zagInTdRowStatus is DEFINED or STOPPED.
zagInProfileTable	Contains all of the authentication table entries.	All of the PROFILE objects currently defined.
zagInProfileEntry	Each row in the zagInProfileTable represents an authentication table entry.	PROFILE object.
zagInPfRowStatus	The value of this object is the means by which authentication table entries are created and removed, and also the means by which their availability for use by the SNMP agent is controlled.	See Table 3-4 on page 3-20.

Table 3-11. SNMP Agent Private MIB Objects That Describe the SNMP Operations Environment (page 4 of 4)

SNMP Agent Private MIB Object	What the Object Defines and Its Default Value	Equivalent SCF Construct
zagInPfName	A name that serves as an index into the table, uniquely identifying the table row defining the entry.	*#profile-name*
zagInPfState	The operational state of the authentication table entry.	Reflects result of issuing SCF ADD, START, STOP/ABORT commands as returned by a STATUS command. See Table 3-4 on page 3-20.
zagInPfHostAddr	The Internet address of the manager or managers whose security is being defined in the authentication table entry. The default ("0.0.0.0") indicates that the authentication table entry applies to all managers that communicate through TCP/IP.	HOSTADDR attribute.
zagInPfCommunity	A data string compared with the community string of requests received through TCP/IP for the purpose of accepting or rejecting the request. The default is "public."	COMMUNITY attribute.
zagInPfAccess	The authority of the community members to retrieve and alter network management information. The default is readOnly (1).	ACCESS attribute.

Managing Configuration Definitions Through SNMP

You can manage the SNMP agent configuration by manipulating rows in the SNMP agent's private MIB tables, as follows:

- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent's private MIB zagInEndpointTable to manage the TCP/IP connection points that the SNMP agent uses for request/response messages. Each row in the zagInEndpointTable corresponds to an SCF ENDPOINT object.
- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent's private MIB zagInTrapdestTable to manage the destinations to which the SNMP agent sends traps. Each row in the zagInTrapdestTable corresponds to an SCF TRAPDEST object. However, the object zagInTdPort cannot be viewed or managed through SCF, only from the manager.

- You create, alter the attributes of, activate, inactivate, and remove rows in the SNMP agent’s private MIB zagInProfileTable to manage the authentication table entries that the SNMP agent uses for accepting or rejecting request messages received through TCP/IP. Each row in the zagInProfileTable corresponds to an SCF PROFILE object.

Note. Because of security issues, zagInProfile group objects can be accessed only through SNMP requests received though NonStop Kernel IPC calls. Requests for operations on objects in the zagInProfile group are performed only if the PAID of the manager process is compatible with that of the agent process. For more details, see Figure 2-5, the authentication table permits the SNMP agent to accept Get and GetNext requests from any SNMP manager belonging to the “public” community, as well as Set requests sent under the “Private” community from the SNMP manager whose address is 130.252.15.230. The zagInProfile group objects are not accessible through TCP/IP’s UDP transport protocol. The zagInProfileTable objects can also be created and managed through SCF as PROFILE objects.

Part II. SCF for the SNMP Agent

Part III consists of the following sections, which describe the Subsystem Control Facility (SCF) interface to the SNMP agent:

4 Introduction to SCF for the SNMP Agent

The Subsystem Control Facility (SCF) provides a common syntax for managing HP subsystems. Managing a subsystem involves configuring, controlling, and inquiring about subsystem components known as objects. This section provides an introduction to using SCF to manage the Simple Network Management Protocol (SNMP) agent product offered by HP—the NonStop agent.

- If you have worked with SCF before, you can skip to TRACE Command on page 5-35.
- If you are not familiar with SCF, reading this section in its entirety should provide enough information to get you started with the SCF interface.

This section describes:

- Tasks you can perform with SCF
- SCF and the DSM family of products
- SCF concepts of objects, object attributes, and object states
- Running SCF
- Entering SCF commands
- SCF error messages
- The SCF online help facility
- A brief introduction to the SNMP network management environment and the part the NonStop agent plays in such an environment
- SCF information specific to the NonStop agent

For systems running a G-series RVU of the HP NonStop Kernel operating system, the *SCF Reference Manual for G-Series Releases* contains comprehensive reference material on SCF.

For systems running an H-series RVU, the *SCF Reference Manual for H-Series RVUs* contains comprehensive reference material on SCF.

Tasks You Can Perform With SCF

You can use SCF to manage various components of a NonStop agent configuration to reflect the needs of your particular operations environment.

If you start the NonStop agent with its built-in defaults, you can use SCF to modify the default configuration values. If you start the NonStop agent without any predefined configuration values, you can enter the complete configuration through SCF.

SCF and the DSM Family of Products

SCF is part of the Distributed Systems Management (DSM) family of products. DSM provides applications, services, and tools to help you manage the operations of networks of distributed systems. Its architecture and interfaces to help you automate management of such networks.

Subsystem Programmatic Interface

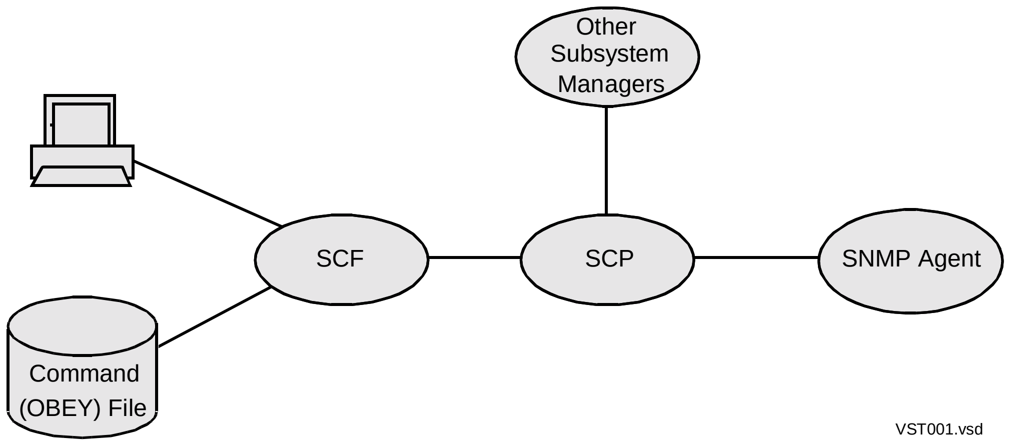
The management services provided by DSM depend on the Subsystem Programmatic Interface (SPI), a set of procedures, data definitions, and protocols that support the interface between the operations and subsystem environments. SPI is the foundation on which some management applications are built, and the foundation of the SCF interface to HP subsystems that support SPI.

The Subsystem Control Point

SCF provides an operator interface to an intermediate process called the Subsystem Control Point (SCP). The SCP process is a network-management process that receives and distributes messages sent by SCF to HP subsystems. The SCP process provides the interface between SCF and the NonStop agent, as Figure 4-1 illustrates.

The SCP process provides security (by restricting access to sensitive commands), version control, and tracing support for subsystems, and it supports NonStop application processes. SCF automatically opens and closes the SCP process as needed. For more information on the SCP process, refer to the *SCF Reference Manual for SCP*.

Figure 4-1. Overview of SCF



Objects

In HP systems, a subsystem is represented as a collection of objects that describe a set of services or resources. An object is a distinct entity that management software can query and control. Objects might represent real resources or a relationship between resources. There might be multiple instances of some objects, but not others. For example, in each NonStop agent environment, there can be only one PROCESS object, but many TRAPDEST objects.

Each subsystem object is identified by an object type and an object name:

- The object type describes the nature of the object, such as a PROCESS or TRAPDEST (trap destination).
- The object name uniquely identifies an object within the system.

The following objects are supported for HP SNMP:

PROCESS object	The PROCESS object represents a NonStop agent process running on an HP system
ENDPOINT object	The ENDPOINT object type represents a network interface the NonStop agent process uses to accept manager requests and return responses
PROFILE object	The PROFILE object type represents an entry in the **authentication table** which the NonStop agent process consults to determine whether to accept an incoming request.
TRAPDEST object	The TRAPDEST object type represents a manager to which the NonStop agent process can send trap messages.

Object Attributes

Most objects have an associated set of attributes. Each object attribute identifies some characteristic of that object. To define an attribute with the ADD command or to change it with the ALTER command, you provide an attribute specifier, consisting of an attribute name and a value. You can add or alter several attributes of an object in a single command. Default values exist for most attributes.

The INFO command displays the current attribute values for an object. In the display generated by the INFO command, attributes whose values can be altered are preceded by an asterisk. For example:

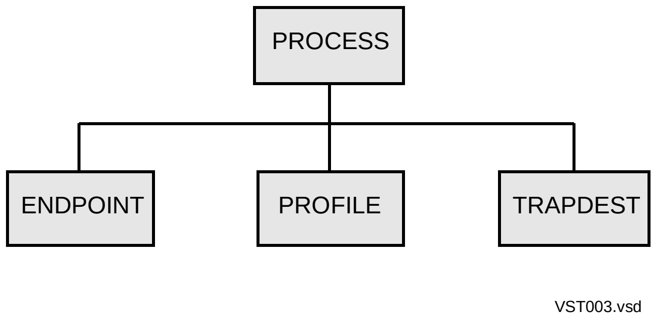
```
15-> INFO TRAPDEST $ZSNMP.#TRAP

SNMP Info TRAPDEST

Name                *Network            *Hostaddr           *Community

$ZSNMP.#TRAP        \EAST.$ZTC0         130.252.85.199      T16

16->
```

For the required syntax for each attribute name and value valid for NonStop agent objects, see the individual commands in Section 5, SCF Commands for the SNMP Agent.

Object Hierarchy

Most subsystems are structured hierarchically, with a group of objects of one type logically subordinate to an object of another type. For example, a number of trap destinations (TRAPDEST objects) can be configured to receive traps from a single NonStop agent process (PROCESS object).

Figure 4-2.



Object States

An object is always in one of several operational states, such as DEFINED, STARTED, or STOPPED. The sequence of states an object goes through varies from object to object and from subsystem to subsystem. The state of an object at a given time is important. Certain commands have no effect on an object when it is in one state but do affect the object when it is in another state.

Table 4-1. States Reported by NonStop Agent Objects

Object Type	DEFINED	STARTED	STARTING	STOPPING	STOPPED
PROCESS		X	X*		
ENDPOINT	X	X	X	X	X
PROFILE	X	X			X
TRAPDEST	X	X	X		X

* The STARTING state is reported in event messages for the PROCESS object when the NonStop agent is performing initialization tasks, but is never displayed in response to an SCF STATUS command.

The PROCESS Object

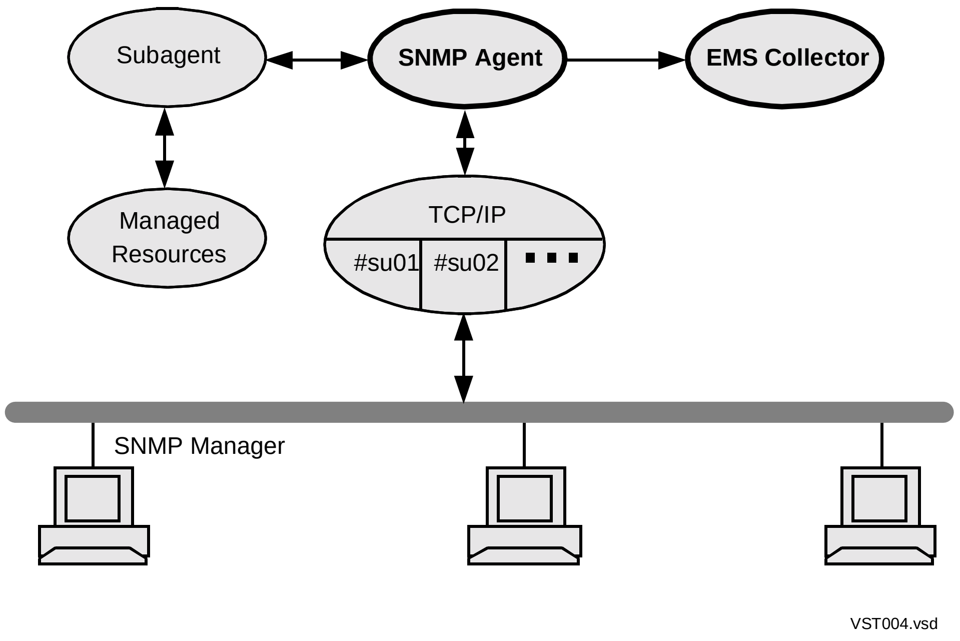
The PROCESS object represents a NonStop agent process running on an HP system.

The NonStop agent process authenticates SNMP requests from managers, processes them or forwards them to appropriate subagents, and returns responses to the manager from which the request originated. The NonStop agent also generates and sends traps it or the subagents with which it is communicating generate to managers configured to receive them.

The PROCESS object has one attribute—EMSCOLL—that defines the EMS **collector** to which the NonStop agent process sends event messages it generates.

The components of the NonStop agent configuration represented by the PROCESS object type and its attributes are highlighted in bold in Figure 4-3.

Figure 4-3. The PROCESS Object



Naming Conventions: PROCESS Object

In SCF commands, the name associated with the PROCESS object is the name assigned to the NonStop agent process when it is invoked with the TACL RUN command.

The process name consists of a dollar sign ($) followed by an alphanumeric string of up to five characters, the first of which must be alphabetic.

For example, suppose you start the NonStop agent process as follows:

```
TACL> RUN SNMPAGT /NAME $ZSNMP, NOWAIT, CPU 4/ 6
```

The associated PROCESS object is named $ZSNMP.

Note. For information on NonStop agent process startup parameters, see the Section 2, Installing and Configuring the SNMP Agent.

The syntax for specifying a PROCESS object in SCF commands:

```
command PROCESS $agent-process
```

Wildcard Support: PROCESS Object

The use of the asterisk (*) as a wildcard is not supported for the PROCESS object.

PROCESS Object States

The PROCESS object is always reported as being in the STARTED state in response to a STATUS command. If the NonStop agent process is not running, SCF returns an Expecting an existing SCF supported object name error.

Note. The PROCESS object enters the STARTING state while the NonStop agent process is performing its initialization tasks. This state is reported in event messages for the PROCESS object, but is never displayed in response to an SCF STATUS command.

Figure 4-4 illustrates the state transition sequence for the PROCESS object.

PROCESS Object Attributes

EMSCOLL is the only attribute supported for the PROCESS object. EMSCOLL specifies the EMS collector process to which the NonStop agent is to send events it generates. For more detail about this attribute, see the ALTER Command on page 5-13.

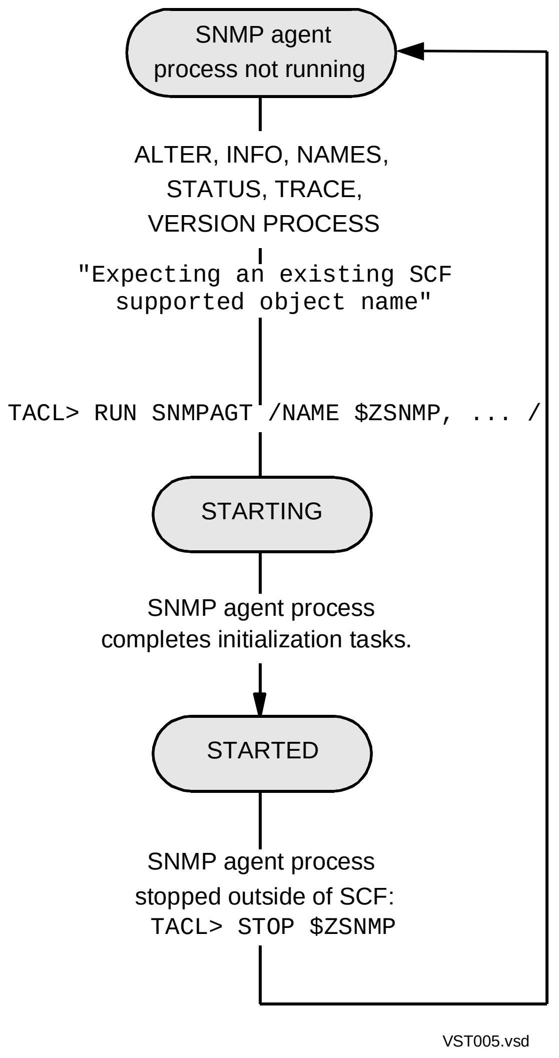
Default PROCESS Object

When you initially start the NonStop agent process, the following PROCESS object is automatically configured and started:

```
PROCESS $ZSNMP, EMSCOLL $0
```

Note. This and all other examples assume that the NonStop agent process was started as $ZSNMP. For a description of the TACL RUN command syntax for starting a NonStop agent process (see RUN Command on page 2-7).

Figure 4-4. PROCESS Object State Transition Sequence


SNMP agent process not running
ALTER, INFO, NAMES, STATUS, TRACE, VERSION PROCESS
"Expecting an existing SCF supported object name"
TACL> RUN SNMPAGT /NAME $ZSNMP, ... /
STARTING
SNMP agent process completes initialization tasks.
STARTED
SNMP agent process stopped outside of SCF:
TACL> STOP $ZSNMP
VST005.vsd

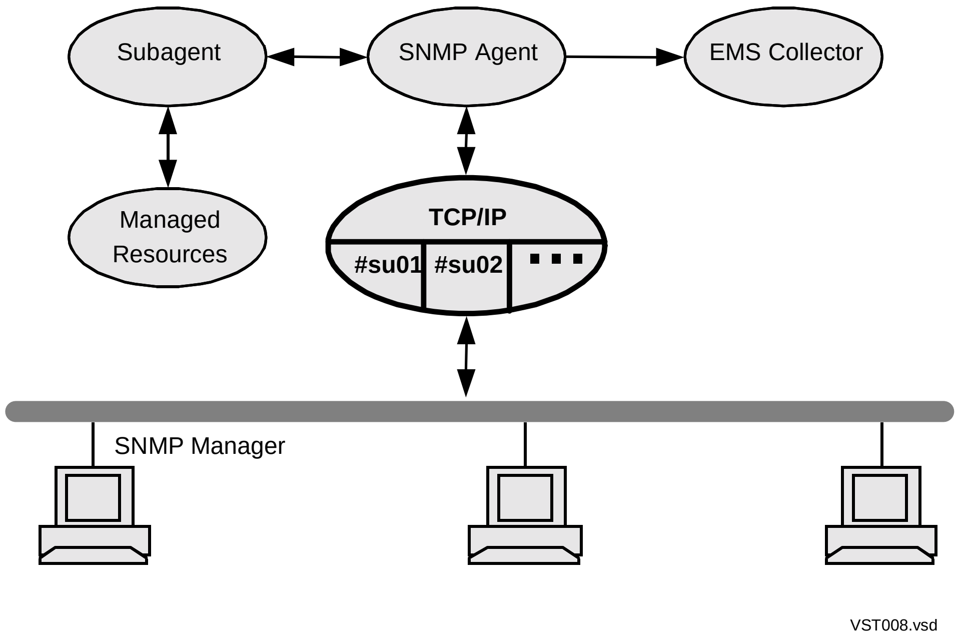
The ENDPOINT Object

The ENDPOINT object type represents a network interface the NonStop agent process uses to accept manager requests and return responses. The attributes of the ENDPOINT object identify:

- The name of the TCP/IP process through which the NonStop agent can be reached by other entities on the network (NETWORK attribute)
- The Internet addresses (TCP/IP **subnet** addresses) by which the NonStop agent is known to other entities on the network (HOSTADDR attribute)

The components of a NonStop agent configuration represented by the ENDPOINT object type and its attributes are highlighted in bold in Figure 4-5.

Figure 4-5. The ENDPOINT Object



Naming Conventions: ENDPOINT Object

You assign a unique name to each request/response connection being used by a given NonStop agent process. The name consists of:

- The name of the PROCESS object that identifies the NonStop agent process being served by the request/response connection
- A period (.)
- A pound sign (#)
- A logical identifier representing the ENDPOINT object definition

The logical identifier consists of an alphabetic character followed by up to six additional alphanumeric characters. For example:

```
$ZSNMP.#END1
```

In the SCF command syntax descriptions, the logical identifier associated with a request/response connection is referred to as its *endpoint-name*. The syntax for specifying an ENDPOINT object type:

```
command ENDPOINT [$agent-process.]#endpoint-name
```

Note. If you set a default PROCESS with the ASSUME PROCESS command, you can omit the NonStop agent process name and period, and just specify *#endpoint-name*. For example:

```
-> ASSUME PROCESS $ZSNMP
-> INFO ENDPOINT #END1
-> STATUS ENDPOINT #END1
```

Wildcard Support: ENDPOINT Object

The use of the asterisk (*) as a wildcard is supported in ENDPOINT object name specifications for the following commands:

ABORT	INFO	STOP
ALTER	START	
DELETE	STATUS	

For example, the following command returns the status of all ENDPOINT objects whose object names begin with #END:

```
-> STATUS ENDPOINT $ZSNMP.#END*
```

The next example returns the status of all ENDPOINT objects defined for the NonStop agent process named $ZSNMP:

```
-> ASSUME PROCESS $ZSNMP
-> STATUS ENDPOINT *
```

ENDPOINT Object States

The ENDPOINT object is always in one of the DEFINED, STARTED, STARTING, STOPPING or STOPPED states.

Table 4-2 describes the states the ENDPOINT object supports.

Table 4-2. ENDPOINT Object States

State	Description
DEFINED	The ENDPOINT object has been configured with the SCF ADD command but has not been initially activated for use in the NonStop agent operations environment.
STARTED	The ENDPOINT object is either ready to accept or in the process of handling requests.
STARTING	The ENDPOINT object has been started, but it has not been able to open the TCP/IP process specified by its NETWORK attribute.
STOPPING	The ENDPOINT object is in the process of stopping. The ENDPOINT object enters this state while the NonStop agent completes any outstanding requests when a STOP or ABORT command was issued.
STOPPED	The ENDPOINT object is configured but not currently being used by the NonStop agent. It has either been explicitly stopped with an SCF STOP or ABORT command, or an external resource on which it depends is unavailable.

Figure 4-6 illustrates the state transition sequence for the ENDPOINT object.

ENDPOINT Object Attributes

The following attributes are supported for the ENDPOINT object:

- HOSTADDR identifies the TCP/IP process handling communication between the NonStop agent process and managers for the receiving and sending of SNMP request/response messages.
- NETWORK specifies the Internet addresses by which the NonStop agent is known to the network.

For a description of these attributes, see the ADD Command on page 5-5.

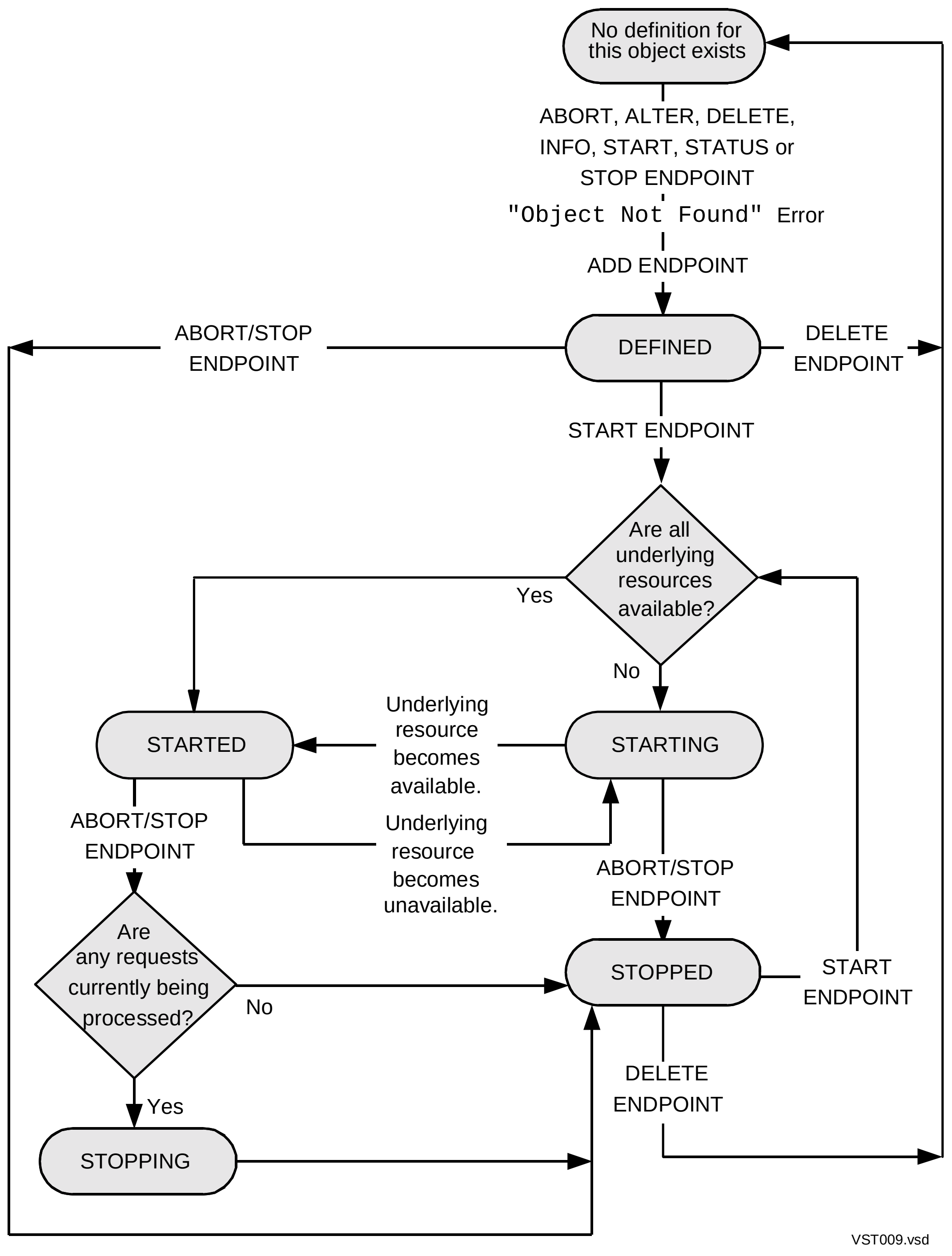
Default ENDPOINT Object

When you initially start the NonStop agent process with the default WARM and TCPIP^PROCESS^NAME options, the following ENDPOINT object is automatically configured and started:

```
ENDPOINT $agent-process.#DEFAULT,NETWORK $ZTC0,HOSTADDR
"0.0.0.0"
```

This default ENDPOINT object is stored in the SNMPCTL file.

Figure 4-6. ENDPOINT Object State Transition Sequence


No definition for this object exists
ABORT, ALTER, DELETE, INFO, START, STATUS or STOP ENDPOINT
"Object Not Found" Error
ADD ENDPOINT
ABORT/STOP ENDPOINT
DEFINED
DELETE ENDPOINT
START ENDPOINT
Are all underlying resources available?
Yes
No
STARTED
Underlying resource becomes available.
STARTING
ABORT/STOP ENDPOINT
Underlying resource becomes unavailable.
ABORT/STOP ENDPOINT
Are any requests currently being processed?
No
STOPPED
START ENDPOINT
Yes
DELETE ENDPOINT
STOPPING
VST009.vsd

The PROFILE Object

The PROFILE object type represents an entry in the **authentication table** which the NonStop agent process consults to determine whether to accept an incoming request. The attributes of the PROFILE object type define the contents of the following table entry fields:

- Community name (COMMUNITY attribute)
- Internet address (HOSTADDR attribute)
- **Access mode** (ACCESS attribute)

If components of an incoming SNMP request match a PROFILE object definition, the NonStop agent either processes or forwards the request to the appropriate subagent. If they do not, the NonStop agent rejects the request.

The components of a NonStop agent configuration represented by the PROFILE object type and its attributes are highlighted in bold in Figure 4-7.

Naming Conventions: PROFILE Object

You assign a unique name to each entry in the authentication table being used by a given NonStop agent process. The name consists of:

- The name of the PROCESS object that identifies the NonStop agent process using the authentication table
- A period (.)
- A pound sign (#)
- A logical identifier representing the PROFILE object definition

The logical identifier consists of an alphabetic character followed by up to six additional alphanumeric characters.

For example:

```
$ZSNMP.#PROFILE
```

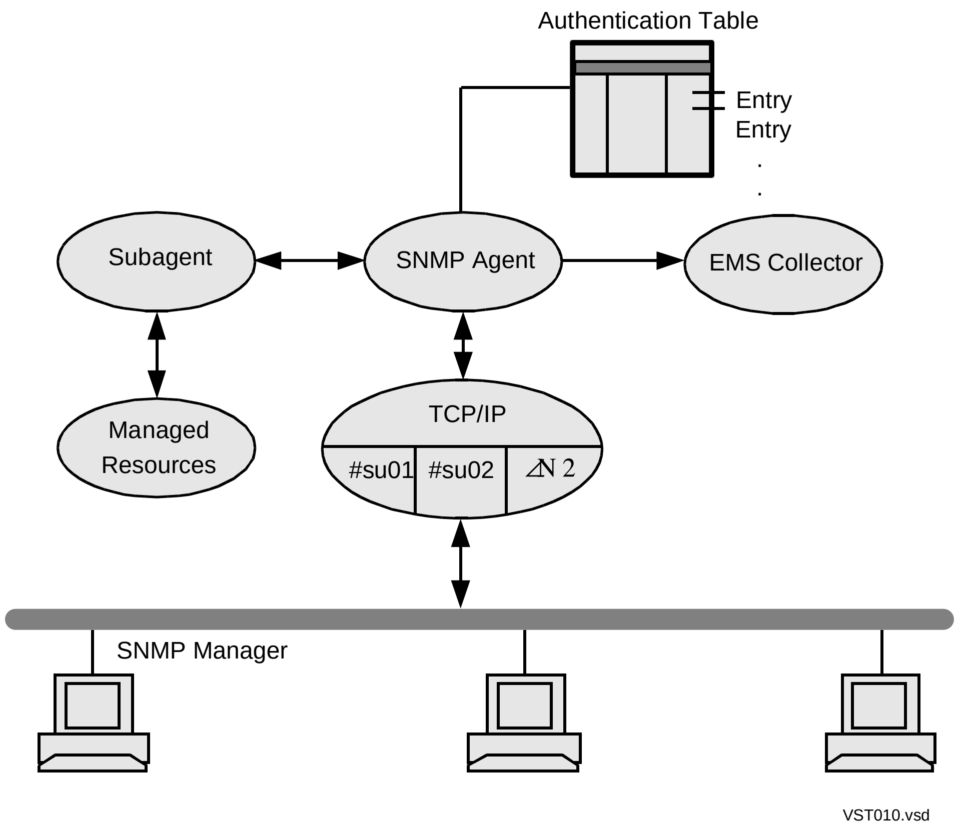
In the SCF command syntax descriptions, the logical identifier associated with an authentication table entry is referred to as its *`profile-name`*. The syntax for specifying a PROFILE object type:

```
command PROFILE [$agent-process.]#profile-name
```

Note. If you set a default PROCESS with the ASSUME PROCESS command, you can omit the NonStop agent process name and period, and just specify #*`profile-name`*, for example:

```
-> ASSUME PROCESS $ZSNMP
-> INFO PROFILE #PROFILE
-> STATUS PROFILE #PROFILE
```

Figure 4-7. The PROFILE Object



Wildcard Support: PROFILE Object

The use of the asterisk (*) as a wildcard is supported in PROFILE object name specifications for the following commands:

ABORT	INFO	STOP
ALTER	START	
DELETE	STATUS	

For example, the following command returns information about all PROFILE objects whose object name begins with #PROF:

```
-> INFO PROFILE $ZSNMP.#PROF*
```

The next example would return information about all PROFILE objects configured for the NonStop agent process $ZSNMP:

```
-> ASSUME PROCESS $ZSNMP
-> INFO PROFILE *
```

PROFILE Object States

The PROFILE object is always in one of the DEFINED, STARTED, or STOPPED states.

Table 4-3 describes the states supported for the PROFILE object:

Table 4-3. PROFILE Object States

State	Description
DEFINED	The PROFILE object has been configured with the SCF ADD command but has not been initially activated for use in the NonStop agent **authentication scheme**.
STARTED	The entry in the authentication table is available for the NonStop agent process to use for security checking.
STOPPED	The PROFILE object is configured but is not currently being used in the NonStop agent authentication scheme. It has been explicitly stopped with the SCF STOP or ABORT command.

Figure 4-8 illustrates the state transition sequence for the PROFILE object.

PROFILE Object Attributes

The following attributes are supported for the PROFILE object:

- ACCESS specifies the level of authority of an associated community to retrieve and alter network management information.
- COMMUNITY specifies the community name that must be present in requests sent by managers for the NonStop agent to accept the request.
- HOSTADDR specifies the Internet address of the manager(s) whose security is being defined in the authentication table entry.

For a detailed description of these attributes, see the ADD Command on page 5-5.

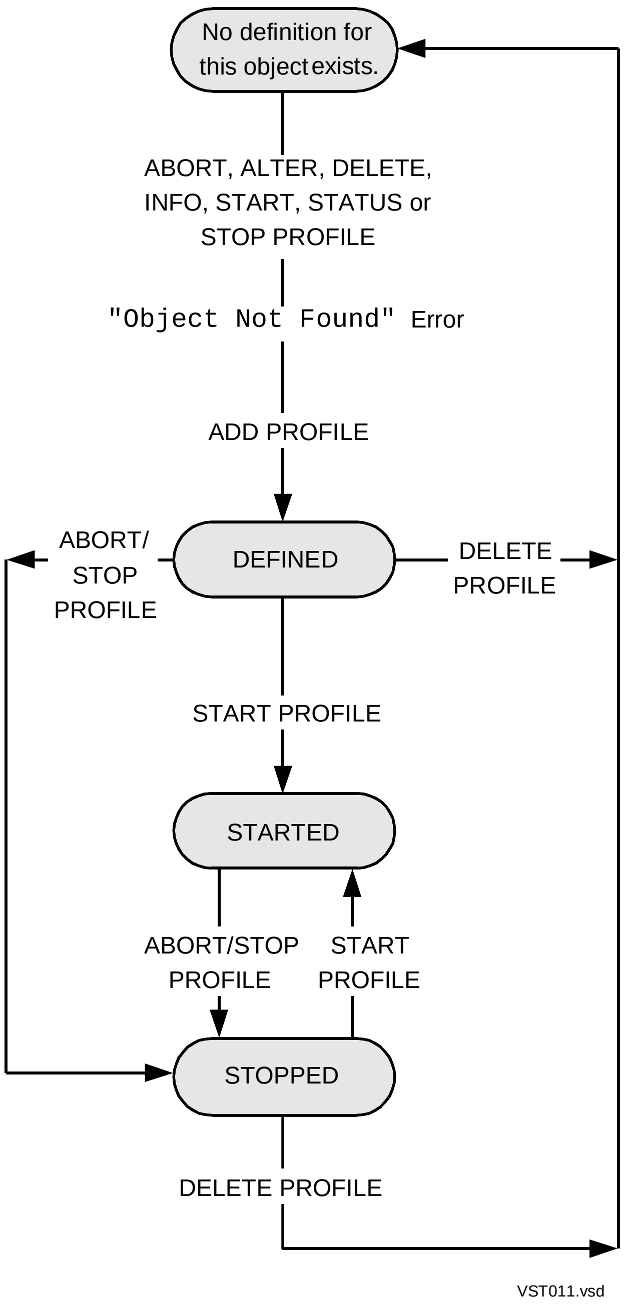
Default PROFILE Object

When you initially start the NonStop agent process with the default WARM option, the following PROFILE object is automatically configured and started:

```
PROFILE $agent-process.#DEFAULT, COMMUNITY "public",
  ACCESS READONLY, HOSTADDR "0.0.0.0"
```

This default PROFILE object is stored in the SNMPCTL file.

Figure 4-8. PROFILE Object State Transition Sequence



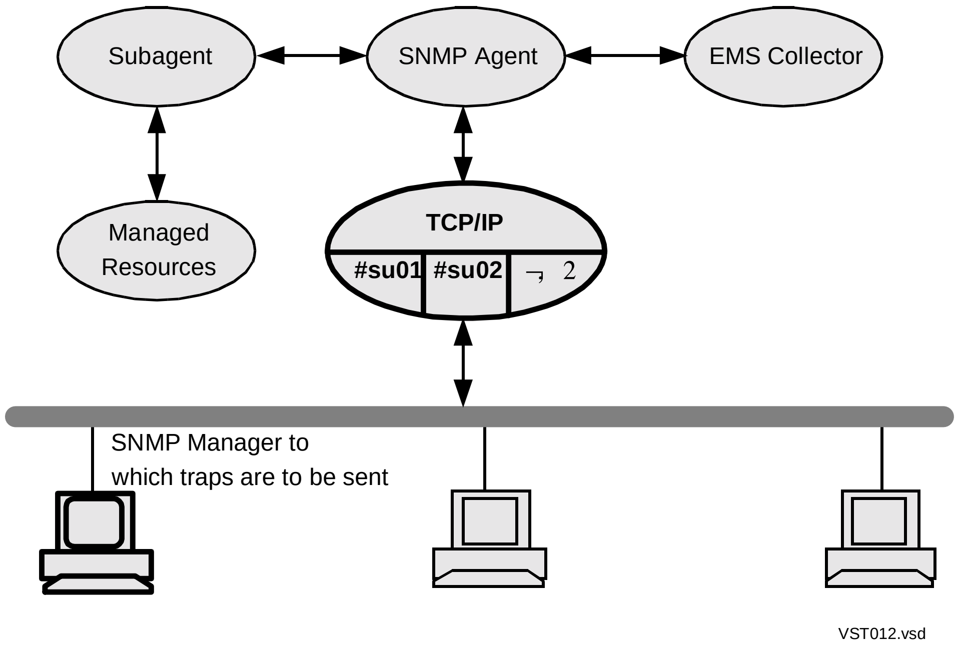
The TRAPDEST Object

The TRAPDEST object type represents a manager to which the NonStop agent process can send trap messages. The attributes of the TRAPDEST object identify:

- The TCP/IP process that the NonStop agent process uses to send traps (NETWORK attribute)
- The community name that the NonStop agent includes in traps it generates (COMMUNITY attribute)
- The Internet address of the manager to which traps are to be sent (HOSTADDR attribute)

The components of a NonStop agent configuration represented by the TRAPDEST object type and its attributes are highlighted in bold in Figure 4-9.

Figure 4-9. The TRAPDEST Object



Naming Conventions: TRAPDEST Object

You assign a unique name to each trap destination configured for a given NonStop agent process. The name consists of:

- The name of the PROCESS object that identifies the NonStop agent process sending traps to the configured trap destination
- A period (.)

- A pound sign (#)
- A logical identifier representing the TRAPDEST object definition

The logical identifier consists of an alphabetic character followed by up to six additional alphanumeric characters. For example:

```
$ZSNMP.#TRAP1
```

In the SCF command syntax descriptions, the logical identifier associated with a trap destination configuration is referred to as its *trapdest-name*. The syntax for specifying a TRAPDEST object type:

```
command TRAPDEST $agent-process.#trapdest-name
```

Note. If you set a default PROCESS with the ASSUME PROCESS command, you can omit the NonStop agent process name and period and just specify #*trapdest-name*. For example:

```
-> ASSUME PROCESS $ZSNMP
-> INFO TRAPDEST #TRAP1
-> STATUS TRAPDEST #TRAP1
```

Wildcard Support: TRAPDEST Object

The use of the asterisk (*) as a wildcard is supported in TRAPDEST object name specifications for the following commands:

ABORT	INFO	STOP
ALTER	START	
DELETE	STATUS	

The following example returns information about all TRAPDEST objects whose object name began with #TRAP:

```
-> INFO TRAPDEST $ZSNMP.#TRAP*
```

The following example returns information about all TRAPDEST objects defined for NonStop agent process $ZSNMP:

```
-> ASSUME PROCESS $ZSNMP
-> INFO TRAPDEST *
```

TRAPDEST Object States

The TRAPDEST object is always in one of the DEFINED, STARTING, STARTED, or STOPPED states.

Table 4-4 describes the states supported for the TRAPDEST object.

Table 4-4. TRAPDEST Object States

State	Description
DEFINED	The TRAPDEST object has been configured with the SCF ADD command, but has not been initially activated for use in the NonStop agent operations environment.
STARTING	The TRAPDEST object is in the process of starting. If an underlying resource is unavailable, it remains in the STARTING state until the resource becomes available. At that time it enters the STARTED state.
STARTED	The TRAPDEST object is either ready to send or in the process of sending trap messages.
STOPPED	The TRAPDEST object is configured but is not currently being used by the NonStop agent. The object has been explicitly stopped with the SCF STOP or ABORT command.

Figure 4-10 illustrates the state transition sequence for the TRAPDEST object.

TRAPDEST Object Attributes

The following attributes are supported for the TRAPDEST object:

- COMMUNITY specifies the community name included in trap messages sent to the associated manager.
- HOSTADDR specifies the Internet address of the manager to which trap messages are to be sent.
- NETWORK specifies the TCP/IP process handling the sending of trap messages by the NonStop agent process to the manager.

For a description of these attributes, see the ADD Command on page 5-5.

Default TRAPDEST Objects

When you initially start the NonStop agent process with the default WARM (see WARM | COLD Custom Configuration Parameters for the RUN Command on page 2-14) and TCPIP^PROCESS^NAME options, the NonStop agent creates a default TRAPDEST object for each Internet address from which it receives a request.

These dynamically generated TRAPDEST objects are named “#DYNA0,” “#DYNA1,” and so on. They are stored in the SNMPCTL file.

For example, if the NonStop agent receives a request from the manager at Internet address “130.252.86.10,” it creates and starts the following TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA0, COMMUNITY "Tandem",
  NETWORK $ZTC0, HOSTADDR "130.252.86.10"
```

Then, if the NonStop agent receives a request from the manager at Internet address “155.186.130.123,” it creates and starts another TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA1, COMMUNITY "Tandem",
  NETWORK $ZTC0, HOSTADDR "155.186.130.123"
```

Dynamic TRAPDEST generation continues until you either:

- ADD a TRAPDEST object (or create one through SNMP).
- Stop and restart the NonStop agent with the WARM option after at least one default TRAPDEST object has been generated

For more information on the circumstances under which dynamic TRAPDEST generation is enabled and disabled, see The TRAPDEST Object on page 4-16.

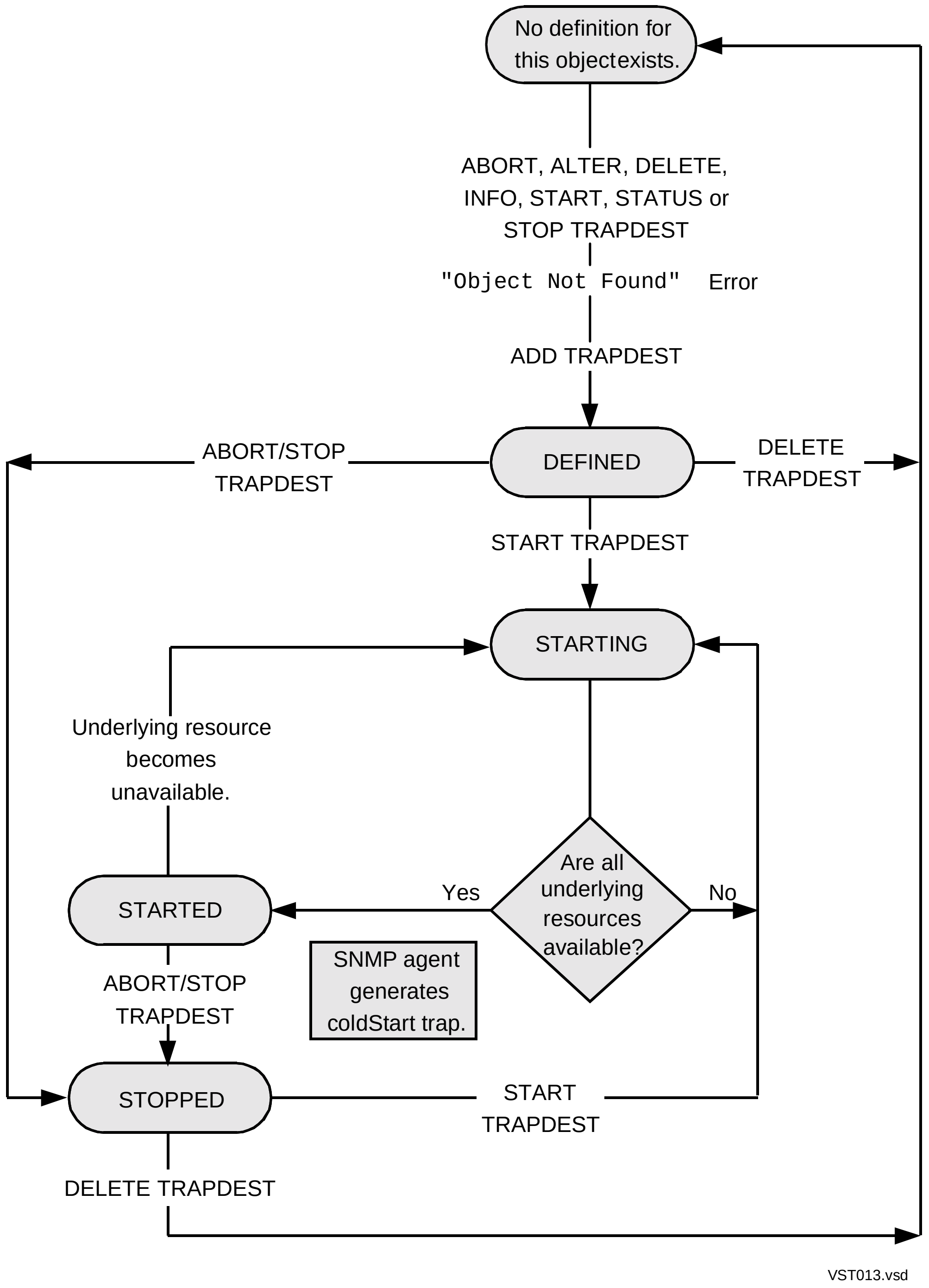
SCF-Configured Trap Destination Limitations: Broadcast Only

The NonStop agent supports two types of trap destinations, **directed** and **broadcast**:

- The NonStop agent sends traps to directed trap destinations only when specifically told to do so by a subagent. Directed trap destinations only receive traps that are specifically directed at them.
- The NonStop agent sends traps to broadcast trap destinations when forwarding traps not directed to a directed trap destination. Broadcast trap destinations receive all traps that are not specifically directed at a particular trap destination.

Trap destinations created with SCF are only of type broadcast. The SCF interface does not provide a means of creating directed trap destinations. However, a manager can change the type of a trap destination configured through SCF from type broadcast to type directed.

Figure 4-10. TRAPDEST Object State Transition Sequence


No definition for this objectexists.
ABORT, ALTER, DELETE, INFO, START, STATUS or STOP TRAPDEST
"Object Not Found" Error
ADD TRAPDEST
ABORT/STOP TRAPDEST
DEFINED
DELETE TRAPDEST
START TRAPDEST
STARTING
Underlying resource becomes unavailable.
Are all underlying resources available?
Yes
No
STARTED
SNMP agent generates coldStart trap.
ABORT/STOP TRAPDEST
STOPPED
START TRAPDEST
DELETE TRAPDEST
VST013.vsd

Running SCF

You can run SCF interactively or noninteractively. SCF can accept input from either a terminal or a command file (OBEY file) and can direct output to either a terminal, disk file, or printer.

In its simplest form, the syntax for running SCF from TACL is as follows:

```
[ RUN ] [[$vol.]subvol.]SCF [ /run-options/ ]
```

For detailed information on running SCF, its valid run options, and input sources and output destinations, see either the *SCF Reference Manual for G-Series Releases* (for systems running a G-series RVU of the NonStop Kernel) or the *SCF Reference Manual for H-Series RVUs* (for systems running an H-series RVUs of the NonStop Kernel).

Note. In this manual, all examples assume that a terminal is being used for both input and output.

At the beginning of an SCF session, SCF displays its product banner. The product banner includes the HP product name, product number, version number, release date, and copyright statement.

SCF indicates it is ready to process commands by displaying its prompt (`->`). You can change this prompt with the SETPROMPT command.

SCF waits for a command, followed by a return. After it receives and processes the command, SCF displays its prompt for the next command.

Entering SCF Commands

An SCF command always begins with a keyword identifying the command (such as START, VOLUME, or TRACE).

Some SCF commands let you control your SCF session. Others allow you to manage and inquire about objects in particular HP subsystems.

The HELP command displays information about SCF sessions in general and also about SCF as it applies to specific subsystems. For more information, see SCF Online Help on page 4-24.

SCF Commands for Managing Your SCF Session

To issue a command that pertains to your current SCF session, follow the command with whatever additional information is required to direct the action. For example, the following SETPROMPT command causes SCF to display the current system name in its prompt:

```
-> SETPROMPT SYSTEM
\EAST->
```

Examples of other SCF commands that pertain only to the current SCF session:

- The SYSTEM and VOLUME commands identify the default system, volume, and subvolume names used for expanding file names.
- The OBEY and OUT commands identify the files used for command input and display output.
- The ASSUME command defines a default object to be used when the object is omitted from an SCF command.
- The ENV command displays the current settings of the SCF command parameters that establish the program environment.

SCF commands that pertain only to the SCF session appear in the *SCF Reference Manual for G-Series Releases*.

SCF Commands for Managing a Subsystem

To issue a command to manage components of a subsystem, follow the command with an object type and an object name. For example, the following STATUS command displays the state of the NonStop agent TRAPDEST object named $ZSNMP.#TRAP:

```
-> STATUS TRAPDEST $ZSNMP.#TRAP

SNMP Status TRAPDEST

Name                      State
$ZSNMP.#TRAP              STARTED
```

Note. You do not explicitly indicate SCF the subsystem with which you are communicating. SCF determines the subsystem by interpreting the object type and object name you supply in the command line.

When including command modifiers (options that affect the scope of the command) or object attributes (to define characteristics of an object), separate each keyword-value pair from the object name with a comma. For example:

```
-> INFO PROCESS $ZSNMP, SUB All, DETAIL
-> ADD TRAPDEST $ZSNMP.#TRAP, HOSTADDR "130.252.85.199"
```

Examples of other SCF commands used to manage subsystem objects:

- The ADD and DELETE commands, which add and delete object configurations to and from the SCF sphere of control for a particular subsystem.
- The START, STOP, and ABORT commands, which cause a configured object to become active or inactive; and the ALTER command, which changes the configuration of an object.
- The INFO command, which displays the current configuration values for an object.
- The STATUS command, which displays the current operational state of an object.

- The TRACE command, which traces the operation of an object according to selection criteria you specify and stores the trace information in a trace file.

Sensitive and Nonsensitive Commands

SCF commands fall into two categories—sensitive and nonsensitive:

- A sensitive command can have detrimental effects if improperly used. These commands can be issued only by a user with super-group access, by the owner of the subsystem, or by a member of the group of the owner of the subsystem.
- A nonsensitive command requests information or status but does not affect operation. These commands are available to all users.

Command Modifiers

Some SCF commands accept modifiers that affect the scope of the command. For example:

- You can use the SUB (subordinate) command modifier to specify that a command be applied to an object (SUB NONE), to an object and its subordinates (SUB ALL), or only to the subordinates of an object and not the object itself (SUB ONLY). For example:

  ```
  INFO PROCESS $ZSNMP, SUB ALL
  ```

- You can use the DETAIL option to request that detailed information be returned in response to a request for information. For example:

  ```
  VERSION $ZSNMP, DETAIL
  ```

For the command modifiers valid for each NonStop agent SCF command, see the syntax descriptions in Section 5, SCF Commands for the SNMP Agent.

Entering Multiple Commands at a Prompt

To enter multiple SCF commands at a single prompt, separate the commands with semicolons. For example:

```
-> ASSUME PROCESS $ZSNMP;STATUS ENDPOINT #SOCKET1
```

SCF executes the commands one at a time from left to right.

Continuing a Command to the Next Line

To continue a command that starts on one line onto a second line, terminate the first line with an ampersand (&). SCF prompts for additional input before executing the command. For example:

```
-> ADD TRAPDEST $ZSNMP.#SUN1, NETWORK $ZTC0, &
-> COMMUNITY "T16", HOSTADDR "130.252.85.199"
```

Do not enter more than 2048 characters for any input command.

Directing Output to a File

To direct output of all SCF commands to a disk file, application process, terminal, or printer, include after the command keyword:

```
/OUT file-spec/
```

where *file-spec* is the file to which the output listing is to be written. If *file-spec* has the form of a disk-file name but the file does not exist, an EDIT file is created. If the name of the file is an existing disk file, the output is appended to the file. For example:

```
-> INFO /OUT $DATA.SNMP.SCFINFO/ PROCESS $ZSNMP, SUB ALL
```

Error Messages

If an **error** occurs during the processing of a command, SCF displays the appropriate error message and ignores the rest of the line. SCF generates the following types of error messages:

- Command parsing error messages
- Common error messages (common to all subsystems)
- SCF-generated error messages
- Subsystem-specific error messages

For SCF error messages, see Appendix B, SCF Error Messages for the NonStop Agent.

SCF Online Help

You access the interactive SCF online help facility with the HELP command. When you enter the HELP command with no options, SCF displays a menu that guides you through the available help options.

You can obtain general information about SCF such as which subsystems and object types it supports, and how to specify attributes, character strings, commands, integers, and file names.

Displaying Help for NonStop Agent SCF Commands

For help specific to the NonStop agent:

```
-> HELP SNMP
```

For information on the syntax of NonStop agent object types and the commands that apply to them:

```
-> HELP SNMP [ command ] [ object-type ]
```

For example, the following request returns information about the ADD command that is specific to the NonStop agent:

```
-> HELP SNMP ADD
```

The next request returns information about the NonStop agent TRAPDEST object:

```
-> HELP SNMP TRAPDEST
```

This request returns information about the ADD command when it is issued against a TRAPDEST object:

```
-> HELP SNMP ADD TRAPDEST
```

Displaying Help for SCF Error Messages

To request help for SCF error messages:

```
-> HELP subsystem error-number
```

For example, suppose the following messages appear on your terminal:

```
SNMP E00001 Attribute conflict: HOSTADDR
SNMP E-00015 Object $ZSNMP.#END1 Already Defined
SCF E20211 Invalid object type
```

To get help:

```
-> HELP SNMP 1
-> HELP SNMP -15
-> HELP SCF 20211
```

SCF Information Specific to the NonStop Agent

SNMP defines a standard way of managing devices from multiple vendors on a TCP/IP-based network. An SNMP network management environment consists of one or more SNMP managers, one or more SNMP agents, and the network management information that is retrieved and manipulated by the agents on behalf of the managers. The NonStop agent provides the means by which HP customers operating multiplatform networks can have HP subsystems managed from SNMP-compliant network managers. The SNMP agent that HP provides is referred to as the NonStop agent.

The NonStop agent interacts with subagents that reside on the same or remote NonStop systems. SNMP subagents are responsible for handling a particular collection of resources. Each subagent handles manager requests forwarded by the NonStop agent pertaining to the hardware or software resources for which the subagent is responsible.

Almost all configuration and management of the NonStop agent can be done through SCF. Much of the NonStop agent configuration can also be done through SNMP. In addition, some special case configuration is done through startup parameters when you start the NonStop agent process. For startup parameters and configuration through SNMP, see Configuring the SNMP Agent Through SNMP Requests on page 2-17.

Note. Currently, subagents are not configured through SCF. They are configured through startup parameters as well as through SNMP requests issued by SNMP managers. For configuration of SNMP subagents that HP supports, see the Part IV, SNMP Subagents.

The SCF Product Module for the NonStop Agent

Each subsystem has a product module associated with it that defines its SCF interface.

The NonStop agent product module is a code 0 file named ZSMPSCF. ZSMPSCF is placed in the $SYSTEM.SYSTEM subvolume when you install the NonStop agent product with the Install or DSM/SCM product.

When you issue an SCF LISTPM (list product modules) command, the NonStop agent is identified as SNMP in the Subsystem column.

Before You Can Issue SCF Commands Against the NonStop Agent

You must know the name under which the NonStop agent process is running (the name of the PROCESS object) before you can issue SCF commands to retrieve information or perform any configuration or management tasks for the NonStop agent. If you know the name of the NonStop agent process, you can find out about the current NonStop agent configuration by issuing SCF INFO and STATUS commands and including the SUB ALL command modifier.

The NonStop agent process is often started under the name $ZSNMP. If you do not know what your NonStop agent process is named, try issuing an SCF STATUS PROCESS $ZSNMP command. If this command is not successful, you might use the TACL STATUS command with the PROG option to find out the name of the NonStop agent process. As installed, the program file for the NonStop agent process is named SNMPAGT. For example:

```
2> STATUS *, PROG $NETMAN.SNMP.SNMPAGT
Process             Pri PFR %WT Userid  Program file            Hometerm
$ZJMC       3,48    149     015 255,209 $NETMAN.SNMP.SNMPAGT      $T100.#TRM4
```

SCF Configuration and the TCPIP^PROCESS^NAME Startup Parameter

The NonStop agent has a startup parameter—TCPIP^PROCESS^NAME—that allows you to specify the TCP/IP process to be used as the default value for the NETWORK attribute when ENDPOINT and TRAPDEST objects are created. If you do not include the TCPIP^PROCESS^NAME startup parameter when you start the NonStop agent, the default TCP/IP process is $ZTC0 on the local node.

5 SCF Commands for the SNMP Agent

This section describes the SCF commands with which you manage a NonStop agent configuration.

The supported commands are:

Table 5-1. Summary of SCF Commands Supported for NonStop Agent Objects

		Object Type for Which Command Is Supported				
Command	**Sensitive?**[1]	**PROCESS**	**PROFILE**	**ENDPOINT**	**TRAPDEST**	***null***[2]
ABORT	Yes		X	X	X	
ADD	Yes		X	X	X	
ALTER	Yes	X	X	X	X	
DELETE	Yes		X	X	X	
INFO	No	X	X	X	X	
NAMES	No	X				X
START	Yes		X	X	X	
STATUS	No	X	X	X	X	
STOP	Yes		X	X	X	
TRACE	Yes	X				
VERSION	No	X				X

[1] A sensitive command can have detrimental effects if improperly used. These commands can be issued only by a user with super-group access, by the owner of the subsystem, or by a member of the group of the owner of the subsystem.

[2] Use the *null* object when the object type is irrelevant. You can issue SCF commands that support the *null* object without specifying an object type. You must still include the object name.

SCF commands for controlling your SCF session, such as the ASSUME and ENV commands, are not documented in this section. The *SCF Reference Manual for G-Series Releases* provides information about general SCF commands.

In the following syntax diagrams, remember that if you set a default PROCESS with the ASSUME PROCESS command, you can omit the NonStop agent process name and period and just specify `#`*`object-name`*. For example:

```
-> ASSUME PROCESS $ZSNMP
-> INFO ENDPOINT #END1
-> STATUS ENDPOINT #END1
```

Configuration Restrictions: Attribute Conflicts

The NonStop agent does not allow conflicting object definitions to coexist in the same operations environment. All defined objects must be capable of being started concurrently. If you want to add new objects or alter existing objects, you must first stop and delete or stop and alter any other current object definitions whose attribute values conflict with the values of the object you are creating.

Attribute Conflicts Within a NonStop Agent Configuration

Object definitions not allowed within a NonStop agent configuration:

- Two ENDPOINT objects cannot use the same TCP/IP Internet address (same NETWORK attribute and same or overlapping HOSTADDR attribute values).
- Two PROFILE objects cannot be defined with the same community (same COMMUNITY attribute value).

When you try to add the conflicting object definition within a NonStop agent configuration with the SCF ADD command, SCF returns an “Attribute Conflict” error See Appendix B, SCF Error Messages for the NonStop Agent.

Attribute Conflicts Between NonStop Agent Configurations

Attribute conflicts between NonStop agent configurations refer to operations environments in which more than one NonStop agent process is being run, either on the same node or on different nodes in a network.

Two NonStop agents cannot be defined with ENDPOINT objects that use the same TCP/IP Internet address or that use an address that is a member of a currently started full wildcard (“0.0.0.0”) address set.

Attribute conflicts between NonStop agent configurations manifest themselves when you try to activate the conflicting object definition with the SCF START command. SCF returns a “Resource Not Available” error.

ABORT Command

The ABORT command:

- Inactivates an object definition in the current NonStop agent configuration
- Behaves the same way as the STOP command for NonStop agent objects
- Is not valid for the PROCESS object
- Is a sensitive command

ABORT ENDPOINT Command

Use the ABORT ENDPOINT command to inactivate a request/response connection between a NonStop agent process and managers.

```
ABORT ENDPOINT [$agent-process.]#endpoint-name
```

`[$`*`agent-process`*`.]#`*`endpoint-name`*

identifies the request/response connection being inactivated.

`$`*`agent-process`*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`endpoint-name`*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: ABORT ENDPOINT

- When you ABORT an ENDPOINT object, you only affect the availability of the TCP/IP subnets for handling communication between the NonStop agent and managers. The TCP/IP process and subnets are still available for handling communication between other entities.
- If the NonStop agent is processing an SNMP request when an ABORT command is issued, the ABORT command waits until processing is finished, at which time the ENDPOINT object enters the STOPPED state.

Example: ABORT ENDPOINT

The following example closes communication between NonStop agent process $ZSNMP and managers through the request/response connection defined by the #END1 object:

```
-> ABORT ENDPOINT $ZSNMP.#END1
```

ABORT PROFILE Command

Use the ABORT PROFILE command to inactivate an authentication table entry.

```
ABORT PROFILE [$agent-process.]#profile-name
```

[$*agent-process*.]#*profile-name*

identifies the authentication table entry being inactivated.

$*agent-process*

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#*profile-name*

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: ABORT PROFILE

When you ABORT a PROFILE object, you inactivate but do not delete the corresponding authentication table entry. To reactivate the table entry so that it is available for use by the NonStop agent, issue a START PROFILE command against the object.

Example: ABORT PROFILE

The following example inactivates an entry in the authentication table used by NonStop agent process $ZSNMP:

```
-> ABORT PROFILE $ZSNMP.#LANMGR
```

ABORT TRAPDEST Command

Use the ABORT TRAPDEST command to disable the sending of trap messages to a specific trap destination.

```
ABORT TRAPDEST [$agent-process.]#trapdest-name
```

`[$`*`agent-process`*`.]#`*`trapdest-name`*

identifies the trap destination being inactivated.

`$`*`agent-process`*

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`trapdest-name`*

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: ABORT TRAPDEST

Traps being generated by or received for forwarding by the NonStop agent when a TRAPDEST object is aborted are not sent to that particular destination.

Example: ABORT TRAPDEST

The following example disables the sending of trap messages by NonStop agent process $ZSNMP to all its dynamically configured trap destinations:

```
-> ABORT TRAPDEST $ZSNMP.#DYNA*
```

ADD Command

The ADD command:

- Defines an object for use in a NonStop agent configuration
- Requires that if you include an attribute keyword, you also include a value
- Uses the default value for an attribute if you omit the attribute keyword/value pair
- Is a sensitive command

ADD ENDPOINT Command

Use the ADD ENDPOINT command to define a request/response connection between a NonStop agent process and managers.

```
ADD ENDPOINT [$agent-process.]#endpoint-name
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address"]
```

`[$agent-process.]#endpoint-name`

identifies the request/response connection being defined.

`$agent-process`

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#endpoint-name`

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`NETWORK [\node.]$tcpip-process`

identifies the TCP/IP process handling communication between the NonStop agent process and managers for the receiving and sending of SNMP request/response messages.

`$tcpip-process` is from one to five alphanumeric characters preceded by a dollar sign ($). The first alphanumeric character must be a letter. The process name can optionally be qualified with a node name (backslash followed by one to seven alphanumeric characters, the first of which must be a letter) identifying the node on which the process is running.

The default is the TCP/IP process specified in the TCPIP^PROCESS^NAME startup parameter when the NonStop agent process was started or, if none was specified, $ZTC0 on the local node.

Do not specify a TCPSAM or TPC6SAM process for this attribute if the process runs on a different node or allow the attribute to default to such a process.

`HOSTADDR "ip-address"`

specifies the Internet addresses by which the NonStop agent is known to the network. In other words, HOSTADDR specifies the TCP/IP subnet addresses through which the NonStop agent listens for request messages from managers.

Enter `"ip-address"` in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: 133.50.85.43.

The default (“0.0.0.0”) indicates that the NonStop agent accepts requests from and returns responses through any available TCP/IP subnet defined for the specified TCP/IP process.

Considerations: ADD ENDPOINT

- The TCP/IP process specified in the NETWORK attribute does not have to be on the same node as the NonStop agent process.
- The value 0 is recognized as a wildcard in the host addressing scheme for the HOSTADDR attribute. The ENDPOINT object only accepts a full wildcard specification (“0.0.0.0”) or a distinct Internet address. For example: 130.50.85.43.
- If you start a NonStop agent process with its default WARM and TCPIP^PROCESS^NAME startup parameters (and name the process $ZSNMP), an ENDPOINT object called $ZSNMP.#DEFAULT with the following attribute values is automatically added and started:

  ```
  ENDPOINT $ZSNMP.#DEFAULT, NETWORK $ZTC0, HOSTADDR "0.0.0.0"
  ```

  By default, then, the NonStop agent receives and sends SNMP requests through any available subnet associated with the TCP/IP process $ZTC0 on the same node as the NonStop agent process is running.
- Two ENDPOINT objects cannot use the same TCP/IP Internet address or use an address that is a member of a full wildcard address specification in an existing ENDPOINT object.

Examples: ADD ENDPOINT

- The following example defines a configuration entry named #END1 that, when activated with the START command, allows NonStop agent process $ZSNMP to be reached at Internet address 130.50.85.43 through TCP/IP process $ZTC1:

  ```
  -> ADD ENDPOINT $ZSNMP.#END1, NETWORK $ZTC1, &
  -> HOSTADDR "130.50.85.43"
  ```
- Because default values are assumed when an attribute keyword/value pair is omitted, the following SCF commands have the same result:

  ```
  -> ADD ENDPOINT $ZSNMP.#END1, HOSTADDR "130.50.85.43"

  -> ADD ENDPOINT $ZSNMP.#END1, NETWORK $ZTC0, &
  -> HOSTADDR "130.50.85.43"
  ```

Note. Do not specify a TCPSAM or TPC6SAM process for this attribute if that process runs on a different node

ADD PROFILE Command

Use the ADD PROFILE command to define an authentication table entry.

```
ADD PROFILE [$agent-process.]#profile-name
   [ , COMMUNITY "community-name" ]
   [ , HOSTADDR "ip-address"]
   [ , ACCESS { READONLY | READWRITE } ]
```

`[$`*`agent-process`*`.]#`*`profile-name`*

identifies the authentication table entry being defined.

`$`*`agent-process`*

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`profile-name`*

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`COMMUNITY "`*`community-name`*`"`

specifies the community name that must be present in requests sent by managers for the NonStop agent to accept the request.

`"`*`community-name`*`"` is a case-sensitive of 1 to 50 character enclosed string within quotation marks.

The default is “public.”

`HOSTADDR "`*`ip-address`*`"`

specifies the Internet address of the managers whose security is being defined in the authentication table entry.

Enter `"`*`ip-address`*`"` in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: 133.50.85.43.

The default (“0.0.0.0”) indicates that the authentication table entry applies to all managers.

`ACCESS { READONLY | READWRITE }`

specifies the level of authority of an associated community to retrieve and alter network management information:

READONLY	The NonStop agent accepts Get and GetNext requests from the associated community. Members of the community can only retrieve information.
READWRITE	The NonStop agent accepts Set, Get, and GetNext requests from the associated community. Members of the community can alter as well as retrieve information.

The default is READONLY.

Considerations: ADD PROFILE

- The value 0 is recognized as a wildcard in the host addressing scheme for the HOSTADDR attribute. The PROFILE object only accepts a full wildcard specification (“0.0.0.0”) or a distinct Internet address (for example, “130.50.85.100”).
- Each request that a NonStop agent receives from an SNMP manager includes a community string composed of one or two discrete sections delimited by two colons (`::`) as follows:

  `community-name[::subagent-password]`

  The community string included in incoming requests is part of the SNMP manager’s configuration. The NonStop agent only compares the *`community-name`* portion of the incoming request’s community string with the COMMUNITY attribute values in its authentication table. Once an authenticated request has been forwarded to a subagent, the subagent might employ additional security based on the *`subagent-password`* portion of the community string.
- If you start a NonStop agent process with its default WARM startup parameter (and name the process $ZSNMP), a PROFILE object called $ZSNMP.#DEFAULT with the following attribute values is automatically added and started:

  ```
  PROFILE $ZSNMP.#DEFAULT, COMMUNITY "public",
    HOSTADDR "0.0.0.0", ACCESS READONLY
  ```

  Unless you explicitly stop this PROFILE object, the NonStop agent continues to accept any incoming request originating from any manager belonging to the “public” community, regardless of any other table entries you define to tighten security.
- You must configure an authentication table entry for each community other than “public” from which the NonStop agent is to receive SNMP requests.
- Two PROFILE objects cannot be defined with the same community (same COMMUNITY attribute value).

Examples: ADD PROFILE

- The following example configures and activates an entry in the authentication table that directs the NonStop agent to process SNMP Set, Get, and GetNext requests from the manager at Internet address “130.252.86.10” sent under the “Private” community:

  ```
  -> ADD PROFILE $ZSNMP.#LANMGR, COMMUNITY "Private", &
  -> ACCESS READWRITE, HOSTADDR "130.252.86.10"
  -> START PROFILE $ZSNMP.#LANMGR
  ```

  Unless you have explicitly inactivated (with the STOP or ABORT command) the delivered PROFILE $ZSNMP.#DEFAULT entry, the NonStop agent also processes Get and GetNext requests generated under the “public” community from this manager.

- Because default values are assumed when an attribute keyword/value pair is omitted, the following SCF commands have the same result:

  ```
  -> ADD PROFILE $ZSNMP.#UBLAN, COMMUNITY "GetComm"

  -> ADD PROFILE $ZSNMP.#UBLAN, COMMUNITY "GetComm", &
  -> HOSTADDR "0.0.0.0", ACCESS READONLY
  ```

ADD TRAPDEST Command

Use the ADD TRAPDEST command to define a manager to which traps are to be sent (trap destination).

```
ADD TRAPDEST [$agent-process.]#trapdest-name
   [ , COMMUNITY "community-name" ]
   [ , NETWORK [\node.]$tcpip-process ]
   , HOSTADDR "ip-address"
```

`[$`*`agent-process`*`.]#`*`trapdest-name`*

identifies the trap destination being defined.

`$`*`agent-process`*

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`trapdest-name`*

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`COMMUNITY "`*`community-name`*`"`

specifies the community name included in trap messages sent to the associated manager.

`"`*`community-name`*`"` is a case-sensitive 1 to 50 character enclosed string within quotation marks.

The default is “Tandem.”

`NETWORK [\`*`node`*`.]$`*`tcpip-process`*

specifies the TCP/IP process handling the sending of trap messages by the NonStop agent process to the manager.

`$`*`tcpip-process`* is from one to five alphanumeric characters preceded by a dollar sign ($). The first alphanumeric character must be a letter. The process name can optionally be qualified with a node name (backslash followed by one to seven alphanumeric characters, the first of which must be a letter) identifying the node on which the process is running.

The default is the TCP/IP process specified in the TCPIP^PROCESS^NAME startup parameter when the NonStop agent process was started. If none was specified, $ZTC0 on the local node

`HOSTADDR "`*`ip-address`*`"`

specifies the Internet address of the manager to which trap messages are to be sent.

Enter `"`*`ip-address`*`"` in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: “133.50.85.43”.

There is no default value for the HOSTADDR attribute: when issuing an ADD TRAPDEST command, you must specify a HOSTADDR keyword/value pair.

Considerations: ADD TRAPDEST

- The TCP/IP process configured for the sending of traps does not have to reside on the same node as the NonStop agent process.
- The TRAPDEST object does not recognize the value 0 as a wildcard in the host addressing scheme. You must provide a unique Internet address (for example, “130.50.85.43”) for each TRAPDEST object.
- There are no restrictions regarding the number of TRAPDEST objects you can define. SCF generates a warning when you configure more than one TRAPDEST object pointing to the same destination but accepts the object definition. Defining duplicate TRAPDEST objects results in duplicate traps being sent to the same address.

- Trap destinations created with SCF are only of type broadcast. The SCF interface does not provide a means of creating directed trap destinations. However, a manager can change the type of a trap destination defined through SCF from type broadcast to type directed. For more information, see the Part II, SCF for the SNMP Agent.

Dynamically Generated TRAPDEST Objects

When you start the NonStop agent with the WARM startup parameter, and no TRAPDEST objects are defined in the SNMPCTL file, the NonStop agent creates a trap destination definition with the default NETWORK and COMMUNITY attribute values for each Internet address from which it receives a request.

These dynamically generated TRAPDEST objects are named “#DYNA0,” “#DYNA1,” and so on. They are stored in the SNMPCTL file.

So, for example, if the NonStop agent receives a request from the manager at Internet address “130.252.86.10,” it creates and starts the following TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA0, COMMUNITY "Tandem",
  NETWORK $ZTC0, HOSTADDR "130.252.86.10"
```

Then, if the NonStop agent receives a request from the manager at Internet address “155.186.130.123,” it creates and starts another TRAPDEST object:

```
TRAPDEST $agent-process.#DYNA1, COMMUNITY "Tandem",
  NETWORK $ZTC0, HOSTADDR "155.186.130.123"
```

Once you explicitly create a TRAPDEST object with the ADD command, dynamic TRAPDEST generation stops.

The following summarizes dynamic TRAPDEST object generation:

- Dynamic TRAPDEST generation is enabled when the NonStop agent is started with the WARM startup parameter and no TRAPDEST objects are in the SNMPCTL file.
- Dynamic TRAPDEST generation is disabled when you either:
    - Use the ADD command to add a TRAPDEST object or create one through SNMP.
    - Stop and restart the NonStop agent with the WARM option after at least one default TRAPDEST object has been generated.

      If you stopped and then restarted the NonStop agent with the WARM option, even if you have never explicitly added a TRAPDEST object, no new TRAPDEST objects are dynamically generated. The SNMPCTL file will contain the #DYNA*n* objects created when the NonStop agent was previously running.

Examples: ADD TRAPDEST

- The following example tells the NonStop agent to send trap messages to the manager at Internet address “130.252.86.10.” The NonStop agent includes the community string “Public” in all trap messages it sends to this address. This example does not include the NETWORK attribute keyword and value because it is using the default $ZTC0:

```
-> ADD TRAPDEST $ZSNMP.#TRAP, COMMUNITY "Public", &
-> HOSTADDR "130.252.86.10"
```

- Because default values are assumed when an attribute keyword/value pair is omitted, the following SCF commands have the same result:

```
-> ADD TRAPDEST $ZSNMP.#TRAP, HOSTADDR "130.252.86.10"

-> ADD TRAPDEST $ZSNMP.#TRAP, COMMUNITY "Tandem", &
-> NETWORK $ZTC0, HOSTADDR "130.252.86.10"
```

ALTER Command

The ALTER command:

- Changes attribute values associated with an object
- Requires that if you include an attribute keyword, you also include a value
- Is a sensitive command

ALTER ENDPOINT Command

Use the ALTER ENDPOINT command to change the TCP/IP process or Internet address in a request/response connection between a NonStop agent process and managers.

```
ALTER ENDPOINT [$agent-process.]#endpoint-name
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address" ]
```

`[$`*`agent-process`*`.]#`*`endpoint-name`*

identifies the request/response connection being altered.

`$`*`agent-process`*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`endpoint-name`*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`NETWORK [\`*`node`*`.]$`*`tcpip-process`*

identifies the TCP/IP process handling communication between the NonStop agent process and managers for the receiving and sending of SNMP request/response messages.

`$`*`tcpip-process`* is from one to five alphanumeric characters preceded by a dollar sign ($). The first alphanumeric character must be a letter. The process name can optionally be qualified with a node name (backslash followed by one to seven alphanumeric characters, the first of which must be a letter) identifying the node on which the process is running.

Do not specify a TCPSAM or TPC6SAM process for this attribute if the process runs on a different node or allow the attribute to default to such a process.

`HOSTADDR "`*`ip-address`*`"`

specifies the Internet addresses by which the NonStop agent is known to the network.

Enter `"`*`ip-address`*`"` in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: “133.50.85.43”

Considerations: ALTER ENDPOINT

The ENDPOINT object must be in the STOPPED or DEFINED state before you can alter it.

For ENDPOINT configuration restrictions, see Considerations: ADD ENDPOINT on page 5-7.

Example: ALTER ENDPOINT

The following example closes, alters the configuration of, and reopens communication between NonStop agent process $ZSNMP and managers through Internet address 130.50.85.55:

```
-> ASSUME PROCESS $ZSNMP
-> STOP ENDPOINT #END1
-> ALTER ENDPOINT #END1, HOSTADDR "130.50.85.55"
-> START #END1
```

ALTER PROCESS Command

Use the ALTER PROCESS command to direct the NonStop agent process to send its event messages to a different EMS collector process.

```
ALTER PROCESS $agent-process
   [ , EMSCOLL [\node.]$ems-collector ]
```

`$agent-process`

identifies the NonStop agent process whose configuration is being altered. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`EMSCOLL [\node.]$ems-collector`

specifies the EMS collector process to which the NonStop agent is to send events it generates.

$ems-collector is one to five alphanumeric characters preceded by a dollar sign ($). The first alphanumeric character must be a letter. The process name can optionally be qualified with a node name (backslash followed by one to seven alphanumeric characters, the first of which must be a letter) identifying the node on which the process is running.

Considerations: ALTER PROCESS

- You can issue an ALTER command against a PROCESS object while the object is in the STARTED state.
- In the event of a takeover by the backup NonStop agent process, the backup process also sends its events to the EMS collector identified in the current EMSCOLL setting.
- The value of the EMSCOLL attribute is not saved in the SNMPCTL file (in which the NonStop agent maintains values for its current configuration). If you restart a NonStop agent process, the NonStop agent sends its events to the default EMS collector, $0, regardless of the EMSCOLL setting in effect when the agent was stopped.
- You can also specify that the NonStop agent send its events to an alternate collector by including a COLLECTOR startup parameter when you start the NonStop agent process. Including the COLLECTOR startup parameter with no argument is the only way you can suppress output of NonStop agent events. You cannot accomplish this through SCF.

  For information on using the COLLECTOR startup parameter rather than the EMSCOLL attribute, see RUN Command on page 2-7.

Example: ALTER PROCESS

The following example tells the NonStop agent process $ZSNMP to send event messages it generates to EMS collector process $A0:

```
-> ALTER PROCESS $ZSNMP, EMSCOLL $A0
```

ALTER PROFILE Command

Use the ALTER PROFILE command to change an entry in the authentication table.

```
ALTER PROFILE [$agent-process.]#profile-name
   [ , COMMUNITY "community-name" ]
   [ , HOSTADDR "ip-address"]
   [ , ACCESS { READONLY | READWRITE } ]
```

`[$`*`agent-process`*`.]#`*`profile-name`*

identifies the authentication table entry being altered.

`$`*`agent-process`*

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`profile-name`*

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`COMMUNITY "`*`community-name`*`"`

specifies the community name that must be present in requests sent by managers for the NonStop agent to accept the request.

`"`*`community-name`*`"` is a case-sensitive of 1 to 50 character enclosed string within quotation marks.

`HOSTADDR "`*`ip-address`*`"`

specifies the Internet address of the managers whose security is being defined in the authentication table entry.

Enter `"`*`ip-address`*`"` in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: “133.50.85.43”

`ACCESS { READONLY | READWRITE }`

specifies the level of authority of an associated community to retrieve and alter network management information as follows:

READONLY	The NonStop agent accepts Get and GetNext requests from the associated community. Members of the community can only retrieve information.
READWRITE	The NonStop agent accepts Set, Get, and GetNext requests from the associated community. Members of the community can alter as well as retrieve information.

Considerations: ALTER PROFILE

The PROFILE object must be in the STOPPED or DEFINED state before you can alter it.

For PROFILE configuration restrictions, see Considerations: ADD PROFILE on page 5-9.

Example: ALTER PROFILE

The following example inactivates, alters, and reactivates the #UBLAN entry in the authentication table being used by NonStop agent process $ZSNMP:

```
-> ASSUME PROCESS $ZSNMP
-> STOP PROFILE #UBLAN
-> ALTER PROFILE #UBLAN, ACCESS READONLY
-> START PROFILE #UBLAN
```

ALTER TRAPDEST Command

Use the ALTER TRAPDEST command to change a trap destination definition.

```
ALTER TRAPDEST [$agent-process.]#trapdest-name
   [ , COMMUNITY "community-name" ]
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address"]
```

[$*agent-process*.]#*trapdest-name*

identifies the trap destination definition being altered.

$*agent-process*

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#trapdest-name

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`COMMUNITY "`*`community-name`*`"`

specifies the community name included in trap messages sent to the associated manager.

"community-name" is a case-sensitive of 1 to 50 character enclosed string within quotation marks.

`NETWORK [\`*`node`*`.]$`*`tcpip-process`*

specifies the TCP/IP process handling the sending of trap messages by the NonStop agent process to the manager.

$*tcpip-process* is from one to five alphanumeric characters preceded by a dollar sign ($); the first alphanumeric character must be a letter. The process name can optionally be qualified with a node name (backslash followed by one to seven alphanumeric characters, the first of which must be a letter) identifying the node on which the process is running.

`HOSTADDR "`*`ip-address`*`"`

specifies the Internet address of the manager to which trap messages are to be sent.

Enter "*ip-address*" in standard Internet address dotted-decimal format—consisting of the ASCII values of four bytes separated by periods—enclosed within quotation marks. For example: "133.50.85.43"

Considerations: ALTER TRAPDEST

The TRAPDEST object must be in the STOPPED or DEFINED state before you can alter it.

For TRAPDEST configuration restrictions, see Considerations: ADD TRAPDEST on page 5-11.

Example: ALTER TRAPDEST

The following example disables, alters the configuration of, and reenables the sending of trap messages by NonStop agent process $ZSNMP to the manager defined by the #SUNTRAP object:

```
-> ASSUME PROCESS $ZSNMP
-> STOP TRAPDEST #SUNTRAP
-> ALTER TRAPDEST $ZSNMP.#SUNTRAP, NETWORK $ZTC1
-> START TRAPDEST #SUNTRAP
```

DELETE Command

The DELETE command:

- Removes an object definition from a NonStop agent configuration
- Can be issued only against an object in the STOPPED or DEFINED state
- Is not valid for the PROCESS object
- Is a sensitive command

DELETE ENDPOINT Command

Use the DELETE ENDPOINT command to remove a request/response connection definition from a NonStop agent configuration.

```
DELETE ENDPOINT [$agent-process.]#endpoint-name
```

`$`*`agent-process`*`.#`*`endpoint-name`*

identifies the request/response connection being removed from the NonStop agent configuration.

`$`*`agent-process`*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`endpoint-name`*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Example: DELETE ENDPOINT

The following example stops and removes the request/response connection defined by the ENDPOINT object #END1 being used by NonStop agent process $ZSNMP:

```
-> ASSUME PROCESS $ZSNMP
-> DELETE ENDPOINT #END1
```

DELETE PROFILE Command

Use the DELETE PROFILE command to remove an entry from the authentication table used by a NonStop agent.

```
DELETE PROFILE [$agent-process.]#profile-name
```

`[$`*`agent-process`*`.]#`*`profile-name`*

identifies the entry being removed from the NonStop agent authentication table.

$agent-process

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#profile-name

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Example: DELETE PROFILE

The following example stops and deletes the #UBLAN entry in the authentication table being used by NonStop agent process $ZSNMP:

```
-> STOP PROFILE $ZSNMP.#UBLAN
-> DELETE PROFILE $ZSNMP.#UBLAN
```

DELETE TRAPDEST Command

Use the DELETE TRAPDEST command to remove a trap destination from a NonStop agent configuration.

```
DELETE TRAPDEST [$agent-process.]#trapdest-name
```

[$*agent-process*.]#*trapdest-name*

identifies the trap destination being removed from the NonStop agent configuration.

$agent-process

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#trapdest-name

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Example: DELETE TRAPDEST

The following example removes all the trap destinations defined for NonStop agent process $ZSNMP:

```
-> DELETE TRAPDEST $ZSNMP.*
```

INFO Command

The INFO command:

- Displays the current attribute values of an object
- Indicates with an asterisk attributes whose values can be changed with the ALTER command
- Is a nonsensitive command

INFO ENDPOINT Command

Use the INFO ENDPOINT command to display the current attribute values for a request/response connection definition.

```
INFO ENDPOINT [$agent-process.]#endpoint-name [, DETAIL ]
```

[$*agent-process*.]#*endpoint-name*

identifies the request/response connection whose attribute values you want to display.

$*agent-process*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#*endpoint-name*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

DETAIL

specifies that the display contain additional information about the object set.

Example: INFO ENDPOINT

The following example displays the attribute values for all connection points defined for NonStop agent process $ZSNMP:

```
-> INFO ENDPOINT $ZSNMP.*

SNMP Info ENDPOINT

Name            *Network        *Hostaddr
$ZSNMP.#DEFAULT \EAST.$ZTC0      0.0.0.0
$ZSNMP.#REMOTE  \WEST.$ZTC0      0.0.0.0
```

INFO PROCESS Command

Use the INFO PROCESS command to display the current attribute values for a NonStop agent process and its defined objects.

```
INFO PROCESS $agent-process [, SUB [ NONE | ALL | ONLY ] ]
   [, DETAIL ] }
```

`$agent-process`

identifies the NonStop agent process whose configuration you want to display. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`SUB [ NONE | ALL | ONLY ]`

selects the NonStop agent objects for which you want to display attribute values:

NONE	returns the current attribute values for the NonStop agent process only. This is the default.
ALL	returns the current attribute values for the NonStop agent process and all of its subordinate objects.
ONLY	returns the current attribute values for the objects subordinate to the NonStop agent process, but not the NonStop agent process itself.

`DETAIL`

specifies that the display contain additional information about the object set.

Examples: INFO PROCESS

- The following example displays detailed information about NonStop agent process $ZSNMP:

```
-> INFO PROCESS $ZSNMP, DETAIL

SNMP Detailed Info PROCESS \EAST.$ZSNMP

 Program file name...... \EAST.$DEV.SNMP.SNMPAGT
 Swap volume............ $QA
*EMS Collector.......... \EAST.$0
```

- The next example displays information about all the objects defined for NonStop agent process $ZSNMP:

```
-> INFO PROCESS $ZSNMP, SUB ALL

SNMP Info PROCESS

Name           *EMS Collector
$ZSNMP          \EAST.$0
```

```
SNMP Info PROFILE

Name            *Access          *Hostaddr
*Community
$ZSNMP.#DEFAULT READONLY         0.0.0.0             public
$ZSNMP.#LANMGR  READWRITE        130.252.86.10       Private

SNMP Info ENDPOINT

Name            *Network         *Hostaddr
$ZSNMP.#DEFAULT \EAST.$ZTC0       0.0.0.0
$ZSNMP.#REMOTE  \WEST.$ZTC0       0.0.0.0

SNMP Info TRAPDEST

Name            *Network         *Hostaddr
*Community
$ZSNMP.#TRAP    \EAST.$ZTC0      130.252.86.10       Tandem
$ZSNMP.#REMOTE  \WEST.$ZTC1      130.252.86.241      public
```

INFO PROFILE Command

Use the INFO PROFILE command to display the current attribute values for an authentication table entry.

```
INFO PROFILE [$agent-process.]#profile-name [, DETAIL ]
```

`[$`*`agent-process`*`.]#`*`profile-name`*

identifies the authentication table entry whose attribute values you want to display.

`$`*`agent-process`*

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`profile-name`*

identifies the PROFILE It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`DETAIL`

specifies that the display contain additional information about the object set.

Considerations: INFO PROFILE

The value in the community name field is truncated if it is too long to fit on a single line. To display the full community name, use the DETAIL option.

Example: INFO PROFILE

The following example displays the attribute values of all the entries in the authentication table used by NonStop agent process $ZSNMP:

```
-> INFO PROFILE $ZSNMP.*

SNMP Info PROFILE

Name             *Access         *Hostaddr           *Community
$ZSNMP.#DEFAULT READONLY         0.0.0.0             public
$ZSNMP.#LANMGR  READWRITE        130.252.86.10       Private
```

INFO TRAPDEST Command

Use the INFO TRAPDEST command to display the current attribute values for a trap destination definition.

```
INFO TRAPDEST [$agent-process.]#trapdest-name [, DETAIL ]
```

`[$`*`agent-process`*`.]#`*`trapdest-name`*

identifies the trap destination definition whose attribute values you want to display.

`$`*`agent-process`*

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`trapdest-name`*

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

`DETAIL`

specifies that the display contain additional information about the object set.

Example: INFO TRAPDEST

The following example displays information about all the trap destinations defined for NonStop agent process $ZSNMP:

```
-> INFO TRAPDEST $ZSNMP.*

SNMP Info TRAPDEST

Name             *Network        *Hostaddr           *Community
$ZSNMP.#TRAP     \EAST.$ZTC0     130.252.86.10       Tandem
$ZSNMP.#REMOTE   \WEST.$ZTC1     130.252.86.241      public
```

NAMES Command

The NAMES command:

- Lists the names of the objects in a NonStop agent configuration
- Is a nonsensitive command

```
NAMES [ PROCESS ] $agent-process [, SUB [ NONE | ALL | ONLY ]
]
```

$agent-process

identifies the NonStop agent process whose defined objects you want to list. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`SUB [ NONE | ALL | ONLY ]`

selects the objects whose names are to be displayed:

NONE	returns the name of the NonStop agent process only.
ALL	returns the names of the NonStop agent process and all of its subordinate objects. This is the default.
ONLY	returns the names of the objects subordinate to the NonStop agent process, but not the NonStop agent process itself.

Example: NAMES

The following example lists the names of all objects defined for NonStop agent process $ZSNMP:

```
-> NAMES $ZSNMP

SNMP Names PROCESS \EAST.$ZSNMP

PROCESS
$ZSNMP

PROFILE
$ZSNMP.#DEFAULT  $ZSNMP.#UBLAN

ENDPOINT
$ZSNMP.#DEFAULT

TRAPDEST
$ZSNMP.#TRAP
```

START Command

The START command:

- Activates an object in a NonStop agent configuration
- Is not valid for the PROCESS object
- Is a sensitive command

START ENDPOINT Command

Use the START ENDPOINT command to activate a request/response connection.

```
START ENDPOINT [$agent-process.]#endpoint-name
```

[$*agent-process*.]#*endpoint-name*

identifies the request/response connection to activate.

$*agent-process*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#*endpoint-name*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: START ENDPOINT

When you START an ENDPOINT object, if the TCP/IP subnet represented by the object definition is not available, the ENDPOINT object enters the STARTING state, and the NonStop agent periodically tries to access the resources. When the resource becomes available, the ENDPOINT object enters the STARTED state.

Example: START ENDPOINT

The following example opens communication between NonStop agent process $ZSNMP and managers through the request/response connection defined by the #SOCKET1 object:

```
-> START ENDPOINT $ZSNMP.#SOCKET1
```

START PROFILE Command

Use the START PROFILE command to activate a defined authentication table entry.

```
START PROFILE [$agent-process.]#profile-name
```

`[$`*`agent-process`*`.]#`*`endpoint-name`*

identifies the authentication table entry to activate.

`$`*`agent-process`*

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`profile-name`*

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Example: START PROFILE

The following example activates all entries in the authentication table used by NonStop agent process $ZSNMP:

```
-> START PROFILE $ZSNMP.*
```

START TRAPDEST Command

Use the START TRAPDEST command to activate the sending of traps to a defined trap destination.

```
START TRAPDEST [$agent-process.]#trapdest-name
```

`[$`*`agent-process`*`.]#`*`trapdest-name`*

identifies the trap destination definition to activate.

`$`*`agent-process`*

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#`*`trapdest-name`*

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: START TRAPDEST

When you START a TRAPDEST object, if an underlying resource is not available (such as the TCP/IP process or one of its subnets), the TRAPDEST object enters the STARTING state.

Every time the NonStop agent generates a trap or receives a trap for forwarding from a subagent, it attempts to reconnect. If the reconnection is successful, the TRAPDEST object enters the STARTED state.

Example: START TRAPDEST

The following example activates the sending of traps to all trap destinations defined for the NonStop agent process $ZSNMP:

```
-> START TRAPDEST $ZSNMP.*
```

STATUS Command

The STATUS command:

- Displays the operational states of objects in a NonStop agent configuration
- Is a nonsensitive command

STATUS ENDPOINT Command

Use the STATUS ENDPOINT command to display the operational state of a request/response connection.

```
STATUS ENDPOINT [$agent-process.]#endpoint-name
```

[$*agent-process*.]#*endpoint-name*

identifies the request/response connection whose operational state you want to display.

$*agent-process*

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#*endpoint-name*

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STATUS ENDPOINT

- A returned status of STARTING indicates that at least one of the following conditions is true:
  - The configured TCP/IP process is not available.
  - The SCP process is not available.
  - The specified subnets are not available.

If the ENDPOINT object is in the STARTING state because an underlying resource is unavailable, the NonStop agent periodically tries to access the resource. Once the resource is available, the ENDPOINT object enters the STARTED state.

- A returned status of STOPPING indicates that a STOP or ABORT command was issued while the NonStop agent was processing an SNMP request. The ENDPOINT object enters the STOPPED state as soon as processing is finished.
- A returned state of STOPPED indicates that the ENDPOINT object has been aborted or stopped.

Example: STATUS ENDPOINT

The following example displays the operational state of all connection points defined for NonStop agent process $ZSNMP:

```
12-> STATUS ENDPOINT $ZSNMP.*

SNMP Status ENDPOINT

Name                State
$ZSNMP.#DEFAULT     STOPPED
$ZSNMP.#END1        STARTED
```

STATUS PROCESS Command

Use the STATUS PROCESS command to display the operational states of the objects defined for a NonStop agent configuration.

The display returned by the STATUS PROCESS command also indicates whether the NonStop agent process is being traced and the name of the trace file.

```
STATUS PROCESS $agent-process [,SUB [ NONE | ALL | ONLY]]
```

`$`*`agent-process`*

identifies the NonStop agent configuration whose operational state you want to display. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`SUB [ NONE | ALL | ONLY ]`

selects the objects whose current operational states you want to display:

NONE	returns the state of the NonStop agent process only. This is the default.
ALL	returns the states of the NonStop agent process and all of its subordinate objects.
ONLY	returns the states of the objects subordinate to the NonStop agent process, but not the NonStop agent process itself.

Considerations: STATUS PROCESS

The NonStop agent process is always reported as being in the STARTED state. If the NonStop agent process has been stopped, the following message returns when you issue a STATUS PROCESS command: Expecting an existing SCF supported object name

Example: STATUS PROCESS

The following example displays the operational states for all objects defined for NonStop agent process $ZSNMP:

```
12-> STATUS PROCESS $ZSNMP, SUB ALL

SNMP Status PROCESS $ZSNMP

Name                Status      Trace   Trace file
$ZSNMP              STARTED     OFF     N/A

SNMP Status PROFILE

Name                State
$ZSNMP.#DEFAULT     STARTED
SNMP Status ENDPOINT

Name                State
$ZSNMP.#DEFAULT     STOPPED
$ZSNMP.#END1        STARTED

SNMP Status TRAPDEST

Name                State
$ZSNMP.#REMOTE      DEFINED
$ZSNMP.#TRAP0       STARTED
$ZSNMP.#TRAP1       STOPPED
```

STATUS PROFILE Command

Use the STATUS PROFILE command to display the operational state of an authentication table entry.

```
STATUS PROFILE [$agent-process.]#profile-name
```

`[$`*`agent-process`*`.]#`*`profile-name`*

identifies the authentication table entry whose operational state you want to display.

$agent-process

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#profile-name

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STATUS PROFILE

- A returned state of DEFINED or STOPPED means the authentication scheme is not currently using the entry.
- A returned state of STARTED means the entry is part of the current authentication scheme.

Example: STATUS PROFILE

The following example displays the operational state of the authentication table entries defined for NonStop agent process $ZSNMP:

```
12-> STATUS PROFILE $ZSNMP.*

SNMP Status PROFILE

Name                State
$ZSNMP.#DEFAULT     STARTED
```

STATUS TRAPDEST Command

Use the STATUS TRAPDEST command to display the operational state of a trap destination definition.

```
STATUS PROFILE [$agent-process.]#trapdest-name
```

[$*agent-process*.]#*trapdest-name*

identifies the trap destination definition whose operational state you want to display.

$agent-process

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#trapdest-name

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STATUS TRAPDEST

- The returned state only gives you information about the ability of the NonStop agent to send trap messages to the associated manager; it does not indicate whether the manager is available to receive those trap messages.
- A returned status of STOPPED indicates that the TRAPDEST object definition has been aborted or stopped.
- A returned status of STARTING indicates that when you tried to start the TRAPDEST object, one of the following conditions was true, or while the TRAPDEST object was in the STARTED state, one of the following conditions occurred:
    - The configured TCP/IP process is not available.
    - None of the subnets associated with the configured TCP/IP process are available.
    - The SCP process is not available.

  Every time the NonStop agent generates a trap or receives a trap to be forwarded from a subagent, it attempts to reconnect. If the reconnection is successful, the TRAPDEST object enters the STARTED state.

Example: STATUS TRAPDEST

The following example displays the operational state of the trap destination definitions for NonStop agent process $ZSNMP:

```
12-> STATUS TRAPDEST $ZSNMP.*

SNMP Status TRAPDEST

Name                State
$ZSNMP.#REMOTE      DEFINED
$ZSNMP.#TRAP0       STARTED
$ZSNMP.#TRAP1       STOPPED
```

STOP Command

The STOP command:

- Inactivates an object definition in a NonStop agent configuration
- Behaves the same way as the ABORT command for NonStop agent objects
- Is not valid for the PROCESS object
- Is a sensitive command

STOP ENDPOINT Command

Use the STOP ENDPOINT command to inactivate a request/response connection between a NonStop agent process and managers.

```
STOP ENDPOINT [$agent-process.]#endpoint-name
```

`[$agent-process.]#endpoint-name`

identifies the request/response connection being inactivated.

$agent-process

identifies the NonStop agent process to which the ENDPOINT object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

#endpoint-name

identifies the ENDPOINT object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STOP ENDPOINT

When you STOP an ENDPOINT object, only you affect the availability of the TCP/IP subnets for handling communication between the NonStop agent and managers. The TCP/IP process and subnets are still available for handling communication between other entities.

If the NonStop agent is processing an SNMP request when a STOP command is issued, the STOP command waits until processing is finished. At that time the ENDPOINT object enters the STOPPED state.

Example: STOP ENDPOINT

The following example closes the communication between NonStop agent process $ZSNMP and managers through the request/response connection defined by the #END1 object:

```
-> STOP ENDPOINT $ZSNMP.#END1
```

STOP PROFILE Command

Use the STOP PROFILE command to inactivate an authentication table entry.

```
STOP PROFILE [$agent-process.]#profile-name
```

`[$agent-process.]#profile-name`

identifies the authentication table entry being inactivated.

`$agent-process`

identifies the NonStop agent process to which the PROFILE object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#profile-name`

identifies the PROFILE object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STOP PROFILE

When you STOP a PROFILE object, the table entry it represents is no longer part of the current authentication scheme. To reactivate the table entry so that it is available for use by the NonStop agent, issue a START command against the object.

Example: STOP PROFILE

The following example inactivates an entry in the authentication table used by NonStop agent process $ZSNMP:

```
-> STOP PROFILE $ZSNMP.#LANMGR
```

STOP TRAPDEST Command

Use the STOP TRAPDEST command to disable the sending of trap messages to a specific trap destination.

```
STOP TRAPDEST [$agent-process.]#trapdest-name
```

`[$agent-process.]#trapdest-name`

identifies the trap destination being inactivated.

`$agent-process`

identifies the NonStop agent process to which the TRAPDEST object belongs. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`#trapdest-name`

identifies the TRAPDEST object. It consists of a pound sign (#) followed by one to seven alphanumeric characters, the first of which must be alphabetic.

Considerations: STOP TRAPDEST

Traps being generated by or received for forwarding by the NonStop agent when a TRAPDEST object is stopped are not sent to that particular destination.

Example: STOP TRAPDEST

The following example disables the sending of trap messages by NonStop agent process $ZSNMP to all its dynamically generated trap destinations:

```
-> STOP TRAPDEST $ZSNMP.DYNA*
```

TRACE Command

The TRACE command:

- Starts a trace operation on a NonStop agent process
- Alters trace parameters set by a previous TRACE command
- Stops a previously requested trace operation
- Is a sensitive command

```
TRACE PROCESS $agent-process { , STOP | { ,trace-option }...
}

trace-option is
    COUNT count
    PAGES pages
    RECSIZE size
    SELECT { METHOD | 3  | ALL | -1}
    TO file-ID
    WRAP
```

`PROCESS $`*`agent-process`*

identifies the NonStop agent process to be traced. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`STOP`

ends the current trace operation.

`COUNT` *`count`*

specifies the number of trace records to be captured. *`count`* is an integer in the range -1 through 32767.

If you omit this option or set it to -1, records are accumulated until you stop the trace operation with the STOP option, or until the end-of-file is reached if the WRAP option is not selected.

`PAGES` *`pages`*

designates how much memory space, in units of 2048-byte pages, is allocated in the extended data segment used for tracing. PAGES can be specified only when a trace is being initiated, not when its parameters are being modified. *`pages`* is an integer in the range 4 through 32767 or is equal to 0.

If you omit this option or set it to 0, the default value of 10,000 is assumed.

`RECSIZE` *`size`*

specifies the length of the data in the trace data records. *`size`* is an integer in the range 16 through 4050 or is equal to 0.

If you omit this option or set it to 0, the default value of 120 bytes is assumed.

`SELECT { METHOD | 3  | ALL | -1 }`

selects the operations to be traced. You can select a trace option by either its keyword or number:

METHOD or 3	Traces all method invocations (function calls).
ALL or -1	Traces all types of records. This is the default.

`TO` *`file-ID`*

specifies the file into which the results of the trace operation are to be placed. This option is required when you are starting a trace operation.

If the file already exists, it is purged of data before the trace is initiated. If the file does not exist, it is created with an extent size based on the value of the PAGES parameter.

If TO is not specified, the existing trace is either stopped (if STOP is specified) or modified as specified in the *`trace-option`*.

`WRAP`

specifies that when the trace disk file end-of-file mark is reached, trace data wraps around to the beginning of the file and overwrites any existing data.

If you omit this option, the default is for wrapping to be turned off.

Considerations: TRACE

- To start a trace operation, issue the TRACE command with the TO option and without the STOP option.
- The standard TRACE command NOCOLL option is automatically selected when you start a trace operation on the NonStop agent process. The NOCOLL option indicates that the trace collector process should not be initiated.

- While being traced, the NonStop agent process continues normal operation. However, during its operation, the NonStop agent process also collects all trace information that meets the selection criteria specified in the TRACE command and passes it to a trace procedure. The trace procedure stores this information in a trace file.
- To alter a previously started trace operation, issue the TRACE command without the STOP or TO options and include a modified COUNT, RECSIZE, or SELECT specification. Trace parameters other than COUNT, RECSIZE, and SELECT cannot be modified during an ongoing trace.
- To stop a trace operation, issue the TRACE command with the STOP option.
- When you initiate a trace operation with the TRACE command, tracing stops when the backup takes over.

  The only way to enable tracing in the backup is to include the TRACE startup parameter when you start the NonStop agent process. On a takeover, the backup begins tracing its reinitialization, storing a complete set of records for all objects. The NonStop agent supports the PTrace utility, described in Section 6, SNMP Agent PTrace Facility.

Examples: TRACE

- The following command starts a trace operation of method invocations only on NonStop agent process $ZSNMP and writes results into the file named $DATA.SNMP.TRACE:

  ```
  -> TRACE PROCESS $ZSNMP, TO $DATA.SNMP.TRACE, SELECT METHOD
  ```
- The following command alters the previously started trace to allow trace data to be overwritten when the end of file (EOF) is reached:

  ```
  -> TRACE PROCESS $ZSNMP, WRAP
  ```
- The next example stops the trace:

  ```
  -> TRACE PROCESS $ZSNMP, STOP
  ```

VERSION Command

The VERSION command:

- Displays the version number, product name, product number, and release date of the NonStop agent product
- Is a nonsensitive command

```
VERSION [ PROCESS ] $agent-process [ , DETAIL ]
```

`$`*`agent-process`*

identifies the NonStop agent process whose version information you want to display. It consists of a dollar sign ($) followed by one to five alphanumeric characters, the first of which must be alphabetic.

`DETAIL`

specifies that complete version information be returned. If it is omitted, only one line of version information is displayed.

Examples: VERSION

The following example shows the display for the VERSION command:

```
-> VERSION PROCESS $ZSNMP
VERSION PROCESS \EAST.$ZSNMP: SNMP Agent (T9576D23 17JAN96)
```

With the DETAIL option, the display looks like this:

```
-> VERSION PROCESS $ZSNMP, DETAIL

Detailed VERSION PROCESS \EAST.$ZSNMP
  SYSTEM \EAST
    SNMP Agent (T9576D23 17JAN96)
    GUARDIAN - T9050 - (N30)
    SCF KERNEL - T9082D30 - (17APR95) (17MAY95)
    SNMP PM - T8603D20-01JUN93-21JUL93
```

Part III. Troubleshooting

Part IV consists of the following sections, which provide diagnostic information:

6 SNMP Agent PTrace Facility

This section contains the following information:

- An introduction to the NonStop agent trace facility and the PTrace utility
- A description of the subsystem-specific PTrace commands and special considerations for using these commands with the NonStop agent
- How the PTrace utility determines the subsystem for which it is formatting records
- How to establish selection criteria for displaying NonStop agent records
- Examples of NonStop agent formatted trace record displays

Introduction to PTrace

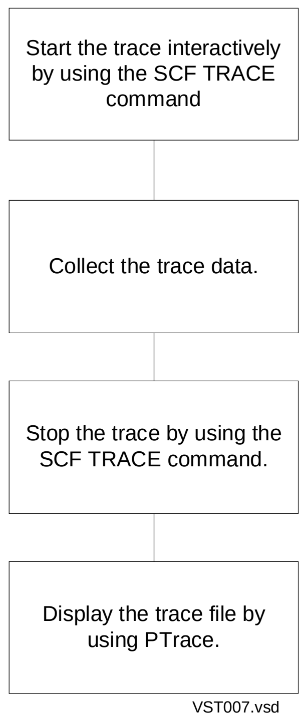
Trace files contain a record of the communication between processes. Each subsystem determines what information is recorded in its trace files. This information varies as to the type of events that are recorded, the amount of detail included, and other subsystem-specific parameters.

You can generate a NonStop agent trace file interactively or programmatically, as follows:

- To start a trace and capture data interactively, use the SCF TRACE command.
- To start a trace and capture data programmatically, use a management programming application that uses the same trace facility as SCF.

The trace files created are unstructured and cannot be printed or displayed directly. Use the PTrace (print trace) utility to display and examine trace files. The PTrace utility formats the data stored in unstructured trace files for output to terminals, printers, or disk files. You can then use this information for troubleshooting. Figure 6-1 shows the four general steps involved in recording and formatting trace data.

Figure 6-1. Recording and Displaying Trace Data


Start the trace interactively by using the SCF TRACE command
Collect the trace data.
Stop the trace by using the SCF TRACE command.
Display the trace file by using PTrace.
VST007.vsd

The following steps are involved in recording and displaying trace data.

1. To start the trace interactively, issue the SCF TRACE command. For example, to start a trace operation of method invocations only on NonStop agent process $ZSNMP and write results into the file named $DATA.SNMP.TRACE, you would enter the following SCF TRACE command:

   ```
   -> TRACE PROCESS $ZSNMP, TO $DATA.SNMP.TRACE, SELECT METHOD
   ```

2. Collect the trace data while you perform operations related to the problem that you are analyzing.

3. Stop the trace by issuing another SCF TRACE command. The following command stops the trace for the NonStop agent process $ZSNMP:

   ```
   -> TRACE PROCESS $ZSNMP, STOP
   ```

4. Display the trace file with the PTrace utility.

   Run PTrace from TACL to display the traces in the file named TRACE:

   ```
   PTRACE FROM $DATA.SNMP.TRACE
   ```

For additional information on using PTrace, refer to the *PTrace Reference Manual*.

Running PTrace

Although the *PTrace Reference Manual* covers this subject in more detail, the descriptions that follow should be enough to get you started running PTrace.

Starting a Noninteractive PTrace Session

The syntax of the TACL or SCF RUN command for PTrace is:

```
PTRACE [ /run-option-list ] [ ptrace-command ] ...
```

run-option-list

is one or more NonStop Kernel run options, as described in the *TACL Reference Manual*. Although other run options are allowed, the following options have special significance for PTrace:

IN *file-name*

designates the initial source for PTrace command input for a session. If you omit the IN option, the input source defaults to the input file for the TACL process through which you started PTrace (usually a terminal).

NOWAIT

indicates that TACL or SCF does not wait while PTrace runs but returns a command interpreter prompt after sending the startup message to PTrace. If you omit this option, TACL or SCF pauses while PTrace runs.

OUT [*file-name*]

designates the initial destination for PTrace display output for the session. You can use this formatted file for your problem diagnosis. If you omit the OUT option, the output destination defaults to the output file of the TACL or SCF process through which you started PTrace (usually a home terminal).

If you specify the OUT option and omit *file-name*, PTrace output is discarded. If you specify an existing disk file, the file's data is purged, and new data is written to the file. If the file does not exist, TACL creates an EDIT file with an extent size of 4 pages for the primary extent and 16 pages for secondary extents (maximum of 400 pages). You can have up to 16 secondary extents for a maximum file size of 260 pages. If you will need a file larger than 260 pages, you must use a previously created file. When the PTrace disk file reaches end of file (EOF), PTrace stops.

ptrace-command

is a PTrace command that is to be executed immediately after PTrace is initiated (such as the SELECT command, described on 6-10). If you specify one or more PTrace commands in the RUN command, PTrace executes the commands and then issues its prompt. If PTrace encounters an error in a command, it terminates immediately.

In the following example, the TACL RUN command instructs PTrace to read its commands from a file called PTRACE.FIRSTRUN and to write its display output to a file called PTRACE.FIRSTOUT. The NOWAIT option allows TACL to resume control of the session while PTrace is executing.

```
PTRACE/IN PTRACE.FIRSTRUN, OUT PTRACE.FIRSTOUT, NOWAIT/
```

Starting an Interactive PTrace Session

To start PTrace interactively, enter PTRACE at the TACL or SCF prompt. PTrace displays its product banner, followed by its prompt, the question mark (?). When you are finished using PTrace, enter the EXIT command to return to TACL or SCF.

Device Type

When a trace file is created, the type of the device being traced is recorded in that file. When PTrace opens the trace file, it uses this information to determine the subsystem for which it is formatting records.

The device type for the NonStop agent process is 31.

PTrace Commands

The PTrace commands provide options for selecting trace records for display. You can suppress these records that do not relate to the problem you are investigating. The PTrace commands also provide options for specifying the way in which the trace records are formatted.

Although PTrace provides a common set of commands for displaying trace records, not all PTrace commands are supported by each subsystem due to the structure of the PTrace code.

PTrace Commands Supported by the NonStop Agent

The PTrace code consists of two modules. The first module contains the code shared by all subsystems. The second contains the additional, subsystem-specific code that actually displays the PTrace records. The following commands implemented by the first PTrace module are supported by all subsystems:

ALLOW	FC	LIMIT	OUT
COUNT	FIND	LOG	PAGESIZE
ENV	FROM	NEXT	RECORD
EXIT	HELP	OBEY	RESET

Additional commands implemented by the subsystem-specific PTrace modules vary from subsystem to subsystem. Of the commands that fall into the subsystem-dependent category, the NonStop agent supports the following:

HEX	DETAIL
LABEL	SELECT
OCTAL	TEXT

For descriptions of these standard PTrace commands available to all subsystems, refer to the *PTrace Reference Manual*. Table 6-1 describes the SELECT command.

PTrace Commands Not Supported by the NonStop Agent

The following subsystem-dependent PTrace commands are not supported by the NonStop agent:

EBCDIC	TEST
FILTER	TRANSLATE
SETTRANSLATE	

Subset of Useful NonStop Agent PTrace Commands

Table 6-1 lists and describes a subset of PTrace commands that are particularly useful when formatting NonStop agent trace records using the PTrace facility. The SELECT command establishes the criteria against which records are compared. The NEXT and RECORD commands determine the range of records in the currently opened trace file that will be examined.

Table 6-1. Summary of NonStop SNMP Agent PTrace Commands (page 1 of 2)

Command	Description
FROM	Opens the specified trace file

Table 6-1. Summary of NonStop SNMP Agent PTrace Commands (page 2 of 2)

Command	Description
NEXT	Specifies the number of records or specifies a time after which records are to be displayed or printed
RECORD	Displays or prints record(s) within the specified range or prints all records
SELECT	Establishes selection criteria for displaying records

The remainder of this subsection describes these commands in detail with a brief description of each command, the command syntax, and any special considerations applicable to the command.

For a description of the notation scheme used here, see Notation Conventions on page -xxviii.

FROM Command

The FROM command causes the PTrace kernel to open the trace file and read the device type and subtype information from the header. If a trace file is currently in use, PTrace closes it before opening the new one.

```
FROM file-name
```

file-name

is the name of a trace file.

The syntax of *file-name* is as follows:

```
[\system-name.] [$volume-name.] [subvolume-name.] file-id
```

where:

- \system-name identifies a specific system within a network. For example, \NOVA.
- $*volume-name* identifies a physical disk pack mounted on a disk unit. For example, $DATA5.
- *subvolume-name* identifies a related set of files defined by the user. For example, SNMT101.
- *file-id* identifies a particular file within the subvolume. For example, ZZSMPTRP.

For convenience, default values are provided for all parts except *file-id*. The initial default values are the values provided by the TACL or SCF process through which you invoked PTrace. For example, if you invoke PTrace while your current system, volume, and subvolume are \NOVA.$DATA5.SNMT101, then your default system, volume, and subvolume are \NOVA.$DATA5.SNMT101.

Considerations: PTrace FROM Command

- The FROM command resets current session parameters to their default values. For more information, see the RESET Command in the *PTrace Reference Manual*.
- The FROM command resets current SELECT options to their default values if the device type and subtype of the new trace file are different from the last trace file.

  If the device type and subtype of the new trace file are the same as the last trace file, the SELECT option is not reset.
- The FROM command automatically displays the trace file header record (record 0) when it opens the new trace file.
- The FROM command does not close a log file if one is specified.

Example: PTrace FROM Command

The following example closes the previous trace file (if any), opens the trace file called ZZSMPTRP in the subvolume SNMT101 in the current system and volume, resets session parameters to their defaults, and displays the trace file header record:

```
?FROM snmt101.zzsmptrp
Trace of:                \NOVA.$ZSNMP, Type (31,30).
Ptrace Data File:        $DATA5.SNMT101.ZZSMPTRP
Trace started:           11/19/97  11:27:50.33
First trace entry:       11/19/97  11:27:50.63
Last trace entry:        11/19/97  11:27:53.76
Trace file is in extended format.
Trace entry size limit:  0
Select options are reset.
```

NEXT Command

The NEXT command displays records starting from the next record in the trace file. The number of records displayed by a single NEXT command never exceeds the number set by the LIMIT command. Records that do not meet the current selection criteria are not counted and are not displayed.

```
N[EXT] [ count ] [ AFTER [date] time ]
```

count

specifies the number of records to be displayed. If *count* is omitted, one record is displayed.

AFTER [*date*]*time*

indicates that PTrace is to move to the first record whose timestamp is greater than or equal to the [*date*] *time* specified before selecting the next records for display.

`date`

is an optional parameter in the following form:

`mm/dd/yyyy`

Note. The year field *must* be four digits.

If *`date`* is omitted, it is replaced by the date found in the previously displayed record. If there is no previously displayed record, the date is taken from the trace file header (start of trace).

`time`

represents a 24-hour-clock time in the following form:

`hh:mm[:ss[.tttttt]]`

You must specify the hours (*`hh`*) and minutes (*`mm`*). You can optionally specify the seconds (*`ss`*) and microseconds (*`tttttt`*).

Once a record is found that has a timestamp greater than or equal to the timestamp specified, the *`count`* parameter (or function key that is pressed, as discussed in the following subsection) determines the number of records sent to the output.

Considerations: PTrace NEXT Command

- Pressing the BREAK key terminates the NEXT command search or display.
- The function keys on an HP 6530 terminal can be used as shorthand for various NEXT *`count`* commands. For example, pressing the F6 function key displays six records.

Examples: PTrace NEXT Command

1. The following example sets detail to on to display the information and then displays the next record (record 1):

```
?DETAIL ON
Ptrace Detail Format:  ON
?NEXT
11:27:50.630 >000.000 #1             Common kernel
SSN : 155
Product banner : SNMP Agent (T9576D23 15MAR97 AAG)
Kernel banner  : COM-KRNL   (T6563D20 01SEP96 T07L^AAS)
```

Record 1 is written by Common Kernel into the trace file. This is the only record that differs from rest of the records written by the NonStop agent.

2. This example displays the next two records whose timestamp is greater than the timestamp specified in the command.

```
?NEXT 2 AFTER 11/19/1997 11:27:52
11:27:52.230 >001.060 #680            Method
Function name:      SnmpAgent_Thread
Thread name:     SnmpAgent    ID:1
Method class:     SnmpAgent
Method name: Thread
Object Count:0
Paramstring:

11:27:52.240 >000.010 #681            Method
Function name:       sysGroup_Load
Thread name:     SnmpAgent    ID:1
Method class:      sysGroup
Method name: Load
Object Count:0
Paramstring:
```

RECORD Command

The RECORD command displays records selected by record number. You can display records individually, in a range, or ALL. The RECORD command never displays more than the number of records specified by the LIMIT command.

```
RECORD [ first ] | first,last | ALL
```

first

is an integer that specifies the record number of the first or only record to be displayed.

last

is an integer that specifies the record number of the last record in a range of records to be displayed.

ALL

specifies that all records in the trace file are to be displayed.

Considerations

- The first record in the trace file, the trace file header record, is record number 0. The first data record is record number 1.
- A slash (/) can be used in place of the comma (,) in the RECORD command syntax.
- Pressing the BREAK key terminates the display of records.

Examples: PTrace RECORD Command

1. The following example displays record 18:

```
?RECORD 18
11:27:51.290 >000.000 #18             Method
Function name:     dictionary_KeySearch
Thread name:           NIL    ID:0
Method class:    dictionary
Method name: KeySearch
Object Count:2
Object:    dictionary       Object ADDR:612610D
Object:       UNKNOWN       Object ADDR:612562D
```

2. The next example sets the detail to off to display header information only and then displays all the records in the range 0 through 4:

```
?DETAIL off
Ptrace Detail Format:  OFF
?record 0/4
Trace of:                 \NOVA.$ZSNMP, Type (31,30).
Ptrace Data File:         $DATA5.SNMT101.ZZSMPTRP
Trace started:            11/19/97  11:27:50.33
First trace entry:        11/19/97  11:27:50.63
Last trace entry:         11/19/97  11:27:53.76
Trace file is in extended format.
Trace entry size limit:  0
11:27:50.630 >000.000 #1             Common kernel
11:27:50.760 >000.130 #2             Method
11:27:50.760 >000.000 #3             Method
11:27:50.760 >000.000 #4             Method
```

SELECT Command

The SELECT command establishes the selection criteria that control which trace records are to be displayed. When PTrace is determining which records to display in response to a NEXT, FIND, or RECORD command, it checks its selection mask to determine whether the record is of a type that you want to display.

You can select specific records through a keyword that is translated into an enumerated value or through a number that is saved as an enumerated value.

```
SELECT [ ALL                          ]
       [ CURRENT                      ]
       [ number                       ]
       [ keyword                      ]
       [ ( keyword [ , keyword ]... ) ]
```

ALL

selects all trace-selection options.

`CURRENT`

selects the currently defined trace-selection options.

`number`

is an integer that specifies an enumerated value corresponding to a specific keyword.

`number` can be specified in decimal, octal, hexadecimal, or binary notation. When you specify *`number`*, it is saved as an enumerated value. The formats and ranges follow:

```
      decimal:    0 - 32767
        octal:   %0 - %77777
  hexadecimal:  %H0 - %H7FFF
       binary:  %B0 - %B111111111111111
```

`keyword`

is one of the keywords listed in Table 6-2.

Table 6-2. Keywords for NonStop SNMP Agent PTrace SELECT Command

Keyword	Value	Meaning
METHOD	1	Enables the tracing of the trace records written by the NonStop agent only
COMMON	254	Enables the tracing of the trace records written by the Common Kernel only

Consideration: PTrace SELECT Command

If the SELECT command is not entered, or if the SELECT command is specified with no keywords, ALL is assumed.

Examples: PTrace SELECT Command

1. The following command sequence positions the record pointer at record 1 and changes the detail to off to display header information only:

```
?RECORD 0
Trace of:                \NOVA.$ZSNMP, Type (31,30).
Ptrace Data File:        $DATA5.SNMT101.ZZSMPTRP
Trace started:           11/19/97  11:27:50.33
First trace entry:       11/19/97  11:27:50.63
Last trace entry:        11/19/97  11:27:53.76
Trace file is in extended format.
Trace entry size limit:  0

?DETAIL off
Ptrace Detail Format:  OFF
```

2. The next example selects the trace records written by the NonStop agent and then displays the next two records:

```
?SELECT METHOD
 Ptrace Select Key:     1

?NEXT 2
11:27:50.760 >000.000 #2              Method
11:27:50.760 >000.000 #3              Method
```

3. The next example selects the trace records written by the Common Kernel and displays the next two records:

```
?SELECT COMMON
Ptrace Select Key:     254

?NEXT 2
End of Trace file.
```

4. This example positions the record pointer at record 1 and displays the next three records:

```
?RECORD 0
Trace of:                 \NOVA.$ZSNMP, Type (31,30).
Ptrace Data File:         $DATA5.SNMT101.ZZSMPTRP
Trace started:            11/19/97  11:27:50.33
First trace entry:        11/19/97  11:27:50.63
Last trace entry:         11/19/97  11:27:53.76
Trace file is in extended format.
Trace entry size limit:  0

?SELECT COMMON
Ptrace Select Key:     254

?NEXT 3
11:27:50.630 >000.000 #1              Common kernel
End of Trace file.
```

7 Troubleshooting the SNMP Agent

This section contains general guidelines about what to do when you encounter problems starting the SNMP agent. In this section, a general strategy for troubleshooting appears first, followed by guidelines for:

- Diagnosing startup errors
- Identifying unavailable resources
- Using EMS filters
- Using trace records
- Diagnosing request errors

Troubleshooting Strategy

Because of the interaction between the agent process, the subagents with which it communicates, the HP TCP/IP subsystem, and the underlying LAN access and X25AM subsystems, it might be difficult to determine whether problems arose within the SNMP agent or one of the resources it uses.

When you are troubleshooting a problem, the best approach is to begin with the assumption that the underlying subsystems are operating normally. Concentrate first on the interactions of the subsystem in which the problem became apparent. The guidelines provided in this section should help you.

If you suspect the trouble is in one of the underlying network resources being used by the SNMP agent, start with the *TCP/IP Configuration and Management Manual* for troubleshooting guidelines for the TCP/IP subsystem.

For systems running on a G-series or H-series RVU of the NonStop Kernel, if you suspect the trouble is in the ServerNet LAN Services Access (SLSA) subsystem, refer to the *LAN Configuration and Management Manual*.

If your SNMP agent is communicating through an X.25 subnet, refer to the *SCF Reference Manual for X25AM* for information on examining the configuration and status of the X25AM subsystem, initiating traces, and examining trace information.

Diagnosing Startup Errors

Most of the errors encountered at startup are configuration errors for either the SNMP agent or the underlying resources it needs.

If you reset any of the default configuration values for the SNMP agent, review your modifications to make sure they are correct. If possible, rename SNMPCTL and run the SNMP agent with the WARM startup parameter, using the built-in default values to isolate the problem.

If any of the underlying resources used by the SNMP agent are not running, an SCF error and corresponding event messages are generated.

Identifying Unavailable Resources

You can display the current operational states of the SNMP agent resources by either:

- Issuing an SCF STATUS command against SNMP agent PROFILE, ENDPOINT, and TRAPDEST objects
- Issuing Get requests from an SNMP manager against SNMP agent private zagInEpState and zagInTdState MIB objects

If an object is not in an expected state, look at the event log for information about the availability of underlying resources. (See the *EMS Manual* for information on viewing the contents of the event log.)

Underlying Request/Response Connection Resources

If a STATUS command indicates that an ENDPOINT object is in the STOPPED state, you can issue the START command to try to start it. Likewise, if an SNMP Get request indicates that a zagInEpState object is stopped(5), you can set the zagInEpStatus object to active(1) through SNMP.

If the object remains in the STARTING state, at least one of the following conditions is true:

- The configured TCP/IP process is not available.
- The SCP process is not available.
- The specified subnet is not available.

The SNMP agent periodically tries to access the resource. As soon as the resource is available, the ENDPOINT object enters the STARTED state.

Underlying Trap Destination Resources

If a STATUS command indicates that a TRAPDEST object is in the STARTING state, or if an SNMP Get request indicates that a zagInTdState object is starting(4), an underlying TCP/IP process might be unavailable.

The SNMP agent tries every 30 seconds to open a connection to the TCP/IP process in question. If the reconnection is successful, the SNMP agent sends a coldStart trap to the trap destination. If the reconnection is unsuccessful, the SNMP agent generates a socket error event message.

Using EMS Filters

Individual event messages or groups of messages can provide information for diagnosing problems. EMS filters are a useful mechanism for selecting individual messages or groups of messages of special interest. For example, you might want to select all the messages sent by the SNMP agent and TCP/IP.

The filtered event stream can be viewed several ways:

- Running a printing distributor. This distributor writes event messages to a printer or other display device or to a file.
- Using the operator console. You can read the event message on the designated operator console controlled by the compatibility distributor ($Z0) or the console used by the printing distributor.
- Using the EMS trap subagents. These subagents, documented in Section 9, EMS Trap Subagent, let you view EMS events as traps on an SNMP manager station.

This subsection provides an example showing how to create and use a filter that selects messages generated by the SNMP agent and TCP/IP subsystem. Section 3, MIBs Supported by the SNMP Agent, describes event messages generated by the SNMP agent. For more information about EMS and event messages, refer to the *EMS Manual* and the *Operator Messages Manua*.

Creating a Filter

The following filter selects all event messages reported by the SNMP agent and TCP/IP on system \MYSYS:

```
== FILE:  SNMPS

FILTER pass^SNMP^TCPIP^MSGS;

BEGIN
IF ZEMS^TKN^SYSTEM = [ #SYSTEMNUMBER \MYSYS] AND
  ((ZSPI^TKN^SSID = SSID(ZSMP^VAL^SSID))  OR
   (ZSPI^TKN^SSID = SSID(ZTCI^VAL^SSID))  OR
   (ZSPI^TKN^SSID = SSID(ZCMK^VAL^SSID))  THEN
      PASS;
END;
```

Compiling the Filter

Before using a filter, you must compile it with the EMS filter compiler, EMF. Because filters are based on the HP Tandem Advanced Command Language (TACL), you load the TACL versions of the data definition language (DDL) files containing the definitions used in your filter. The following TACL routine defines a TACL macro named USESPI, which makes the SPI SEGF files usable to TACL. The routine then calls USESPI to set up the files needed to EMF-compile the filter in the file named SNMPS. Finally, the routine calls EMF to compile the filter:

```
?TACL ROUTINE
== FILE:  EXSNMPS
[#DEF USESPI MACRO |BODY|
  #DEF %1% DIRECTORY SHARED $SYSTEM.ZSPISEGF.%1%
  #SET #USELIST [#USELIST] %1%
]
USESPI ZSPISEGF
USESPI ZCOMSEGF
USESPI ZEMSSEGF
USESPI ZSMPSEGF
USESPI ZTCISEGF
USESPI ZCMKSEGF
EMF /IN SNMPS/ SNMPF
```

Displaying Filtered Event Messages

When EMS is running and the log files are secured so that you can access them, it is easy to run a printing distributor. The simplest case is to run a printing distributor to your terminal by entering a TACL command such as the following:

```
EMSDIST /NAME $DIST1/ TYPE PRINTING, COLLECTOR $0, &
    TEXTOUT $TERM1, FILTER $SYSTEM.MYFLTRS.SNMPF
```

This command starts distributor process $DIST1, which sends all messages received after startup to terminal $TERM1 if they pass through filter SNMPF. The source of the event messages is $0, the primary collector on the local system.

$DIST1 continues sending messages until you stop it. Assigning the distributor a name enables you to stop the process using that name. To stop the distributor, press the Break key and then use a TACL STOP $DIST1 command.

Using Trace Records

The SNMP agent and the HP TCP/IP subsystem both provide a facility for generating trace records to help you diagnose problems.

While being traced, subsystems continue normal operation but pass all message traffic to a trace procedure. When the trace procedure recognizes a message that meets its selection criteria, it stores the message in a trace file. Trace operations are usually started with the SCF TRACE command.

TCP/IP

For the HP TCP/IP subsystem, you can use the PTrace utility to format, display, and examine the trace information collected by the TRACE command. You can limit the information you format by using the options provided in the PTrace SELECT and FILTER commands. For information on analyzing trace information with the PTrace utility, see the *TCP/IP Configuration and Management Manual* (G-series).

SNMP Agent

You can generate trace records for the SNMP agent in two ways:

- By issuing an SCF TRACE command
- By including the TRACE startup parameter when you start the SNMP agent

The SNMP agent subsystem does not currently support the PTrace utility. Generally, you generate trace records at the request of and for analysis by your HP support representative.

Using SCF

The SCF TRACE command lets you:

- Start a trace operation on the agent process.
- Alter trace parameters set by a previous TRACE command.
- Stop a previously requested trace operation.

The complete syntax of the SCF TRACE command is in the TRACE Command on page 5-35.

The following command starts a trace operation on the agent process, writes results into the file named $DATA1.TRC.TRACE, and allows the trace data to be overwritten when the EOF is reached:

```
TRACE PROCESS $ZSNMP, TO $DATA.SNMP.TRACE, WRAP
```

This command stops the trace:

```
TRACE PROCESS $ZSNMP, STOP
```

The SCF trace facility for the SNMP agent does not support the BACKUP parameter, so tracing stops if the backup process takes over. To enable tracing in the backup, include the TRACE startup parameter when you start the SNMP agent.

Using the RUN Command

The TRACE startup parameter of the RUN command enables tracing of the agent process if a takeover by the backup process occurs. The backup begins tracing its reinitialization and stores a complete set of records for all objects.

If you start the SNMP agent with the TRACE startup parameter, it creates primary and backup trace files—ZZSMPTRP and ZZSMPTRB, respectively—in the subvolume from which the agent process was started. To override the default trace files, you must provide DEFINE statements specifying the locations of the primary and backup trace files. For example:

```
ADD DEFINE =PRIMARY-TRACE-FILE, CLASS MAP, FILE $NETMAN.SNMP.SNMPTRCP
ADD DEFINE =BACKUP-TRACE-FILE, CLASS MAP, FILE $NETMAN.SNMP.SNMPTRCB
```

Using DEFINE statements is described in the *TACL Reference Manual*.

Diagnosing Request Errors

Table 7-1 describes what the SNMP agent does when it or one of the subagents with which it is communicating receives Get, GetNext, or Set requests that do not result in a normal response. The SNMP agent usually increments one of the counters in its SNMP group. In most cases, the SNMP agent also assigns a value to the Error-Status field of a GetResponse PDU or generates a trap.

Table 7-1. Handling Requests That Cannot Be Processed (page 1 of 3)

Error Condition	SNMP Group Counter Incremented	Error-Status Value Assigned	Message Handling
The results of an operation cannot fit into a single SNMP message.	snmpOutTooBigs	tooBig	SNMP agent stops processing the request and returns an error.
The SNMP version number in the request does not match the version supported.	snmpInBadVersions		SNMP agent discards the request.
The incoming request could not be authenticated.	snmpInBadCommunityNames		SNMP agent discards the request and generates an authenticationFailur e trap.
The SNMP agent could not decode the request PDU.	snmpInASNParseErrs		SNMP agent discards the request.

Table 7-1. Handling Requests That Cannot Be Processed (page 2 of 3)

Error Condition	SNMP Group Counter Incremented	Error-Status Value Assigned	Message Handling
The SNMP agent receives a GetRequest PDU for an object that HP does not support.	snmpOutNoSuchNames	noSuchName	Depends on how the subagent is implemented. The TCP/IP Subagent, for example, returns a null or a 0, depending on whether the object is a string or a counter.
The SNMP agent receives a SetRequest PDU from a community assigned READONLY access.	snmpInBadCommunityUses		SNMP agent stops processing the request and generates an authenticationFailur e trap.
The SNMP agent receives a SetRequest PDU for a MIB object assigned read-only access.	snmpOutNoSuchNames	noSuchName	SNMP agent stops processing the request.
The SNMP agent receives a SetRequest for a MIB object that has been assigned read write access but is not in the System or SNMP group.	snmpOutNoSuchNames	noSuchName	SNMP agent stops processing the request.

Table 7-1. Handling Requests That Cannot Be Processed (page 3 of 3)

Error Condition	SNMP Group Counter Incremented	Error-Status Value Assigned	Message Handling
A SetRequest PDU attempts to assign an invalid value to a MIB object.	snmpOutBadValues	badValue	SNMP agent stops processing the request.
Any other condition that does not produce a normal response, such as certain errors returned in the SPI interface.	snmpOutGenErrs	genErr	SNMP agent stops processing the request.
The SNMP agent receives no response from a subagent.		noSuchName	SNMP agent stops processing the request.

Part IV. SNMP Subagents

Part II consists of the following sections, which describe the SNMP subagents:

8 TCP/IP Subagent

The TCP/IP Subagent supports the MIB-II groups that allow TCP/IP resources (including Parallel Library TCP/IP, and NonStop TCP/IPv6) on NonStop systems to be managed from SNMP managers. It also supports additional private MIB objects defined by HP that let you monitor and manage the subagent itself. This section describes the TCP/IP Subagent and the MIBs it supports.

Architectural Overview

The TCP/IP Subagent supports management of HP TCP/IP resources and the subagent in the following ways:

- The TCP/IP Subagent implements five of the MIB-II groups defined in RFC 1213 that describe TCP/IP resources.
- Additional private objects in the TCP/IP Subagent's MIB defined by HP allow the TCP/IP Subagent to be monitored and managed through SNMP.
- The TCP/IP Subagent also includes the set of private MIB objects which provides information about the subnets associated with the TCP/IP subsystems.

The following interaction between the SNMP agent, the TCP/IP Subagent, and the managed TCP/IP resources is illustrated in Figure 8-1, Architectural Overview of the TCP/IP Subagent, on page 8-2:

- The SNMP agent receives and authenticates SNMP request messages regarding HP TCP/IP resources and forwards authenticated requests to the TCP/IP Subagent with which it is communicating.

  Note. The TCP/IP Subagent performs no further authentication of SNMP requests forwarded to it by the SNMP agent.

- The TCP/IP Subagent translates the request, retrieves the requested objects from a cache of MIB-II object values, and returns them in an SNMP response to the SNMP agent.
- Independent of SNMP requests, the TCP/IP Subagent periodically updates the MIB-II object cache using the Subsystem Programmatic Interface (SPI) to exchange messages with the Subsystem Control Point (SCP) process or the LAN Management Library (ZLANSMRL).

  Requests are forwarded to the subsystem processes that manage the HP TCP/IP resources for which the TCP/IP Subagent is responsible. When the TCP/IP Subagent receives a response, it uses the contents to update the values for MIB-II cache objects.

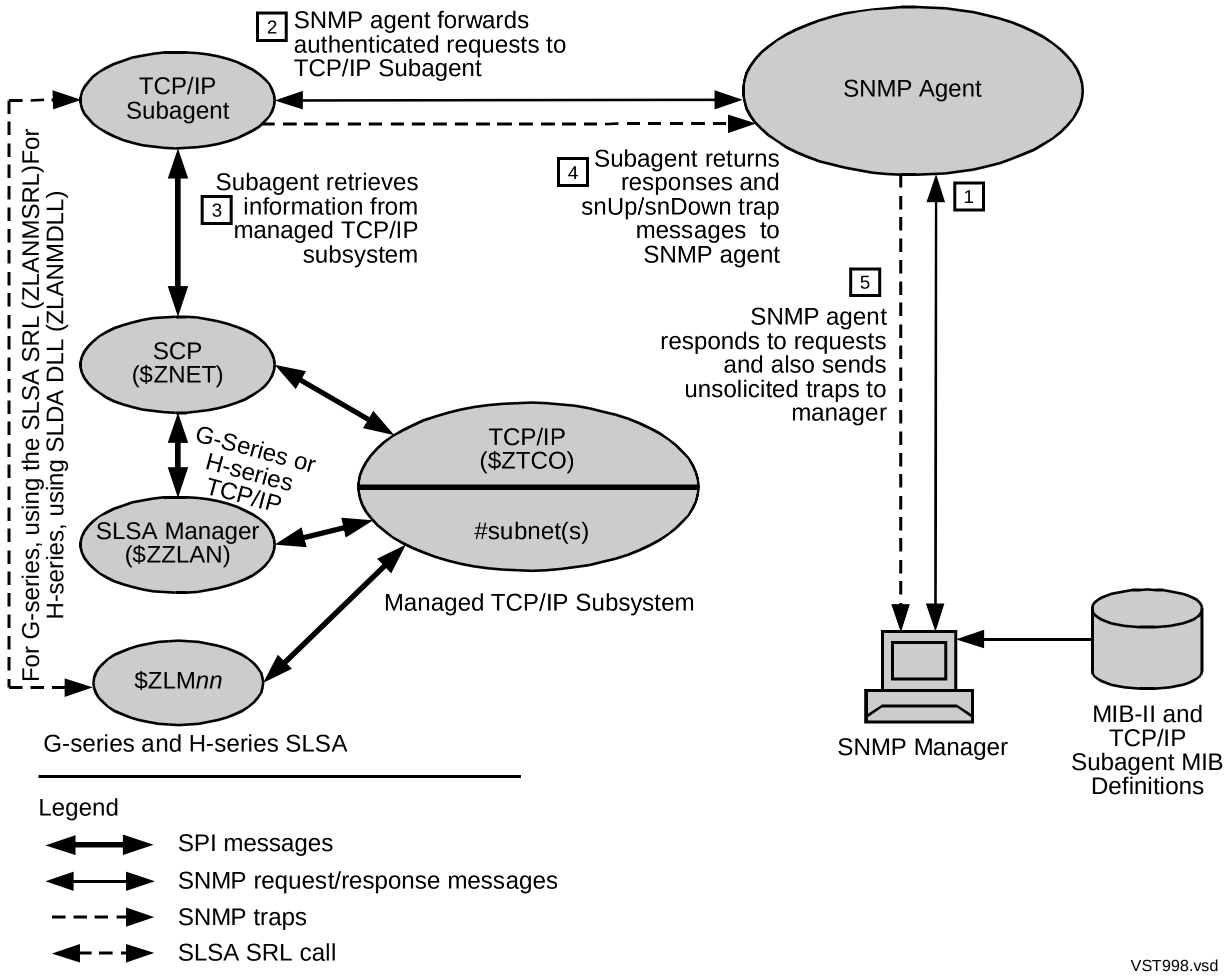
- The SNMP agent sends response messages to the SNMP manager from which the request originated.

- The SNMP agent sends response messages to the SNMP manager from which the request originated.

Most of the information in the MIB-II groups as supported by the TCP/IP Subagent describe the HP TCP/IP subsystem and, if the interface is the ServerNet LAN Systems Access (which supports parallel LAN I/O in a G-series or H-series ServerNet based system), the underlying SLSA subsystem.

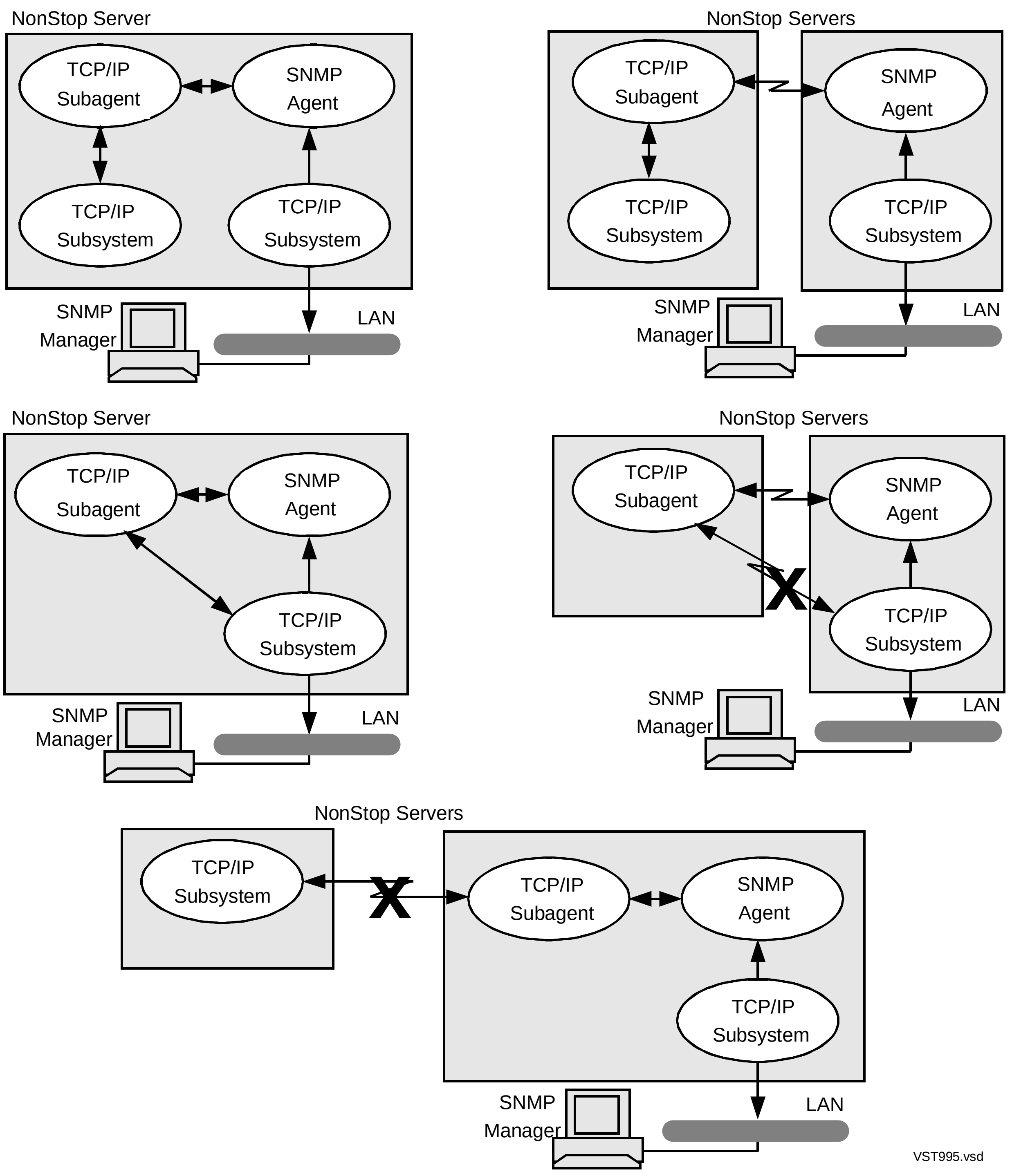
The only information the TCP/IP Subagent retrieves regarding an underlying X.25 Access Method (X25AM) subsystem (which supports wide area network data communications over an X.25 packet-switching data network) is information returned from inquiries to the HP TCP/IP subsystem.

Figure 8-1. Architectural Overview of the TCP/IP Subagent


2 SNMP agent forwards authenticated requests to TCP/IP Subagent
TCP/IP Subagent
SNMP Agent
For G-series, using the SLSA SRL (ZLANMSRL)For H-series, using SLDA DLL (ZLANMDLL)
3 Subagent retrieves information from managed TCP/IP subsystem
4 Subagent returns responses and snUp/snDown trap messages to SNMP agent
1
5 SNMP agent responds to requests and also sends unsolicited traps to manager
SCP ($ZNET)
G-Series or H-series TCP/IP
TCP/IP ($ZTCO)
#subnet(s)
SLSA Manager ($ZZLAN)
Managed TCP/IP Subsystem
$ZLMnn
G-series and H-series SLSA
SNMP Manager
MIB-II and TCP/IP Subagent MIB Definitions
Legend
SPI messages
SNMP request/response messages
SNMP traps
SLSA SRL call
VST998.vsd

TCP/IP Subagent Scenarios on page 8-3 illustrates valid and invalid configuration scenarios to keep in mind when planning for your TCP/IP Subagent configuration. An **X** indicates an invalid configuration.

Figure 8-2. TCP/IP Subagent Scenarios



The TCP/IP Subagent and Its Managed Resources

Each TCP/IP Subagent manages only one HP TCP/IP subsystem. The TCP/IP resources being managed by the TCP/IP Subagent must reside on the same node as the TCP/IP Subagent. The TCP/IP Subagent can monitor the same TCP/IP resources that the SNMP agent is using to communicate with SNMP managers. Each TCP/IP Subagent can monitor subnets associated with the multiple TCP/IP subsystems.

You define the TCP/IP resources to be monitored and the SCP process with which the TCP/IP Subagent communicates when you start the subagent. By default, the TCP/IP Subagent monitors the TCP/IP process $ZTC0 running on the local node and communicates through the SCP process $ZNET. You have to define the TCP/IP resource(s) while starting the TCP/IP Subagent to monitor the subnet information associated with them.

The NonStop TCP/IP Subagent and the SNMP Agent

Any SNMP agent process can communicate with only one TCP/IP Subagent process. Because each TCP/IP Subagent can monitor only one TCP/IP subsystem, you must start one SNMP agent/TCP/IP Subagent pair for each TCP/IP subsystem you want to monitor.

The TCP/IP Subagent does not have to reside on the same node as the SNMP agent with which it is communicating, but the TCP/IP resources being managed by the TCP/IP Subagent must reside on the same node as the TCP/IP Subagent.

You define the SNMP agent with which the TCP/IP Subagent communicates when you start the subagent. By default, the TCP/IP Subagent communicates with the SNMP agent $ZSNMP on the same node as the subagent.

The Parallel Library TCP/IP Subagent and Its Managed Resources

Note. H-series RVUs do not support Parallel Library TCP/IP.

To run SNMP over Parallel Library TCP/IP, you must define the name of a TCPSAM process. You can find this process by issuing a LISTDEV TCPIP command to find a process whose program name ends in TCPSAM. See the *TCP/IP (Parallel Library) Configuration and Management Manual* for information about starting and using the Parallel Library TCP/IP environment. Also, in Parallel Library TCP/IP, if you do want to run multiple SNMP agents in each processor, you must bind the SNMP agents to a specific IP address. To do this, add multiple subnets to the Parallel Library TCP/IP subsystem and then bind your SNMP subagents to the IP addresses associated with those subnets by altering the endpoint and changing the Hostaddr from 0.0.0.0 to a specific IP address.

The NonStop TCP/IPv6 Subagent and its Managed Resources

To run SNMP over TCP/IPv6, you must define the name of a TCP6SAM process. You can find this process by issuing a LISTDEV TCPIP command to find a process whose program name ends in TCP6SAM.

If you want to implement Logical Network Partitions (LNPs), you must start an instance of the SNMP agent for every LNP though which you want to provide SNMP services.

See the *TCP/IPv6 Configuration and Management Manual* for information about starting and using the TCP/IPv6 environment.

Standard MIB-II Groups Supported by the TCP/IP Subagent

The TCP/IP Subagent supports the following MIB-II groups defined in RFC 1213:

This MIB-II Group...	**Provides Information About...**
Interfaces	Network interfaces of the TCP/IP subsystem being managed
IP	Internet Protocol (IP) layer of the TCP/IP subsystem being managed
ICMP	Internet Control Message Protocol (ICMP) layer of the TCP/IP subsystem being managed
TCP	Transmission Control Protocol (TCP) layer of the TCP/IP subsystem being managed
UDP	User Datagram Protocol (UDP) layer of the TCP/IP subsystem being managed

These groups, a subset of the MIB-II groups, are identified by a check mark in the following list:

```
iso (1)
      identified-organization (3)
        dod (6)
          internet (1)
            mgmt (2)
              mib-II (1)
                system (1)
                interfaces (2) √
                at (3)
                ip (4) √
                icmp (5) √
                tcp (6) √
                udp (7) √
                egp (8)
                cmot (9)
                transmission (10)
                snmp (11)
```

The objects in these groups are described in the subsections dedicated to each supported MIB-II group later in this section.

Private MIB Objects Supported by the TCP/IP Subagent

The TCP/IP Subagent also supports private MIB objects that:

- Provide information about the subagent process itself, such as:
  - The name of the subagent process
  - The processor in which subagent's primary process is running
  - The name of the TCP/IP process controlling the managed TCP/IP subsystem
- Allow you to control certain aspects of the subagent's behavior, such as:
  - Causing a backup process to take over and a new backup process to be created
  - Controlling the rate at which MIB-II object values are updated
  - Requesting that the subagent refresh MIB-II object values on demand
- Allow you to monitor information about the subnets associated with the TCP/IP subsystems:
  - Name of the TCP/IP subsystem being monitored
  - Status of the TCP/IP subsystem
  - Name of the subnets associated with the TCP/IP subsystem
  - Status of the subnets
  - IP address of the subnets

These private MIB objects are defined by HP and reside in the ztsa subtree within the nonstopsystems subtree registered to HP:

```
iso (1)
      identified-organization (3)
        dod (6)
          internet (1)
            private (4)
              enterprises (1)
                tandem (169)
                  nonstopsystems (3)
                    ztsa (202)
```

The ztsa objects supported by the TCP/IP Subagent are described in detail in Table 8-3, Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent, on page 8-21.

MIB Value Derivation

The TCP/IP Subagent retrieves and alters the values of most MIB-II objects by exchanging SPI buffers with the HP TCP/IP subsystem. Most values are retrieved from a token, a field within a token, or a word offset within a field extracted from an SPI reply buffer in response to various SPI commands.

If the MIB-II object has a directly corresponding SPI attribute, the TCP/IP Subagent performs a simple SPI interaction with a single subsystem. The TCP/IP Subagent simply encodes and sends an SPI request to the subsystem. When the response is received, the TCP/IP Subagent extracts the desired token, field, or word offset from the reply buffer and returns the obtained value.

Certain requests require more complex SPI interaction. When an intermediate value must be obtained, the value returned is obtained after two or more successive SPI interactions.

A few MIB-II objects have constant values declared. Some objects (mostly counters) are implemented by the TCP/IP Subagent itself.

For MIB values from the SLSA subsystem, ZLANMSRL calls are made, the content buffer returned is processed, and fields are extracted from it.

Specific information about how the TCP/IP Subagent maintains MIB values is provided in later subsections that describe each group in detail.

Unavailable Resources

If the TCP/IP subsystem being managed by the TCP/IP Subagent becomes unavailable (for instance, if the SCP process or the subsystem management or I/O process becomes unavailable), the entire MIB-II table is cleared, and an SNMP noSuchName error is returned to the SNMP manager in the response to the request.

Note. When the managed TCP/IP resources are unavailable, you can still access private TCP/IP Subagent MIB objects to obtain information about and control various aspects of the TCP/IP Subagent.

The value of the TCP/IP Subagent's private ztsaKeepAliveTimer object defines the interval at which the TCP/IP Subagent looks for lost resources. If the SCP process goes away, for example, and ztsaKeepAliveTimer is set to 30, the TCP/IP Subagent attempts to connect to the SCP process every 30 seconds. Refer to Starting the TCP/IP Subagent on page 8-12 for complete information on the ztsaKeepAliveTimer object.

Reporting Values for Uninstrumented Objects

When it receives a request against objects for which there is no HP instrumentation, the TCP/IP Subagent returns the following values:

- A zero (0) for Get and GetNext requests against unsupported MIB objects defined as integers, counters, or gauges
- A null value for Get and GetNext requests against unsupported MIB objects defined as strings

Unsupported objects are identified in the RFC compliance tables following each supported MIB-II group's object definition table.

Refreshing MIB Values

The TCP/IP Subagent does not access TCP/IP data on demand. At startup time and at regular intervals, data is refreshed, stored in a cache, and returned to the manager through the SNMP agent when MIB values are requested.

Cache Refresh Timers

The TCP/IP Subagent supports three configurable timers for refreshing cached MIB values:

- The information cache refresh timer
- The status cache refresh timer
- The statistics cache refresh timer

You set these refresh timer values with the -info, -status, and -stats startup parameters respectively when you start the subagent, as described in Starting the TCP/IP Subagent on page 8-12.

After the subagent is running, you can refresh values on demand by setting the TCP/IP Subagent private ztsaInfoCache, ztsaStatusCache, and ztsaStatsCache objects from an SNMP manager. These objects are described in detail in Table 8-3, Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent, on page 8-21.

Note. HP does not recommend disabling cache timers (by setting their values to 0). If one timer is set to 0, set all timers to 0 for consistency.

The Information Cache Refresh Timer

The information cache or “info cache” timer controls the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI INFO command. This information is relatively static. The info cache timer is initially set with the -info startup parameter and can be managed through SNMP by setting the ztsaInfoCache object.

You can disable the info cache timer by setting the ztsaInfoCache object to 0. A value of 0 indicates that the info cache is refreshed only when the ztsaRefreshNow object is set to 1 (forceDynamicRefresh).

The Status Cache Refresh Timer

The status cache timer controls the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI STATUS command. The status cache timer is initially set with the -status startup parameter and can be managed through SNMP by setting the ztsaStatusCache object.

You can disable the status cache timer by setting the ztsaStatusCache object to 0. A value of 0 indicates that the cache is refreshed only when the ztsaRefreshNow object is set to 1 (forceDynamicRefresh).

The Statistics Cache Refresh Timer

The statistics cache or “stats cache” timer controls the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI STATISTICS command. Statistical data is relatively volatile. The stats cache timer is initially set with the -stats startup parameter and can be managed through SNMP by setting the ztsaStatsCache object.

You can disable the stats cache timer by setting the ztsaStatsCache object to 0. A value of 0 indicates that the cache is refreshed only when the ztsaRefreshNow object is set to 1 (forceDynamicRefresh).

Dynamic Refresh

Setting the TCP/IP Subagent private ztsaRefreshNow object to 1 forces a refresh of the MIB-II irrespective of the refresh timer settings. Once the MIB-II values have been updated, the value of ztsaRefreshNow returns to 0 (autoDynamicRefresh), and the values in the refresh timer objects (ztsaInfoCache, ztsaStatusCache, and ztsaStatsCache) once again determine the rate at which MIB-II values are refreshed.

These objects are described in detail in Table 8-3, Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent, on page 8-21.

Initiating Backup Process Takeover

The TCP/IP Subagent can be run as a process pair to achieve basic fault tolerance. The support includes a persistent subagent process and checkpointing of the startup parameters.

To run the TCP/IP Subagent as a process pair and have the backup process take over in case the primary process fails, include the -b startup parameter when you start the subagent. For example:

```
RUN TCPIPSA /NAME $ZTSA, NOWAIT/ -b 6
```

Note. You must specify a backup CPU. Otherwise the TCP/IP Subagent does not run as a process pair.

You can force the backup process to take over and a new backup process to be created by setting the value of the ztsaSwitchToBackupNow object to 1 (forceBackupTakeover) from an SNMP manager.

Related Documents

The *TCP/IP Management Programming Manual* provides information to help interpret the meanings of values for MIB-II group objects supported by the TCP/IP Subagent.

RFC Compliance

The following tables contain information about how TCP/IP Subagent support for each of its MIB-II groups complies with RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*:

- Table 8-6, Compliance With Interfaces Group Definitions in RFC 1213
- Table 8-8, Compliance With IP Group Definitions in RFC 1213
- Table 8-10, Compliance With ICMP Group Definitions in RFC 1213
- Table 8-12, Compliance With TCP Group Definitions in RFC 1213
- Table 8-14, Compliance With UDP Group Definitions in RFC 1213

Installing the TCP/IP Subagent

When the TCP/IP Subagent runs on NonStop systems, these products must also be configured and running:

- SNMP agent
- EMS
- TCP/IP G00.00 or later for G-series systems
- At least one SNMP manager that can send requests to the SNMP agent for processing by the TCP/IP Subagent

Installation Steps

Run the Install or Distributed Systems Management/Software Configuration Manager (DSM/SCM) product to install the SNMP agent and the TCP/IP Subagent.

Then load the ASN.1 source code for the TCP/IP Subagent's private MIB onto any SNMP manager station from which you want to monitor TCP/IP resources. The MIB definitions are contained in two files:

- RFC1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II*, contains definitions for the MIB-II objects.
- The **ZTSAMIB** file contains definitions for the private TCP/IP Subagent MIB objects defined by HP.

Compile the ZTSAMIB file (and the RFC1213 file if necessary) as described in the documentation provided with your SNMP manager. Compiling the MIB makes it possible for the SNMP manager to display the names (rather than only numeric object identifiers) of MIB objects.

Once the MIB definition files have been compiled, the TCP/IP Subagent can be started.

Before Starting the TCP/IP Subagent

The SNMP agent must be running before you start the TCP/IP Subagent. Start the SNMP agent with the RUN command, as described in Section 2, Installing and Configuring the SNMP Agent. For example:

```
RUN SNMPAGT /NAME $agent-process, NOWAIT/
```

If the SNMP agent is not available when the TCP/IP Subagent is started, the subagent fails to start. If the SNMP agent disappears after the TCP/IP Subagent starts, the subagent enters a loop trying to reconnect to the SNMP agent at the rate specified in its ztsakeepAliveTimer object.

In addition, for the TCP/IP Subagent to communicate with a HP TCP/IP subsystem, the SCP process ($ZNET, or the SCP process specified for the -scp startup parameter) must be running. If the SCP process is not available, the TCP/IP Subagent enters a loop trying to reconnect to it at the rate specified in its ztsaKeepAliveTimer object.

Note. You can access the TCP/IP Subagent’s private MIB objects regardless of the status of the SCP process.

Starting the TCP/IP Subagent

Start the TCP/IP Subagent using the TACL RUN command. Ensure that you specify the name of a running SNMP agent. Following is an example of starting a TCP/IP Subagent named $ZTSA, with a backup process running in processor 6, that communicates with SNMP agent $ZSMP1:

```
RUN TCPIPSA /NAME $ZTSA, NOWAIT/ -a $ZSMP1 -b 6
```

To ensure that it can retrieve data from the HP TCP/IP subsystem, the TCP/IP Subagent process should be run at a priority equal to or less than that of the SCP process with which it is communicating.

Syntax of the startup options follows established conventions used in the Guardian environment.

```
[RUN] [[$volume.]subvolume.]TCPIPSA
   / NAME $tcpip-subagent-process [,other-run-option]... /
   [ ? | startup-parameter [startup-parameter]...]
```

volume

identifies the volume on which the TCP/IP Subagent program file (TCPIPSA) resides. You can omit it if TCPIPSA resides on your current subvolume. By default, the Install or DSM/SCM program puts TCPIPSA into $SYSTEM.

subvolume

identifies the subvolume on which TCPIPSA resides. You can omit it if it is named in your TACL #PMSEARCHLIST. By default, the Install or DSM/SCM program puts TCPIPSA into SYSTEM.

tcpip-subagent-process

identifies the TCP/IP Subagent process. You can specify from one through five alphanumeric characters, but the first character must be alphabetic.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the NOWAIT option so that you can resume TACL operations once the subagent is started.

Note. Run the TCP/IP Subagent process at a priority equal to or less than that of the SCP process with which it is communicating.

?

displays help information.

startup-parameter

is one of the following parameters with which you can control attributes of the TCP/IP Subagent process.

Note. Each of the TCP/IP Subagent startup parameters corresponds to one of its private (ztsa) MIB objects, as indicated in the following syntax diagram and described in Configuring a Running TCP/IP Subagent on page 8-16.

```
-a [\node.]$agent-process                   |ztsaAgentName
-b backup-cpu-number                        |ztsaBackupCPU
-c [\node.]$collector-process               |ztsaEmCollectorName
-info info-cache-timer                      |ztsaInfoCache
-k keep-alive-timer                         |ztsaKeepAliveTimer
-s [($tcpip-subsystem1[,...,
$tcpip-subsystemn])]                        |ztsaTcpIpName
-scp $scp-process                           |ztsaScpProcess
-stats stats-cache-timer                    |ztsaStatsCache
-status status-cache-timer                  |ztsaStatusCache
-t $tcpip-process                           |ztsaTcpIpProcess
```

`-a [\`*`node`*`.]$`*`agent-process`*

is the name of the SNMP agent process with which you want the TCP/IP Subagent to communicate.

The default value is $ZSNMP on the same node as the subagent.

`-b` *`backup-cpu-number`*

is the processor number in which a backup process is to be created.

The default value is -1, indicating that no backup is to be created.

`-c [\`*`node`*`.]$`*`collector-process`*

is the name of the collector to which the subagent is to send EMS event messages it generates.

The default is $0, the primary EMS collector, on the local node.

Note. EMS event generation is always enabled at process startup. To disable the sending of events, set the value of the ztsaEmsCollectorState object to disabled (3) once the TCP/IP Subagent is running.

`-info` *`info-cache-refresh-timer`*

specifies, in seconds, the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI INFO command.

The range for *`info-cache-refresh`* is 0 to 32000. A value of 0 disables this timer, indicating that SPI INFO-derived MIB values are refreshed only when the ztsaRefreshNow object is set to forceDynamicRefresh (1).

The default is 180 seconds.

`-k` *`keep-alive-timer`*

specifies the time interval, in seconds, between attempts by the TCP/IP Subagent to reconnect to a TCP/IP, SNMP agent, backup, or EMS collector process that is in the disconnected state.

The default value is 60.

Note. This timer value does not apply to objects that are placed in the disabled state by an SNMP Set request. See State Object/Resource Object Pairs on page 8-28 for information on the possible states that can be set for processes used by the TCP/IP Subagent.

`-s` *`[($tcpip-subsystem1[,...,$tcpip-subsystemn])]`*

identifies the TCP/IP subsystem(s) for which the subnets information is to be instrumented. If you do not specify any TCP/IP subsystems, the TCP/IP subagent instruments the information about the subnets of all the TCP/IP subsystems.

Users of the -s option can monitor only a the following private MIB objects for information about subnets of the TCP/IP processes: TCPIP process name, TCPIPstatus, subnet name, subnet status, and subnet address. Users cannot monitor all MIB-II entries of the TCP/IP processes.

The maximum number of TCP/IP subsystems that can be monitored using this option is 32.

Note. If you omit the -s startup parameter the instrumentation of new private MIB objects will be suppressed.

`-scp $`*`scp-process`*

is the name of the SCP process to be used for SPI requests for deriving MIB object values. The SCP process named must be running and on the same node as the subagent process.

The default is $ZNET.

`-stats` *`stats-cache-refresh`*

specifies, in seconds, the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI STATISTICS command.

The range for *`stats-cache-refresh`* is 0 to 32000. A value of 0 disables this timer, indicating that SPI STATISTICS-derived MIB values are only refreshed when the ztsaRefreshNow object is set to forceDynamicRefresh (1).

The default is 60 seconds.

`-status` *`status-cache-refresh`*

specifies, in seconds, the rate at which the TCP/IP Subagent updates values of objects that it retrieves by issuing a SPI STATUS command.

The range for *`status-cache-refresh`* is 0 to 32000. A value of 0 disables this timer, indicating that SPI STATUS-derived MIB values are refreshed only when the ztsaRefreshNow object is set to forceDynamicRefresh (1).

The default is 90 seconds.

`-t $`*`tcpip-process`*

is the name of the TCP/IP process the subagent is to query. The TCP/IP process must be running and on the same node as the subagent process. For Parallel Library TCP/IP and NonStop TCP/IPv6, the TCPSAM or TCP6SAM process must be running, but does not have to be on the same node as the subagent.

The default is $ZTC0.

See The TCP/IP Subagent and Its Managed Resources on page 8-4 for further considerations for NonStop TCP/IP. See the *TCP/IP (Parallel Library)*

Configuration and Management Manual for more information about Parallel Library TCP/IP. For more information about NonStop TCPIPv6, see the *TCP/IPv6 Configuration and Management Manual*.

Stopping the TCP/IP Subagent

Issue a TACL STOP command to stop a TCP/IP Subagent process. For example:

```
STOP $ZTSA
```

Configuring a Running TCP/IP Subagent

Once a TCP/IP Subagent is running, you can issue SNMP Get and Set requests from an SNMP manager to query and control the subagent.

Querying a Running TCP/IP Subagent

Table 8-1. Querying a TCP/IP Subagent Through SNMP (page 1 of 2)

You can find out this about the TCP/IP Subagent...	By issuing an SNMP Get request against this ztsa object...
SNMP agent process with which subagent is communicating	ztsaAgentName
State of the connection between subagent and SNMP agent	ztsaAgentState
Name of subagent process	ztsaProcessName
Version of subagent process object file	ztsaProcessVersion
Name of subagent program file	ztsaProgramFile
Current process priority of subagent process	ztsaProcessPriority
Group ID under which subagent was started	ztsaProcessGroupID
User ID under which subagent was started	ztsaProcessUserID
Processor in which subagent’s primary process is running	ztsaPrimaryCPU
Process number of subagent’s primary process	ztsaPrimaryPIN
Processor in which subagent’s backup process is running or will start	ztsaBackupCPU
Process number of subagent’s backup process	ztsaBackupPIN
State of subagent’s backup process	ztsaBackupState
EMS collector process to which subagent sends events	ztsaEmsCollectorName
State of the connection to the EMS collector	ztsaEmsCollectorState
Rate at which subagent makes reconnection attempts	ztsaKeepAliveTimer
Rate at which SPI STATISTICS based object values are refreshed	ztsaStatsCache
Rate at which SPI STATUS based object values are refreshed	ztsaStatusCache

* Name of the TCP/IP subsystem(s) monitored using the -s startup option.

Table 8-1. Querying a TCP/IP Subagent Through SNMP (page 2 of 2)

You can find out this about the TCP/IP Subagent...	By issuing an SNMP Get request against this ztsa object...
Rate at which SPI INFO based object values are refreshed	ztsaInfoCache
TCP/IP process being monitored	ztsaTcpIpProcess
State of the connection between subagent and monitored TCP/IP process	ztsaTcpIpState
SCP process used for SPI requests	ztsaScpProcess
Name of the TCP/IP subsystem* being monitored	ztsaTcpIpName
Status of the TCP/IP subsystem	ztsaTcpIpStatus
Name of the subnets associated with the TCP/IP subsystem	ztsaSubnetName
Status of the subnets	ztsaSubnetStatus
IP address of the subnets	ztsaSubnetAddr

* Name of the TCP/IP subsystem(s) monitored using the -s startup option.

These objects are described in detail in Table 8-3, Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent, on page 8-21.

Controlling a Running TCP/IP Subagent

In addition to getting information about a running TCP/IP Subagent, you can control certain aspects of its behavior by issuing SNMP Set requests from an SNMP manager, as described in Table 8-2.

All subagent attributes that can be controlled by setting MIB object values are reinitialized when the subagent is restarted to default values or values specified in startup parameters.

Table 8-2. Controlling a TCP/IP Subagent Through SNMP (page 1 of 2)

You can control this attribute of a TCP/IP Subagent...	By issuing SNMP Set requests against this ztsa object...	Initially set with this startup parameter...	With this default value...
Current process priority of subagent process	ztsaProcessPriority		None; determined by operating system.
Processor in which subagent's backup process is running or will start	ztsaBackupCPU	-b	-1 (no backup process) The ztsaBackupState* object must be set to disabled (3) before the ztsaBackupCPU object can be set.
State of the subagent's backup process	ztsaBackupState		None.
Switching to the backup subagent process	ztsaSwitchToBackup Now		0 (autoProcessPairControl) at startup.
EMS collector process to which subagent sends events	ztsaEmsCollectorName	-c	$0 on the same node as the subagent. ztsaEmsCollectorState* must be set to disabled (3) before the ztsaEmsCollectorName object can be set.
State of the connection to the EMS collector	ztsaEmsCollectorState		None; in normal circumstances, value is "connected."
Rate at which subagent makes reconnection attempts	ztsaKeepAliveTimer	-k	60 seconds between attempts made by the subagent to connect to a SNMP agent or TCP/IP, or to restart the backup process.
Cause subagent to update MIB tables and generate traps if needed	ztsaRefreshNow		0 (autoDynamicRefresh) at startup.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-2. Controlling a TCP/IP Subagent Through SNMP (page 2 of 2)

You can control this attribute of a TCP/IP Subagent...	By issuing SNMP Set requests against this ztsa object...	Initially set with this startup parameter...	With this default value...
Rate at which SPI STATISTICS based object values are refreshed	ztsaStatsCache	-stats	60 seconds.
Rate at which SPI STATUS based object values are refreshed	ztsaStatusCache	-status	90 seconds.
Rate at which SPI INFO based object values are refreshed	ztsaInfoCache	-info	180 seconds.
TCP/IP process being monitored	ztsaTcpIpProcess	-t	$ZTC0 on the same node as the subagent. ztsaTcpIpState* must be set to disabled (3) before the ztsaTcpIpProcess object can be set.
State of the connection to the monitored TCP/IP process	ztsaTcpIpState		None: in normal circumstances, value is “connected.”

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

These read-write objects are described in detail in Table 8-3, Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent, on page 8-21.

ZTSA MIB Objects

The following collection of scalar objects in the ztsa subtree provide information about the TCP/IP Subagent and allow you to control some aspects of its behavior:

```
iso (1)
      identified-organization (3)
        dod (6)
          internet (1)
            private (4)
              enterprises (1)
                tandem (169)
                  nonstopsystems (3)
                    ztsa (202)
                      ztsaAgentName (1)
                      ztsaAgentState (2)
                      ztsaProcessName (3)
                      ztsaProcessVersion (4)
                      ztsaProgramFile (5)
                      ztsaProcessPriority (6)
                      ztsaProcessGroupID (7)
                      ztsaProcessUserID (8)
                      ztsaPrimaryCPU (9)
                      ztsaPrimaryPIN (10)
                      ztsaBackupCPU (11)
                      ztsaBackupPIN (12)
                      ztsaBackupState (13)
                      ztsaSwitchtoBackupNow (14)
                      ztsaEmsCollectorName (15)
                      ztsaEmsCollectorState (16)
                      ztsaKeepAliveTimer (17)
                      ztsaRefreshNow (18)
                      ztsaStatsCache (19)
                      ztsaStatusCache (20)
                      ztsaInfoCache (21)
                      ztsaTcpIpProcess (22)
                      ztsaTcpIpState (23)
                      ztsaScpProcess (25)
                      zstaTcpIpTable (26)
                        zstaTableEntry (1)
                          zstaTcpIpName (1)
                          zstaTcpIpStatus (2)
                      zstaSubnetTable (27)
                        zstaSubnetEntry (1)
                          zstaSubnetName (1)
                          zstaSubnetStatus (2)
                          zstaSubnetAddr (3)
```

Table 8-3 defines each object in the ztsa subtree.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 1 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaAgentName 1.3.6.1.4.1.169.3.202.1 read-only DisplayString (SIZE (0..31))	The name of the SNMP agent process with which the TCP/IP Subagent is communicating.	\node.$*agent-process* specified for the -a startup parameter. If none specified, $ZSNMP on the local node.
ztsaAgentState* 1.3.6.1.4.1.169.3.202.2 read-only INTEGER	The status of the interface between the TCP/IP Subagent and the SNMP agent.	A value of connected (1) is always returned in a response message to a management request, indicating that the TCP/IP Subagent is communicating with the SNMP agent.
ztsaProcessName 1.3.6.1.4.1.169.3.202.3 read-only DisplayString (SIZE (0..49))	The fully qualified NonStop Kernel process name of this subagent.	\node.$*process*:*seq-no* identifying the process name specified for the TACL NAME option when the TCP/IP Subagent was started, and the sequence number assigned to the process by the NonStop Kernel.
ztsaProcessVersion 1.3.6.1.4.1.169.3.202.4 read-only DisplayString (SIZE (0..255))	The version of the TCP/IP Subagent process object file.	T7862D*nn*_date -- HP SNMP TCP/IP Subagent, where D*nn* is the product version of the TCP/IP Subagent; for example, D20.
ztsaProgramFile 1.3.6.1.4.1.169.3.202.5 read-only DisplayString (SIZE (0..32))	The name of the TCP/IP Subagent program file.	\node.$*vol*.*subvol*.*filename* returned by a Guardian PROCESS_GETINFO_ procedure call.
ztsaProcessPriority 1.3.6.1.4.1.169.3.202.6 read-only INTEGER (0..199)	The priority of the TCP/IP Subagent process.	Initially determined by the NonStop Kernel or from a TACL run option. After startup, the value can be set by an SNMP manager.
ztsaProcessGroupID 1.3.6.1.4.1.169.3.202.7 read-only INTEGER (0..255)	The group number of the user who started the TCP/IP Subagent.	*group-ID-number* returned by a Guardian PROCESS_GETINFO procedure call.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 2 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaProcessUserID 1.3.6.1.4.1.169.3.202.8 read-only INTEGER (0..255)	The user ID number of the user who started the TCP/IP Subagent.	*`user-ID-number`* returned by a Guardian PROCESS_GETINFO procedure call.
ztsaPrimaryCPU 1.3.6.1.4.1.169.3.202.9 read-only INTEGER (0..255)	The number of the processor in which the primary TCP/IP Subagent process is running.	*`cpu-number`* returned by a Guardian PROCESS_GETPAIRINFO procedure call.
ztsaPrimaryPIN 1.3.6.1.4.1.169.3.202.10 read-only INTEGER (0..255)	The process identification number (PIN) of the primary TCP/IP Subagent process.	*`PIN-number`* returned by a Guardian PROCESS_GETPAIRINFO procedure call.
ztsaBackupCPU 1.3.6.1.4.1.169.3.202.11 read-write INTEGER (0..255)	The number of the processor in which the backup process will start or is running.	*`cpu-number`* specified for the -b startup parameter. After startup, the value can be set by an SNMP manager. A value of -1 indicates the backup is not configured. Setting this object is permitted only if ztsaBackupState* is set to disabled (3). Once ztsaBackupCPU has been set to a new value, ztsaBackupState must be set to enabled (4) for the new value to take effect.
ztsaBackupPIN 1.3.6.1.4.1.169.3.202.12 read-only INTEGER (0..255)	The process identification number (PIN) of the backup TCP/IP Subagent process.	*`PIN-number`* returned by a Guardian PROCESS_GETPAIRINFO procedure call. A value of -1 indicates that the backup is not running.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 3 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaBackupState* 1.3.6.1.4.1.169.3.202.13 read-write INTEGER	The state of the TCP/IP Subagent backup process.	Following are valid states for the backup process: connected (1) disconnected (2) disabled (3) enabled (4) This value can be set with an SNMP request as illustrated by the state change diagram in Figure 8-3 on page 8-29.
ztsaSwitchToBackupNow 1.3.6.1.4.1.169.3.202.14 read-write INTEGER	Causes the TCP/IP Subagent backup process to take over and a new backup process to be created.	One of these values: autoProcessPairControl (0) forceBackupTakeover (1) At startup, the subagent sets this value to 0, but it can be changed by an SNMP manager if the TCP/IP Subagent was started using the -b startup parameter. Setting this object to 1 causes the TCP/IP Subagent backup process to take over and a new backup process to be created. Setting this object is permitted only if ztsaBackupState* is connected (1).
ztsaEmsCollectorName 1.3.6.1.4.1.169.3.202.15 read-write DisplayString (SIZE (0..15))	The name of the EMS collector process to which the TCP/IP Subagent sends events it generates.	\node.$*ems-collector* specified for the -c startup parameter. After startup, the value can be set by an SNMP manager. Setting this object is permitted only if ztsaEmsCollectorState* is disabled (3).

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 4 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaEmsCollectorState* 1.3.6.1.4.1.169.3.202.16 read-write INTEGER	The state of the connection between the TCP/IP Subagent and the EMS collector.	Following are valid values for the state of the connection: connected (1) disconnected (2) disabled (3) enabled (4) The connected and disconnected states are set internally. The disabled and enabled states are set with an SNMP request as illustrated by the state change diagram in Figure 8-3 on page 8-29.
ztsaKeepAliveTimer 1.3.6.1.4.1.169.3.202.17 read-write INTEGER (0..32000)	The number of seconds between attempts the subagent makes to connect to a lost resource or to restart the backup process.	A value in the range 0 to 32000 specified for the -k startup parameter. After startup, the value can be set by an SNMP manager. The default is 60 seconds.
ztsaRefreshNow 1.3.6.1.4.1.169.3.202.18 read-write INTEGER	A means of forcing an update of the MIB-II tables and generating traps (if needed).	One of these values: autoDynamicRefresh (0) forceDynamicRefresh (1) This object normally reads 0, indicating that the MIB values are refreshed at the rates indicated by the values of the ztsaStatsCache, ztsaInfoCache, and ztsaStatusCache timers. Setting this object to 1 causes the subagent to update the MIB tables and generate traps (if needed). The object returns a value of 1 until the refresh is complete.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 5 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaStatsCache 1.3.6.1.4.1.169.3.202.19 read-write INTEGER (0..32000)	The cache update interval (in seconds) for objects updated using the SPI STATISTICS command.	A value in the range 0 to 32000 specified for the -stats startup parameter. The default is 60 seconds. After startup, the value can be set by an SNMP manager. Value 0 disables cache timing such that updates occur to objects derived from STATISTICS only when ztsaRefreshNow is set to forceDynamicRefresh (1).
ztsaStatusCache 1.3.6.1.4.1.169.3.202.20 read-write INTEGER (0..32000)	The cache update interval (in seconds) for SPI STATUS command based objects.	A value in the range 0 to 32000 specified for the -status startup parameter. The default is 90 seconds. After startup, the value can be set by an SNMP manager. Value 0 disables cache timing such that updates occur to objects derived from STATUS only when ztsaRefreshNow is set to forceDynamicRefresh (1).
ztsaInfoCache 1.3.6.1.4.1.169.3.202.21 read-write INTEGER (0..32000)	The cache update interval (in seconds) for SPI INFO command based objects.	A value in the range 0 to 32000 specified for the -info startup parameter. The default is 180 seconds. After startup, the value can be set by an SNMP manager. Value 0 disables cache timing such that updates occur to objects derived from INFO only when ztsaRefreshNow is set to forceDynamicRefresh (1).
ztsaTcpIpProcess 1.3.6.1.4.1.169.3.202.22 read-write DisplayString (SIZE (0..15))	The name of the TCP/IP process monitored by the subagent.	$*tcpip-process* specified for the -t startup parameter. The default is $ZTC0. After startup, the value can be set by an SNMP manager.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 6 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaTcpIpState* 1.3.6.1.4.1.169.3.202.23 read-write INTEGER	The state of the connection between the TCP/IP Subagent and the monitored TCP/IP process.	Following are valid values for the state of the connection: connected (1) disconnected (2) disabled (3) enabled (4) The connected and disconnected states are set internally. The disabled and enabled states are set with an SNMP request as illustrated by the state change diagram in Figure 8-3 on page 8-29.
ztsaScpProcess 1.3.6.1.4.1.169.3.202.25 read-only DisplayString (SIZE (0..15)	The name of the SCP process used for SPI requests.	$*scp-process* specified for the -scp startup parameter. The default is $ZNET.
ztsaTcpIpTable 1.3.6.1.4.1.169.3.202.26 not-accessible	The table that holds information about the TCP/IP subsystems.	Contains up to 32 ztsaTableEntry objects.
ztsaTableEntry 1.3.6.1.4.1.169.3.202.26.1 not-accessible	The entry that contains information related to the TCP/IP subsystems.	Contains the ztsaTcpIpName and ztsaTcpIpStatus fields.
ztsaTcpIpName 1.3.6.1.4.1.169.3.202.26.1.1 read-only DisplayString	The TCP/IP subsystem name monitored by the TCP/IP SA.	$*tcpip-process* specified for the -s start up parameter.
ztsaTcpIpStatus 1.3.6.1.4.1.169.3.202.26.1.2 read-only INTEGER	The status of the TCP/IP subsystem monitored by the TCP/IP SA.	One of these values: up (1) down (2) The value (up/down) depends on the status of the ztsaTcpIpName.
ztsaSubnetTable 1.3.6.1.4.1.169.3.202.27 not-accessible	The table that holds the subnet information associated with the TCP/IP subsystems being monitored.	Contains the ztsaSubnetEntry objects.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 8-3. Private (ZTSA) MIB Objects Supported by the TCP/IP Subagent (page 7 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ztsaSubnetEntry 1.3.6.1.4.1.169.3.202.27.1 not-accessible	The entry that contains information related to the subnets associated with the TCP/IP subsystems.	Contains the ztsaSubnetName, ztsaSubnetAddr and ztsaSubnetStatus fields.
ztsaSubnetName 1.3.6.1.4.1.169.3.202.27.1.1 read-only DisplayString	The subnet name corresponding to the TCP/IP subsystem monitored by TCP/IP SA.	The value ZCOM-TKN-OBJNAME returned from SPI INFO SUBNET command on subsystem $*tcpip-subsystem*.
ztsaSubnetStatus 1.3.6.1.4.1.169.3.202.27.1.2 read-only INTEGER	The status of the subnet corresponding to the TCP/IP subsystem monitored by TCP/IP SA.	One of these values: aborting(0), defined(1), diagnosis(2), started(3), starting(4), stopped(5), stopping(6), suspended(7), unknown(8), suspending(9) Contains the status in the ZSUBNET-STATE field of the ZTCI-MAP-STATUS-SUBNET structure, returned from a SPI STATUS SUBNET command on $*tcpip-subsystem*.
ztsaSubnetAddr 1.3.6.1.4.1.169.3.202.27.1.3 read-only IpAddress	The IP address of the subnet corresponding to the TCP/IP subsystem monitored by TCP/IP SA.	Contains the IP address in the ZIP-ADDR field of the ZTCI_MAP_INFO_SUBNET structure, returned from a SPI INFO SUBNET command on $*tcpip-subsystem*.

* See State Object/Resource Object Pairs on page 8-28 for information about TCP/IP Subagent objects that control or indicate the internal processing state of processes used by the subagent.

State Object/Resource Object Pairs

The private TCP/IP Subagent MIB objects that control or indicate the internal processing state of other objects are referred to as “state objects.” The objects whose internal processing states are indicated or controlled by state objects are referred to as “resource objects.”

Table 8-4 lists the resource objects and their associated state objects and state values.

Table 8-4. TCP/IP Subagent State Object/Resource Object Pairs

The operational state of this resource object...	Is indicated or controlled by this state object...	Which can contain one of the following states...
ztsaAgentName	ztsaAgentState	Connected (1) Disconnected (2)
ztsaBackupCPU ztsaBackupPIN	ztsaBackupState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)
ztsaTcpIpProcess	ztsaTcpIpState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)
ztsaEmsCollectorName	ztsaEmsCollectorState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)

State objects cannot be set to “Connected” and “Disconnected” with a Set request. They are internally generated states indicating the state of the connection between the TCP/IP Subagent process and the process represented by the resource object. “Disabled” and “Enabled” are SNMP Set request-generated states used when resource objects are being reconfigured.

The meanings of each state is as follows:

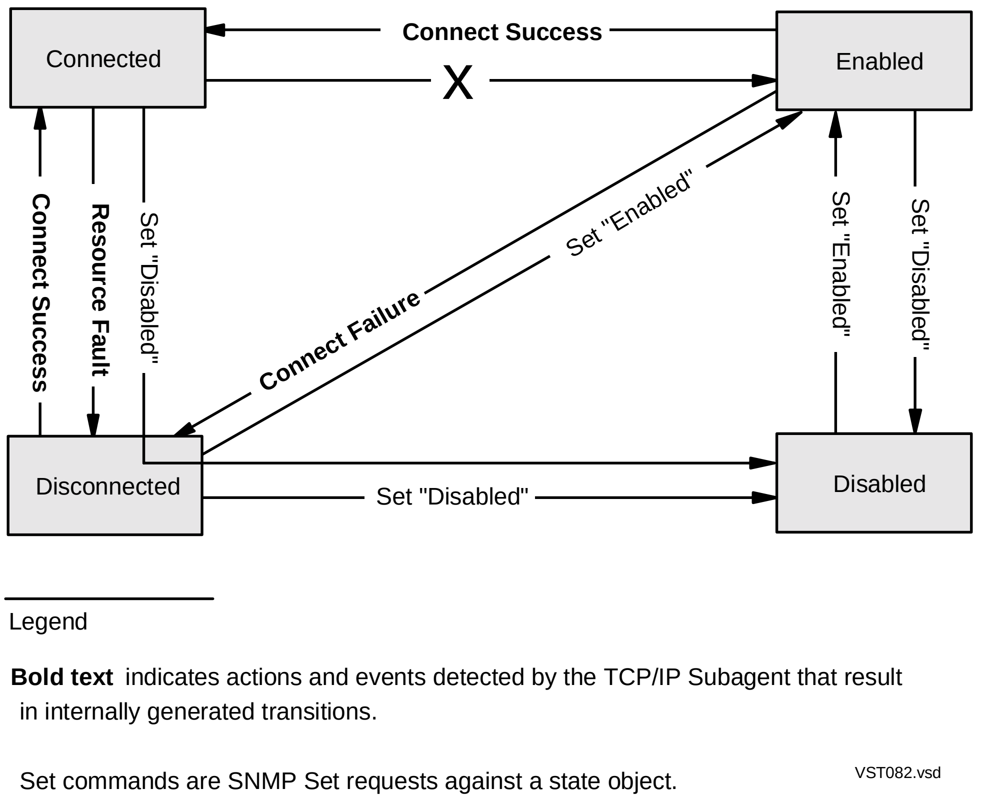
Connected	The subagent is functioning normally.
Disconnected	Subagent function dependent on the process or interface is not available.
Enabled	The subagent will immediately attempt to establish connections with the process or interface.
Disabled	The process or interface has been taken out of service explicitly by an operator.

State Transitions for Resource Objects

Figure 8-3 illustrates how resource objects move between states. Note that:

- Setting a “Connected” object to “Disabled” forces a transition through the “Disconnected” state. This permits the orderly shutdown of the resource and the generation of a state change event for the disconnected state.
- “Connect Success,” “Connect Failure,” and “Resource Fault” all relate to actions and events detected by the subagent. Internally generated transitions produce an EMS event indicating the state change, as described in Event Message Descriptions on page 8-70.
- An SNMP general error is returned if the TCP/IP Subagent receives a Set “enabled” request against a state object that is in the connected state.

Figure 8-3. TCP/IP Subagent Resource Object States



Modifying the Value of a Resource Object

To modify the value of a resource object:

1. Set the value of resource object’s associated state object to “Disabled.”
2. Set the resource object to a new value.
3. Set the value of the resource object’s associated state object to “Enabled.”

When the state object has been placed in the “Enabled” state, the TCP/IP Subagent attempts to connect to the underlying process.

- If the connection attempt is successful, the object enters the “Connected” state.
- If the connection attempt fails, the object enters the “Disconnected” state. The TCP/IP Subagent continues to attempt to reconnect at the rate specified for the ztsaKeepAliveTimer object.

Interfaces Group

The Interfaces group contains information about the interfaces of the TCP/IP subsystem being managed. An interface is defined as the association between a TCP/IP subnet and the underlying I/O process that controls access to the LAN or X25 network.

The Interfaces group consists of one scalar object and one table (the ifTable). The objects identified by a check mark in the following list are objects for which there is corresponding HP instrumentation:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              interfaces (2) √
                ifNumber (1)  √
                ifTable (2) √
                  ifEntry (1) √
                    ifIndex (1) √
                    ifDescr (2) √
                    ifType (3) √
                    ifMtu (4) √
                    ifSpeed (5) √
                    ifPhysAddress (6) √
                    ifAdminStatus (7)
                    ifOperStatus (8) √
                    ifLastChange (9) √
                    ifInOctets (10) √
                    ifInUcastPkts (11) √
                    ifInNUcastPkts (12) √
                    ifInDiscards (13) √
                    ifInErrors (14) √
                    ifInUnknownProtos (15) √
                    ifOutOctets (16) √
                    ifOutUcastPkts (17) √
                    ifOutNUcastPkts (18) √
                    ifOutDiscards (19) √
```

ifOutErrors (20) √
ifOutQLen (21) √
ifOutSpecific (22)

MIB Objects

Table 8-5 describes how the TCP/IP Subagent supports objects in the MIB-II Interfaces group that have HP instrumentation.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 1 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifNumber 1.3.6.1.2.1.2.1 read-only INTEGER	The number of network interfaces of the TCP/IP resources managed or used by the subagent since the subagent was started. Corresponds to the number of supported interfaces comprising the ifTable.	Integer with maximum value 32,767 returned from a SPI INFO command on SUBNET *`ztsaTcpIpProcess`*.*
ifTable Objects:	Entries describing the interfaces of the TCP/IP resources being managed.	Refer to ifTable Maintenance on page 8-38 for information on how entries are created and updated.
ifIndex 1.3.6.1.2.1.2.2.1.1 read-only INTEGER	Index number for each network interface listed in the ifTable.	Integer ranging from 1 to value of ifNumber, computed by subagent.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 2 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifDescr 1.3.6.1.2.1.2.2.1.2 read-only DisplayString (SIZE (1..255))	Identification of a physical interface.	$ZZLAN.`lif` *zzz* , where $`line` is the name of a TLAM or X.25 line associated with a TCP/IP subnet. *xxx* is the server banner returned from a GETVERSION command on a TLAM LINE object. *yyy* is the server banner returned from a GETVERSION command on a TLAM CONTROLLER object. `lif` is a logical interface name for an SLSA device. *zzz* is the version of the LAN Monitor (LANMON) process. If multiple versions of LANMON are running, the version running on the first CPU that is not down is used. Server banners identify the product name, product number, release and compilation dates. For X.25 interface type, only $`line` is returned.
ifType 1.3.6.1.2.1.2.2.1.3 read-only INTEGER	The type of interface, distinguished according to the physical/link protocols immediately below the network layer in the protocol stack.	The ZTYPE field of the ZTCI-DDL-INFO-SUBNET structure returned from the SPI INFO command on SUBNET $*ztsaTcpIpProcess*.* maps to: 6 (Ethernet-csmacd) 24 (loopback) If this field returns a value of SNAP, a further SPI STATUS command is made on the CONTROLLER $*controller* returned by the previous command and the ZCTLR-FIELD returned is mapped to: 7 (iso88023-csmacd)) 9 (token ring) X.25 interface types are not supported in this implementation.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 3 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifMtu 1.3.6.1.2.1.2.2.1.4 read-only INTEGER	The size of the largest datagram that can be sent or received on the interface, in octets.	The value depends on the ifType value, as follows: ifType — ifMtu 6 or 7 — On D series, value of ZMAX-REQ-SIZE field of ZLAM-DDL-INFO-PORT structure 6 — On G series (SLSA), ifMtu obtained using the call LM_Get_Attributes_ on PIF name 9 — ifMtu is the value of ZMAX-REQ-SIZE field of the ZLAM -DDL-INFO-PORT structure obtained from the SPI INFO command on PORT #SNAP of the TLAM process 24 — 0 A SPI INFO command on SUBNET *ztsaTcpIpProcess*.* returns a corresponding TLAM process name. A SPI INFO command on PORT #IP of the TLAM process returns ZMAX-REQ-SIZE value. The largest value possible is 32,000 bytes. ifType 5 (X.25) is not supported in this implementation). If ifType is 24 (loopback), ifMtu is set to 0 to alert the SNMP manager that this value is meaningless for a loopback interface.
ifSpeed 1.3.6.1.2.1.2.2.1.5 read-only Gauge	An estimate of the interface's current bandwidth, in bits per second.	Constant value that depends on the ifType value, as follows: ifType — ifSpeed 5 (X.25) — 0 6 (TLAM) — 10,000,000 6 (SLSA) — call LM_Get_Attributes_ on PIF name 7 (88023) — 10,000,000 9 (tokenring) — value of ZRING-SPEED field of ZLAM-DDL-INFO-SERV-8023 structure 24 (loopback) — 0

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 4 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifPhysAddress 1.3.6.1.2.1.2.2.1.6 read-only PhysAddress	The interface's address at the protocol layer immediately below the network layer in the protocol stack.	The value depends on the ifType value, as follows: <u>ifType</u> — <u>ifPhysAddress</u> 6 (Ethernet) — On D series, the value of ZMFG_ID field of ZLAM-DDL-INFO-SERV-8022 structure 6 (Ethernet) — On G series (SLSA), the call LM_Get_Attributes_ on PIF name is used 7 (88023) — value of ZINIT_NET-ADDR field of ZLAM-DDL-INFO-SERV-8023 structure 9 (tokenring) — value of ZNODE-MAC-ADDR field of ZLAM-DDL-INFO-SERV-8025 structure 24 (loopback) — octet string of 0 length A SPI INFO command on SUBNET `ztsaTcpIpProcess`.* returns a corresponding TLAM process name. A SPI INFO command on SERVICE #8022 (for Ethernet) or 8023 (for SNAP) of the TLAM process returns current attribute values for the #L28022 or #L18023 services, respectively.
ifOperStatus 1.3.6.1.2.1.2.2.1.8 read-only INTEGER	The current operational state of the interface.	The value depends on the contents of the ZSUBNET-STATE field of the ZTCI-DDL-STATUS-SUBNET structure, returned from a SPI STATUS command on SUBNET `ztsaTcpIpProcess`.* or LM_Get_Status_ on PIF: <u>ifOperStatus</u> — <u>ZSUBNET-STATE</u> 1 (up) — ZCOM-VAL-SUMSTATE-STARTED 2 (down) — ZCOM-VAL-SUMSTATE-STOPPED 3 (testing) — any other summary state State 3 (testing) indicates that although the interface is not in the stopped state, no operational packets can be passed.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 5 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifLastChange 1.3.6.1.2.1.2.2.1.9 read-only TimeTicks	The value of sysUpTime at the time the interface entered its current operational state.	If the current state existed prior to the last reinitialization of the SNMP agent, the value is 0. This value is updated whenever transitions in the ifOperStatus object are detected.
ifInOctets 1.3.6.1.2.1.2.2.1.10 read-only Counter	The total number of octets received on the interface, including framing characters.	LM_Get_Statistics_ on PIF name returns ifInOctets as a 64-bit entity for SLSA. This is truncated to 32 bits since ifInOctets is a 32 bit counter per RFC1213.
ifInUcastPkts 1.3.6.1.2.1.2.2.1.11 read-only Counter	The number of subnet-work-unicast packets delivered to a higher-layer protocol.	LM_Get_Statistics_ on PIF name returns ifInUcastkts for SLSA.
ifInNUcastPkts 1.3.6.1.2.1.2.2.1.11 read-only Counter	The number of non-unicast packets delivered to a higher-layer protocol.	LM_Get_Statistics_ on PIF name returns ifInNUcastPkts for SLSA.
ifInDiscards 1.3.6.1.2.1.2.2.1.13 read-only Counter	The number of inbound packets that were discarded even though no errors were detected to prevent their being deliverable to a higher-layer protocol.	LM_Get_Statistics on PIF name returns ifInDiscards for SLSA. ZAGGR-SDUS-NOT-DELIV field of the ZLAM-DDL-STATS-PORT structure returns ifInDiscards for TLAM. A SPI STATISTICS command on PORT #IP of the TLAM process returns ZAGGR-SDUS-NOT-DELIV value. For ifType 9, SPI STATISTICS command on PORT #SNAP of the TLAM process returns ZAGGR-SDUS-NOT-DELIV value.
ifInErrors 1.3.6.1.2.1.2.2.1.14 read-only Counter	The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.	LM_Get_Statistics_ on PIF name returns ifInErrors for SLSA. ZSTATS-SUBNET-IN-ERRS field of the ZTCI-DDL-STATS-SUBNET structure, returned from a SPI STATISTICS command on SUBNET `ztsaTcpIpProcess`.* returns ifInErrors for TLAM.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 6 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifInUnknownProtos 1.3.6.1.2.1.2.2.1.15 read-only Counter	The number of packets received via the interface that were discarded because of an unknown or unsupported protocol.	LM_Get_Statistics_ on PIF name returns ifInUnknownProtos for SLSA.
ifOutOctets 1.3.6.1.2.1.2.2.1.16 read-only Counter	The total number of octets transmitted out of the interface, including framing characters.	LM_Get_Statistics_ on PIF name returns ifOutOctets as a 64-bit entity for SLSA. This is truncated to 32 bits since ifOutOctets is a 32-bit counter per RFC 1213. ifOutOctets is not supported for the D-series TLAM interface; the constant 0 is returned.
ifOutUcastPkts 1.3.6.1.2.1.2.2.1.17 read-only Counter	The number of packets that a higher-level protocol requested to transmit to a subnetwork unicast address, including those that were discarded or not sent.	LM_Get_Statistics_ on PIF name returns ifOutUcastPkts for SLSA. ifOutUcastPkts is not supported for the D-series TLAM interface; the constant 0 is returned.
ifOutNUcastPkts 1.3.6.1.2.1.2.2.1.18 read-only Counter	The number of packets that a higher-level protocol requested to transmit to a non-unicast address, including those that were discarded or not sent.	LM_Get_Statistics_ on PIF name returns ifOutNUcastPkts for SLSA. ifOutNUcastPkts is not supported for the D-series TLAM interface; the constant 0 is returned.

Table 8-5. Interfaces Group Objects Supported by TCP/IP Subagent (page 7 of 7)

Object and Attributes	Definition	Format and Derivation of Value
ifOutDiscards 1.3.6.1.2.1.2.2.1.19 read-only Counter	The number of outbound packets that were discarded even though no errors had been detected to prevent their being transmitted.	LM_Get_Statistics_ on PIF name returns ifOutDiscards for SLSA. ifOutDiscards is not supported for the D-series TLAM interface; the constant 0 is returned.
ifOutErrors 1.3.6.1.2.1.2.2.1.20 read-only Counter	The number of outbound packets that could not be transmitted because of errors.	LM_Get_Statistics_ on PIF name returns ifOutErrors for SLSA. ZSTATS-SUBNET-OUT-ERRS field of the ZTCI-DDL-STATS-SUBNET structure, returned from a SPI STATISTICS command on SUBNET *ztsaTcpIpProcess*.* returns ifOutErrors for TLAM.
ifOutQLen 1.3.6.1.2.1.2.2.1.21 read-only Counter	The length of the output queue (in packets).	LM_Get_Statistics_ on PIF name returns ifOutQLen for SLSA. ifOutQLen is not supported for the D-series TLAM interface; the constant 0 is returned.

RFC Compliance

Table 8-6 summarizes compliance of Interfaces group support with RFC 1213.

Table 8-6. Compliance With Interfaces Group Definitions in RFC 1213

Object Descriptor	Compliance	Explanation
ifNumber	Yes	See Table 8-5.
ifIndex	Yes	See Table 8-5.
ifDescr	Yes	See Table 8-5.
ifType	Partial	See Table 8-5.
ifMtu	Partial	See Table 8-5.
ifSpeed	Yes	See Table 8-5.
ifPhysAddress	Partial	See Table 8-5.
ifAdminStatus	Partial	Constant value of up (1).
ifOperStatus	Yes	See Table 8-5.
ifLastChange	Yes	See Table 8-5.
ifInOctets	Partial	See Table 8-5.
ifInUcastPkts	Partial	See Table 8-5.
ifInNUcastPkts	Partial	See Table 8-5.
ifInDiscards	Partial	See Table 8-5.
ifInErrors	Yes	See Table 8-5.
ifInUnknownProtos	Partial	See Table 8-5.
ifOutOctets	Partial	See Table 8-5.
ifOutUcastPkts	Partial	See Table 8-5.
ifOutNUcastPkts	Partial	See Table 8-5.
ifOutDiscards	Partial	See Table 8-5.
ifOutErrors	Yes	See Table 8-5.
ifOutQLen	Partial	See Table 8-5.
ifOutSpecific	Partial	No corresponding HP instrumentation exists; 0:0 is returned.

ifTable Maintenance

Ethernet, SNAP (Subnetwork Access Point), and X.25 interfaces are configured by the system administrator through the Subsystem Control Facility (SCF) or programmatically using SPI. The TCP/IP Subagent uses this configuration information to derive ifTable entries.

For each subnet found in response to an INFO SPI command for the SUBNET object, the TCP/IP Subagent extracts the subnet type and the associated I/O process name. Using this information, the TCP/IP Subagent sends the necessary INFO, STATUS, and STATISTICS SPI commands for the PORT object.

It then maps the values found in the following structures into the appropriate MIB objects:

ZTCI-DDL-INFO-SUBNET
ZTCI-DDL-STATS-SUBNET
ZLAM-DDL-STATUS-PORT
ZLAM-DDL-STATS-PORT
ZLAM-DDL-INFO-PORT

If the underlying interface of the subnet is SLSA, the TCP/IP Subagent maps the values from the following structures:

LMPIFAttr
LMPIFStatus
LMPIFStatistics

If the TCP/IP Subagent finds a change in the subnet configuration while building the ifTable, it also updates the indexes in the ipAddrTable, ipRouteTable, and ipNetToMediaTable in the IP group.

The TCP/IP Subagent assigns a number to each SUBNET object defined in the HP TCP/IP subsystem it is managing. That number represents both the index of the ifTable and the instance of the MIB object. This number is assigned when an interface is discovered and never changes while the TCP/IP Subagent executes. However, backup takeovers can change the assignment of the index value.

When the TCP/IP Subagent finds that a SUBNET object has been deleted or added, it marks it “deleted” or adds a corresponding ifTable entry. If a SUBNET object is deleted and then readded with the same attributes, the TCP/IP Subagent removes the “deleted” mark and reactivates the entry.

IP Group

The IP group contains information about the internet protocol (IP) layer of the TCP/IP subsystem being managed.

The IP group consists of a collection of scalar objects and three tables: ipAddrTable, ipRouteTable, and ipNetToMediaTable. Objects with a check mark in the following list are objects for which there is corresponding HP instrumentation:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              ip (4) √
                ipForwarding (1)  √
                ipDefaultTTL (2)
                ipInReceives (3) √
                ipInHdrErrors (4) √
                ipInAddrErrors (5)
                ipForwDatagrams (6) √
                ipInUnknownProtos (7)
                ipInDiscards (8) √
                ipInDelivers (9) √
                ipOutRequests (10) √
                ipOutDiscards (11)
                ipOutNoRoutes (12) √
                ipReasmTimeout (13)
                ipReasmReqds (14) √
                ipReasmOKs (15) √
                ipReasmFails (16) √
                ipFragOKs (17)
                ipFragFails (18)
                ipFragCreates (19)
                ipAddrTable (20) √
                  ipAddrEntry (1) √
                    ipAdEntAddr (1) √
                    ipAdEntIfIndex (2) √
                    ipAdEntNetMask (3) √
                    ipAdEntBcastAddr (4) √
                    ipAdEntReasmMaxSize (5)
```

```
ipRouteTable (21) √
   ipRouteEntry (1) √
      ipRouteDest (1) √
      ipRouteIfIndex (2) √
      ipRouteMetric1 (3) √
      ipRouteMetric2 (4) √
      ipRouteMetric3 (5) √
      ipRouteMetric4 (6) √
      ipRouteNextHop (7) √
      ipRouteType (8) √
      ipRouteProto (9) √
      ipRouteAge (10)
      ipRouteMask (11) √
      ipRouteMetric5 (12) √
      ipRouteInfo (13) √
ipNetToMediaTable (22) √
   ipNetToMediaEntry (1) √
      ipNetToMediaIfIndex (1) √
      ipNetToMediaPhysAddress (2) √
      ipNetToMediaNetAddress (3) √
      ipNetToMediaType (4) √
ipRoutingDiscards (23)
```

MIB Objects

Table 8-7 describes how the TCP/IP Subagent supports objects in the MIB-II IP group that have HP instrumentation.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 1 of 8)

Object and Attributes	**Definition**	**Format and Derivation of Value**
ipForwarding 1.3.6.1.2.1.4.1 read-only INTEGER	An indication of whether datagrams addressed to other entities are being forwarded by the managed TCP/IP subsystem.	The constant value is 1 (forwarding) because the HP TCP/IP subsystem always attempts to forward datagrams to other destinations.
ipInReceives 1.3.6.1.2.1.4.3 read-only Counter	The total number of input datagrams received, including those received in error.	The value (integer) of the ZSTATS-IP-TOTAL field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 2 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipInHdrErrors 1.3.6.1.2.1.4.4 read-only Counter	The number of input datagrams discarded because of errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, and errors discovered in processing their IP options.	The sum (integer) of the values in the following fields of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*: ZSTATS-IP-BAD-XSUM ZSTATS-IP-TOO-SHORT ZSTATS-IP-TOO-SMALL ZSTATS-IP-BAD-HDR-LEN ZSTATS-IP-BAD-LEN
ipForwDatagrams 1.3.6.1.2.1.4.6 read-only Counter	The number of input datagrams that required forwarding.	The value (integer) of the ZSTATS-IP-FWD field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
ipInDiscards 1.3.6.1.2.1.4.8 read-only Counter	The number of input datagrams that were not erroneous but had to be discarded (because of a lack of buffer space, for example). This counter does not include any datagrams discarded while awaiting reassembly.	The value (integer) of the ZSTATS-IP-FRAG-DROP field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
ipInDelivers 1.3.6.1.2.1.4.9 read-only Counter	The total number of input datagrams successfully delivered to IP user protocols.	The difference (integer) between the value in the ZSTATS-IP-TOTAL field and the sum of the values in the following fields of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*: ZSTATS-IP-BAD-XSUM ZSTATS-IP-TOO-SHORT ZSTATS-IP-TOO-SMALL ZSTATS-IP-BAD-HDR-LEN ZSTATS-IP-BAD-LEN ZSTATS-IP-FRAG-TIMEOUT ZSTATS-IP-FRAG-DROP ZSTATS-IP-CANT-FWD

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 3 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipOutRequests 1.3.6.1.2.1.4.10 read-only Counter	The total number of IP datagrams that local IP user protocols supplied to IP in requests for transmission. This counter does not include any datagrams counted in ipForwDatagrams.	The value (integer) of the ZSTATS-IP-OUT-PKTS field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*.
ipOutNoRoutes 1.3.6.1.2.1.4.12 read-only Counter	The number of IP datagrams discarded because no route could be found to transmit them to their destination. This counter includes any packets counted in ipForwDatagrams which meet the “no-route” criterion, including any datagrams that a host cannot route because all of its default gateways are down.	The value (integer) of the ZSTATS-IP-CANT-FWD field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*.
ipReasmReqds 1.3.6.1.2.1.4.14 read-only Counter	The number of IP fragments received that needed to be reassembled.	The value (integer) of the ZSTATS-IP-FRAG field of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*.
ipReasmOKs 1.3.6.1.2.1.4.15 read-only Counter	The number of IP datagrams successfully reassembled.	The difference (integer) between the value in the ZSTATS-IP-FRAG field and the sum of the values in the following fields of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*: ZSTATS-IP-FRAG-DROP ZSTATS-IP-FRAG-TIMEOUT

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 4 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipReasmFails 1.3.6.1.2.1.4.16 read-only Counter	The number of failures detected by the IP reassembly algorithm.	The sum (integer) of the values in the following fields of the ZTCI-DDL-IP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*: ZSTATS-IP-FRAG-DROP ZSTATS-IP-FRAG-TIMEOUT
ipAddrTable Objects:	Entries describing IP addressing.	Refer to ipAddrTable Maintenance on page 8-50 for information on how entries are maintained.
ipAdEntAddr 1.3.6.1.2.1.4.20.1.1 read-only IpAddress	The IP address to which this entry's addressing information pertains.	The IP address contained in the ZIP-ADDR field of the ZTCI-DDL-INFO-SUBNET structure, returned from a SPI INFO command on SUBNET *`ztsaTcpIpProcess`*.*.
ipAdEntIfIndex 1.3.6.1.2.1.4.20.1.2 read-only INTEGER	The value uniquely identifying the interface to which this entry is applicable. Each interface is also identified by the same value of ifIndex in the Interfaces group.	An integer value computed by the TCP/IP Subagent.
ipAdEntNetMask 1.3.6.1.2.1.4.20.1.3 read-only IpAddress	The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the host bits set to 0.	The IP address contained in the ZSUBNET-MASK field of the ZTCI-DDL-INFO-SUBNET structure, returned from a SPI INFO command on SUBNET *`ztsaTcpIpProcess`*.*.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 5 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipAdEntBcastAddr 1.3.6.1.2.1.4.20.1.4 read-only INTEGER	The value of the least significant bit in the IP broadcast address used for sending datagrams on the logical interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value is 1. This value applies to both the subnet and network broadcast addresses used by the entity on this (logical) interface.	The constant value for this object is 1, as specified by RFC 1213.
ipRouteTable Objects:	An entry for each known route.	Refer to ipRouteTable Maintenance on page 8-50 for information on how entries are maintained.
ipRouteDest 1.3.6.1.2.1.4.21.1.1 read-only IpAddress	The destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple routes to a single destination can appear in the table.	The IP address contained in the ZDEST-IP-ADDR field of the ZTCI-DDL-INFO-ROUTE structure, returned from a SPI INFO command on SUBNET *ztsaTcpIpProcess*.*.
ipRouteIfIndex 1.3.6.1.2.1.4.21.1.2 read-only INTEGER	The value uniquely identifying the local interface through which the next hop of this route should be reached. The interface identified by a particular value of this index is the same interface identified by the same ifIndex value.	An integer value computed by TCP/IP Subagent. If no ipRouteNextHop matches the ipAdEntAddr entry, a zero is returned.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 6 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipRouteMetric1 1.3.6.1.2.1.4.21.1.3 ipRouteMetric2 1.3.6.1.2.1.4.21.1.4 ipRouteMetric3 1.3.6.1.2.1.4.21.1.5 ipRouteMetric4 1.3.6.1.2.1.4.21.1.6 ipRouteMetric5 1.3.6.1.2.1.4.21.1.12 read-only INTEGER	Primary and alternate routing metrics for this route. The semantics of these metrics are determined by the routing protocol specified for the route's ipRouteProto value. If these metrics are not used, their values are set to -1.	The constant value for these objects is -1 (not used).
ipRouteNextHop 1.3.6.1.2.1.4.21.1.7 read-only IpAddress	The IP address of the next hop of this route. In the case of a route bound to an interface that is realized via a broadcast media, the value of this field is the agent's IP address on that interface.	The IP address contained in the ZGW-IP-ADDR field of the ZTCI-DDL-INFO-ROUTE structure, returned from SPI INFO command on ROUTE `ztsaTcpIpProcess`.*.
ipRouteType 1.3.6.1.2.1.4.21.1.8 read-only INTEGER	The type of route.	The value depends on the contents of the ZIF-GW field of the ZTCI-DDL-INFO-ROUTE structure, returned from a SPI INFO command on ROUTE `ztsaTcpIpProcess`.*, as follows: <u>ipRouteType</u> — <u>ZIF-GW</u> 3 = direct; the route is directly connected to a [sub]network — ZSPI-VAL-FALSE 4 = indirect; the route is connected to a nonlocal [sub]network — ZSPI-VAL-TRUE If a route ceases to exist, its entry is removed from the table. Therefore, this implementation never returns a value of 2 (invalid) to alert the SNMP manager that a route has been deleted.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 7 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipRouteProto 1.3.6.1.2.1.4.21.1.9 read-only INTEGER	The routing mechanism by which this route was learned.	The constant value for this object is 2 (configured through local network management).
ipRouteMask 1.3.6.1.2.1.4.21.1.11 read-only IpAddress	The mask to be logically ANDed with the destination address before being compared to the value of the ipRouteDest object If the value of the ipRouteDest is 0.0.0.0 (a default route), then the mask value is also 0.0.0.0 (all IP routing subsystems implicitly use this mechanism).	The mask associated with a route is retrieved by first obtaining the name of the subnet from the ZSUBNET field in the ZTCI-DDL-INFO-ROUTE structure. Using the subnet object name, an INFO SPI command is sent to the SUBNET object. The mask value is contained in the ZSUBNET-ZMASK field of the ZTCI-DDL-INFO-SUBNET structure. Because setting individual route masks is not supported on the SPI interface, the associated subnet mask is used to add new routes.
ipRouteInfo 1.3.6.1.2.1.4.21.1.13 read-only OBJECT IDENTIFIER	A reference to MIB definitions specific to the particular routing protocol responsible for this route, as determined by the value specified for the route's ipRouteProto object.	The constant value of this object is {0.0}, meaning that this information is not present, as specified by RFC 1213.
ipNetToMediaTable Objects:	Entries used for mapping from IP addresses to physical addresses.	Refer to ipNetToMediaTable Maintenance on page 8-51 for information on how entries are maintained.
ipNetToMediaIfIndex 1.3.6.1.2.1.4.22.1.1 read-only INTEGER	The interface for which this entry applies. The interface identified by a specific value of this index is the interface identified by the same ifIndex value.	The ipAddrTable.ipAdEntIfIndex is copied into the ipNetToMediaTable.ipNetToMediaIfIndex.

Table 8-7. IP Group Objects Supported by TCP/IP Subagent (page 8 of 8)

Object and Attributes	Definition	Format and Derivation of Value
ipNetToMediaPhysAddress 1.3.6.1.2.1.4.22.1.2 read-only PhysAddress	The media-dependent physical address.	The value of this object is the same as the value of ifPhysAddress for the corresponding ifIndex.
ipNetToMediaNetAddress 1.3.6.1.2.1.4.22.1.3 read-only IpAddress	The IP address corresponding to the media-dependent physical address.	The IP address contained in the ZIP-ADDR field of the ZTCI-DDL-INFO-SUBNET structure, returned from a SPI INFO command on SUBNET *`ztsaTcpIpProcess`*.*. The ipAddrTable.ipAdEntAddr entries are copied into the ipNetToMediaTable.ipNetToMediaNetAddress.
ipNetToMediaType 1.3.6.1.2.1.4.22.1.4 read-only INTEGER	The method used to map physical to IP addresses.	This object has the constant value of 4 (static).

RFC Compliance

Table 8-8 summarizes compliance of IP group support with RFC 1213.

Table 8-8. Compliance With IP Group Definitions in RFC 1213 (page 1 of 3)

Object Descriptor	Compliance	Explanation
ipForwarding	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipDefaultTTL	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipInReceives	Yes	See Table 8-7.
ipInHdrErrors	Yes	See Table 8-7.
ipInAddrErrors	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipForwDatagrams	Yes	See Table 8-7.
ipInUnknownProtos	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipInDiscards	Yes	See Table 8-7.
ipInDelivers	Yes	See Table 8-7.
ipOutRequests	Yes	See Table 8-7.

Table 8-8. Compliance With IP Group Definitions in RFC 1213 (page 2 of 3)

Object Descriptor	Complianc e	Explanation
ipOutDiscards	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipOutNoRoutes	Yes	See Table 8-7.
ipReasmTimeout	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipReasmReqds	Yes	See Table 8-7.
ipReasmOKs	Yes	See Table 8-7.
ipReasmFails	Yes	See Table 8-7.
ipFragOKs	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipFragFails	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipFragCreates	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipAdEntAddr	Yes	See Table 8-7.
ipAdEntIfIndex	Yes	See Table 8-7.
ipAdEntNetMask	Yes	See Table 8-7.
ipAdEntBcaseAddr	Yes	See Table 8-7.
ipAdEntReasmMaxSize	Partial	No corresponding HP instrumentation exists; 0 is returned.
ipRouteDest	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRouteIfIndex	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7- for description of retrievable information.
ipRouteMetric1 through ipRouteMetric5	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRouteNextHop	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRouteType	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRouteProto	Yes	See Table 8-7.
ipRouteAge	Partial	No corresponding HP instrumentation exists; 0 is returned.

Table 8-8. Compliance With IP Group Definitions in RFC 1213 (page 3 of 3)

Object Descriptor	Complianc e	Explanation
ipRouteMask	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRouteInfo	Yes	See Table 8-7.
ipNetToMediaIfIndex	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipNetToMediaPhysAddress	Yes	This object does not comply with RFC 1213 standards.
ipNetToMediaNetAddress	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipNetToMediaType	Yes	Set operation not supported; attempt returns noSuchName error. See Table 8-7 for description of retrievable information.
ipRoutingDiscards	Partial	No corresponding HP instrumentation exists; 0 is returned.

ipAddrTable Maintenance

Subnets are configured interactively through SCF or programmatically through SPI. The TCP/IP Subagent uses this configuration information to derive table entries.

The TCP/IP Subagent builds entries by sending an INFO SPI command for the SUBNET object. It then maps the values found in the ZTCI-DDL-INFO-SUBNET structure into the appropriate MIB objects.

If the TCP/IP Subagent finds a change in the subnet configuration while building the ipAddrTable, it also updates the indexes in the ifTable and the ipNetToMediaTable.

When the TCP/IP Subagent finds that a SUBNET object has been deleted or added, it removes or adds the corresponding ifAddrTable entry.

When you configure subnets in Shared failover mode, only the shared IPaddress appears in displays. The reserved IP address will not appear,

ipRouteTable Maintenance

Route objects are configured interactively through SCF or programmatically through SPI. The TCP/IP Subagent uses this configuration information to derive table entries.

The TCP/IP Subagent builds entries by sending an INFO SPI command to the ROUTE object. It then maps the values found in the ZTCI-DDL-INFO-ROUTE token into the appropriate MIB objects.

When the TCP/IP Subagent finds that a ROUTE object has been deleted or added, it removes or adds the corresponding ipRouteTable entry.

ipNetToMediaTable Maintenance

The TCP/IP Subagent builds entries by sending an INFO SPI command for the SUBNET object. For each subnet found, the TCP/IP Subagent extracts the subnet type and the associated I/O process name.

Then, using the I/O process name, the TCP/IP Subagent sends the necessary INFO SPI commands for the PORT object. The values found in the ZTCI-DDL-INFO-SUBNET and the ZLAM-DDL-INFO-PORT and lmPifAttr structures is mapped into the appropriate MIB objects.

If the TCP/IP Subagent finds a change in the subnet configuration while building the ipNetToMediaTable, it also updates the indexes in the ifTable and the ipAddrTable.

When the TCP/IP Subagent finds that a SUBNET object has been deleted or added, it removes or adds the corresponding ipNetToMediaTable entry.

ICMP Group

The ICMP group contains information about the Internet Control Message Protocol (ICMP) layer of the TCP/IP subsystem being managed.

The ICMP group consists of a collection of scalar objects. The objects identified by a check mark in the following list are objects for which there is corresponding HP TCP/IP subsystem instrumentation:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              icmp (5) √
                icmpInMsgs (1)  √
                icmpInErrors (2) √
                icmpInDestUnreachs (3) √
                icmpInTimeExcds (4) √
                icmpInParmProbs (5) √
                icmpInSrcQuenchs (6) √
                icmpInRedirects (7) √
                icmpInEchos (8) √
                icmpInEchoReps (9) √
                icmpInTimestamps (10) √
                icmpInTimestampReps (11) √
                icmpAddrMasks (12)
                icmpInAddrMaskReps (13)
```

icmpOutMsgs (14) √
icmpOutErrors (15) √
icmpOutDestUnreachs (16) √
icmpOutTimeExcds (17) √
icmpOutParmProbs (18) √
icmpOutSrcQuenchs (19) √
icmpOutRedirects (20) √
icmpOutEchos (21) √
icmpOutEchoReps (22) √
icmpOutTimestamps (23) √
icmpOutTimestampReps (24) √
icmpOutAddrMasks (25)
icmpOutAddrMaskReps (26)

MIB Objects

Table 8-9 describes how the TCP/IP Subagent supports objects in the MIB-II ICMP group that have HP instrumentation.

Table 8-9. ICMP Group Objects Supported by TCP/IP Subagent (page 1 of 4)

Object and Attributes	Definition	Derivation of Value
icmpInMsgs 1.3.6.1.2.1.5.1 read-only Counter	The total number of ICMP messages received, including those counted by icmpErrors.	The sum of all ICMP message counters in the ZSTATS-ICMP-IN-HIST array plus the sum of the values in the following fields of the ZTCI-DDL-ICMP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*: ZSTATS-ICMP-BADCODE ZSTATS-ICMP-TOO-SHORT ZSTATS-ICMP-XSUM ZSTATS-ICMP-BAD-LEN
icmpInErrors 1.3.6.1.2.1.5.2 read-only Counter	The number of ICMP messages received but determined to have ICMP-specific errors. (such as bad ICMP checksums or bad length).	The sum of the values in the following fields of the ZTCI-DDL-ICMP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*: ZSTATS-ICMP-BADCODE ZSTATS-ICMP-TOO-SHORT ZSTATS-ICMP-XSUM ZSTATS-ICMP-BAD-LEN
icmpInDestUnreachs 1.3.6.1.2.1.5.3 read-only Counter	The number of ICMP destination-unreachable messages received.	Offset 5 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.

Table 8-9. ICMP Group Objects Supported by TCP/IP Subagent (page 2 of 4)

Object and Attributes	Definition	Derivation of Value
icmpInTimeExcds 1.3.6.1.2.1.5.4 read-only Counter	The number of ICMP time-exceeded messages received.	Offset 13 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInParmProbs 1.3.6.1.2.1.5.5 read-only Counter	The number of ICMP parameter-problem messages received.	Offset 15 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInSrcQuenchs 1.3.6.1.2.1.5.6 read-only Counter	The number of ICMP source-quench messages received.	Offset 6 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInRedirects 1.3.6.1.2.1.5.7 read-only Counter	The number of ICMP redirect messages received.	Offset 7 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInEchos 1.3.6.1.2.1.5.8 read-only Counter	The number of ICMP echo-request messages received.	Offset 10 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInEchoReps 1.3.6.1.2.1.5.9 read-only Counter	The number of ICMP echo-reply messages received.	Offset 2 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInTimestamps 1.3.6.1.2.1.5.10 read-only Counter	The number of ICMP timestamp-request messages received.	Offset 15 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpInTimestampReps 1.3.6.1.2.1.5.11 read-only Counter	The number of ICMP timestamp-reply messages received.	Offset 16 in the ZSTATS-ICMP-IN-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutMsgs 1.3.6.1.2.1.5.14 read-only Counter	The total number of ICMP messages that were sent, including all those counted by icmpoutErrors.	The sum of all ICMP message counters in the ZSTATS-ICMP-OUT-HIST array, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*, plus the value of the icmpOutErrors object.

Table 8-9. ICMP Group Objects Supported by TCP/IP Subagent (page 3 of 4)

Object and Attributes	Definition	Derivation of Value
icmpOutErrors 1.3.6.1.2.1.5.15 read-only Counter	The number of ICMP messages not sent because of problems discovered within ICMP (such as lack of buffers). This value does not include errors discovered outside of the ICMP layer such as the inability of IP to route the resultant datagram.	The difference between the value in the ZSTATS-ICMP-ERR field of the ZTCI-DDL-ICMPSTATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*, and the value of the icmpInErrors object.
icmpOutDestUnreachs 1.3.6.1.2.1.5.16 read-only Counter	The number of ICMP destination-unreachable messages sent.	Offset 3 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutTimeExcds 1.3.6.1.2.1.5.17 read-only Counter	The number of ICMP time-exceeded messages sent.	Offset 11 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutParmProbs 1.3.6.1.2.1.5.18 read-only Counter	The number of ICMP parameter-problem messages sent.	Offset 12 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutSrcQuenchs 1.3.6.1.2.1.5.19 read-only Counter	The number of ICMP source-quench messages sent.	Offset 4 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutRedirects 1.3.6.1.2.1.5.20 read-only Counter	The number of ICMP redirect messages sent.	Offset 5 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutEchos 1.3.6.1.2.1.5.21 read-only Counter	The number of ICMP echo-request messages sent.	Offset 8 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.

Table 8-9. ICMP Group Objects Supported by TCP/IP Subagent (page 4 of 4)

Object and Attributes	Definition	Derivation of Value
icmpOutEchoReps 1.3.6.1.2.1.5.22 read-only Counter	The number of ICMP echo-reply messages sent.	Offset 0 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutTimestamps 1.3.6.1.2.1.5.23 read-only Counter	The number of ICMP timestamp-request messages sent.	Offset 13 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.
icmpOutTimestampReps 1.3.6.1.2.1.5.24 read-only Counter	The number of ICMP timestamp-reply messages sent.	Offset 14 in the ZSTATS-ICMP-OUT-HIST array in the ZTCI-DDL-ICMP-STATS structure.

RFC Compliance

Table 8-10 summarizes compliance of ICMP group support with RFC 1213.

Table 8-10. Compliance With ICMP Group Definitions in RFC 1213 (page 1 of 2)

Object Descriptor	Compliance	Explanation
icmpInMsgs	Yes	See Table 8-9.
icmpInErrors	Yes	See Table 8-9.
icmpInDestUnreachs	Yes	See Table 8-9.
icmpInTimeExcds	Yes	See Table 8-9.
icmpInParmProbs	Yes	See Table 8-9.
icmpInSrcQuenchs	Yes	See Table 8-9.
icmpInRedirects	Yes	See Table 8-9.
icmpInEchos	Yes	See Table 8-9.
icmpInEchoReps	Yes	See Table 8-9.
icmpInTimestamps	Yes	See Table 8-9.
icmpInTimestampReps	Yes	See Table 8-9.
icmpAddrMasks	Partial	No corresponding HP instrumentation exists; 0 is returned.
icmpInAddrMaskReps	Partial	No corresponding HP instrumentation exists; 0 is returned.
icmpOutMsgs	Yes	See Table 8-9.
icmpOutErrors	Yes	See Table 8-9.
icmpOutDestUnreachs	Yes	See Table 8-9.

Table 8-10. Compliance With ICMP Group Definitions in RFC 1213 (page 2 of 2)

Object Descriptor	Compliance	Explanation
icmpOutTimeExcds	Yes	See Table 8-9.
icmpOutParmProbs	Yes	See Table 8-9.
icmpOutSrcQuenchs	Yes	See Table 8-9.
icmpOutRedirects	Yes	See Table 8-9.
icmpOutEchos	Yes	See Table 8-9.
icmpOutEchoReps	Yes	See Table 8-9.
icmpOutTimestamps	Yes	See Table 8-9.
icmpOutTimestampReps	Yes	See Table 8-9.
icmpOutAddrMasks	Partial	No corresponding HP instrumentation exists; 0 is returned.
icmpOutAddrMaskReps	Partial	No corresponding HP instrumentation exists; 0 is returned.

TCP Group

The TCP group contains information about the Transmission Control Protocol (TCP) layer of the TCP/IP subsystem being managed.

The TCP group consists of a collection of scalar objects and one table (tcpConnTable). The objects identified by a check mark in the following list are objects for which there is corresponding HP TCP/IP subsystem instrumentation:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              tcp (6) √
                tcpRtoAlgorithm (1)  √
                tcpRtoMin (2) √
                tcpRtoMax (3) √
                tcpMaxConn (4) √
                tcpActiveOpens (5) √
                tcpPassiveOpens (6) √
                tcpAttemptFails (7)
                tcpEstabResets (8)
                tcpCurrEstab (9) √
                tcpInSegs (10) √
                tcpOutSegs (11) √
```

```
tcpRetransSegs (12) √
tcpConnTable (13) √
   tcpConnEntry (1) √
      tcpConnState (1) √
      tcpConnLocalAddress (2) √
      tcpConnLocalPort (3) √
      tcpConnRemAddress (4) √
      tcpConnRemPort (5) √
tcpInErrs (14) √
tcpOutRsts (15)
```

MIB Objects

Table 8-11 describes how the TCP/IP Subagent supports objects in the MIB-II TCP group that have HP instrumentation.

Table 8-11. TCP Group Objects Supported by TCP/IP Subagent (page 1 of 4)

Object and Attributes	Definition	Format and Derivation of Value
tcpRtoAlgorithm 1.3.6.1.2.1.6.1 read-only INTEGER	The algorithm used to determine the timeout value used for retransmitting unacknowledged octets.	The constant value for this object is 3 (MIL-STD 1178).
tcpRtoMin 1.3.6.1.2.1.6.2 read-only INTEGER	The minimum value permitted for a retransmission timeout.	The constant value for this object is 1000 (milliseconds).
tcpRtoMax 1.3.6.1.2.1.6.3 read-only INTEGER	The maximum value permitted for a retransmission timeout.	The constant value for this object is 30000 (milliseconds).
tcpMaxConn 1.3.6.1.2.1.6.4 read-only INTEGER	The limit on the total number of TCP connections that can be supported.	The constant value for this object is -1 (maximum number of connections is dynamic).
tcpActiveOpens 1.3.6.1.2.1.6.5 read-only Counter	The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.	ZSTATS-TCP-OUT-CONN field of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
tcpPassiveOpens 1.3.6.1.2.1.6.6 read-only Counter	The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.	ZSTATS-TCP-IN-CONN field of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.

Table 8-11. TCP Group Objects Supported by TCP/IP Subagent (page 2 of 4)

Object and Attributes	Definition	Format and Derivation of Value
tcpCurrEstab 1.3.6.1.2.1.6.9 read-only Gauge	The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.	The count of the connections having the following values: ZTCI-VAL-TCP-STATE-ESTAB ZTCI-VAL-TCP-STATE-CLOSE-WAIT in the ZNET-STATUS-STATE field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*.
tcpInSegs 1.3.6.1.2.1.6.10 read-only Counter	The total number of segments received, including those received in error. This count includes segments received on currently established connections.	ZSTATS-TCP-IN-PKTS field of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
tcpOutSegs 1.3.6.1.2.1.6.11 read-only Counter	The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets.	ZSTATS-TCP-OUT-PKTS field of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
tcpRetransSegs 1.3.6.1.2.1.6.12 read-only Counter	The number of TCP segments transmitted containing one or more previously transmitted octets.	ZSTATS-TCP-REXMIT field of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*.
tcpConnTable Objects:	Contains entries describing the TCP connection.	Refer to tcpConnTable Maintenance on page 8-61 for information on how entries are created and updated.

Table 8-11. TCP Group Objects Supported by TCP/IP Subagent (page 3 of 4)

Object and Attributes	Definition	Format and Derivation of Value
tcpConnState 1.3.6.1.2.1.6.13.1.1 read-only INTEGER	The state of the TCP connection.	One of the following values, derived from the indicated SPI value in the ZNET-STATUS-STATE field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsatcpIpProcess*: <u>tcpConnState</u> <u>VAL-TCP-STATE</u> 1 (listen) LISTEN 2 (synSent) SYNC-SENT 3 (synReceived) SYNC-RECV 4 (established) ESTAB 5 (finWait1) FIN-WAIT-1 6 (finWait2) FIN-WAIT-2 7 (closeWait) CLOSE-WAIT 8 (lastAck) LAST-ACK 9 (closing) CLOSING 10 (timeWait) TIME-WAIT
tcpConnLocalAddress 1.3.6.1.2.1.6.13.1.2 read-only IpAddress	The local IP address for the TCP connection.	ZNET-STATUS-LOC-ADDR field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*. For a connection in the LISTEN state that is willing to accept connections for any IP interface associated with the node, the value is 0.0.0.0.
tcpConnLocalPort 1.3.6.1.2.1.6.13.1.3 read-only INTEGER	The local port number for the TCP connection.	ZNET-STATUS-LOC-PORT field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*.

Table 8-11. TCP Group Objects Supported by TCP/IP Subagent (page 4 of 4)

Object and Attributes	Definition	Format and Derivation of Value
tcpConnRemAddress 1.3.6.1.2.1.6.13.1.4 read-only IpAddress	The remote IP address for the TCP connection.	The IP address in the ZNET-STATUS-FORGN-ADDR field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*.
tcpConnRemPort 1.3.6.1.2.1.6.13.1.5 read-only INTEGER	The remote port number for the TCP connection.	ZNET-STATUS-FORGN-PORT field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*.
tcpInErrs 1.3.6.1.2.1.6.14 read-only Counter	The total number of segments received in error.	The sum of the values in the following fields of the ZTCI-DDL-TCP-STATS structure, returned from a SPI STATISTICS command on PROCESS *ztsaTcpIpProcess*: ZSTATS-TCP-BAD-OFF ZSTATS-TCP-BAD-SEGS ZSTATS-TCP-BAD-XSUM ZSTATS-TCP-BAD-HDR-LEN

RFC Compliance

Table 8-12 summarizes compliance of TCP group support with RFC 1213.

Table 8-12. Compliance With TCP Group Definitions in RFC 1213 (page 1 of 2)

Object Descriptor	Complianc e	Explanation
tcpRtoAlgorithm	Yes.	See Table 8-11.
tcpRtoMin	Yes.	See Table 8-11.
tcpRtoMax	Yes	See Table 8-11.
tcpActiveOpens	Yes	See Table 8-11.
tcpPassiveOpens	Yes	See Table 8-11.
tcpAttemptFails	Partial	No corresponding HP instrumentation exists; 0 is returned.
tcpEstabResets	Partial	No corresponding HP instrumentation exists; 0 is returned.
tcpCurrEstab	Yes	See Table 8-11.
tcpInSegs	Yes	See Table 8-11.
tcpOutSegs	Yes	See Table 8-11.
tcpRetransSegs	Yes	See Table 8-11.

Table 8-12. Compliance With TCP Group Definitions in RFC 1213

Object Descriptor	Complianc e	Explanation
tcpConnState	Partial	See Table 8-11 for description of retrievable information. The MIB-II value 12 (delete TCB) is not supported in this implementation; attempts to set this object to any value return a badValue error.
tcpConnLocalAddress	Yes	See Table 8-11.
tcpConnLocalPort	Yes	See Table 8-11.
tcpConnRemAddress	Yes	See Table 8-11.
tcpConnRemPort	Yes	See Table 8-11.
tcpInErrs	Yes	See Table 8-11.
tcpOutRsts	Partial	No corresponding HP instrumentation exists; 0 is returned.

tcpConnTable Maintenance

The TCP/IP Subagent builds entries by sending a STATUS SPI command to the PROCESS object, using the name of the TCP/IP subsystem being managed (*ztsaTcpIpProcess*). It then maps the values found in the ZTCI-DDL-STATUS-PROTO structure into the appropriate MIB objects. The ZTCI-DDL-STATUS-PROTO structure returns connection information for both TCP and UDP. The subagent ensures that only the TCP connections are selected. Instances of object types that represent information about a particular TCP connection are transient. They persist only as long as the connection in question.

Rebuilding the tcpConnTable also causes the indexes in the udpTable to be updated.

UDP Group

The UDP group contains information about the User Datagram Protocol (UDP).

The UDP group consists of a collection of scalar objects and one table (udpTable). The objects identified by a check mark in the following list are objects for which there is corresponding HP TCP/IP subsystem instrumentation:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              udp (7) √
                udpInDatagrams (1)  √
                udpNoPorts (2)
                udpInErrors (3) √
```

```
udpOutDatagrams (4) √
udpTable (5) √
   udpEntry(1) √
      udpLocalAddress (1) √
      udpLocalPort (2) √
```

MIB Objects

Table 8-13 describes how the TCP/IP Subagent supports objects in the MIB-II UDP group that have HP instrumentation.

Table 8-13. UDP Group Objects Supported by TCP/IP Subagent (page 1 of 2)

Object and Attributes	Definition	Format and Derivation of Value
udpInDatagrams 1.3.6.1.2.1.7.1 read-only Counter	The total number of UDP datagrams delivered to UDP users.	ZSTATS-UDP-IN-PKTS field of the ZTCI-DDL-UDP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*.
udpInErrors 1.3.6.1.2.1.7.3 read-only Counter	The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.	The sum of the values in the following fields of the ZTCI-DDL-UDP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*: ZSTATS-UDP-BAD-LEN ZSTATS-UDP-BAD-XSUM ZSTATS-UDP-BAD-HDR-LEN
udpOutDatagrams 1.3.6.1.2.1.7.4 read-only Counter	The total number of datagrams sent.	ZSTATS-UDP-OUT-PKTS field of the ZTCI-DDL-UDP-STATS structure, returned from a SPI STATISTICS command on PROCESS *`ztsaTcpIpProcess`*.

Table 8-13. UDP Group Objects Supported by TCP/IP Subagent (page 2 of 2)

Object and Attributes	Definition	Format and Derivation of Value
udpTable Objects:	Contains entries describing UDP listener information.	Refer to udpTable Maintenance on page 8-63 for information on how entries are created and updated.
udpLocalAddress 1.3.6.1.2.1.7.5.1.1 read-only IpAddress	The local IP address for the UDP listener.	The IP address contained in the ZNET-STATUS-LOC-ADDR field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*. In the case of a UDP listener that is willing to accept datagrams for any IP interface associated with the node, the value is 0.0.0.0.
udpLocalPort 1.3.6.1.2.1.7.5.1.2 read-only INTEGER	The local port number of the UDP listener.	ZNET-STATUS-LOC-PORT field of the ZTCI-DDL-STATUS-PROTO structure, returned from a SPI STATUS command on PROCESS *ztsaTcpIpProcess*.

RFC Compliance

Table 8-14 summarizes compliance of UDP group support with RFC 1213.

Table 8-14. Compliance With UDP Group Definitions in RFC 1213

Object Descriptor	Compliance	Explanation
udpInDatagrams	Yes	See Table 8-13.
udpNoPorts	Partial	No corresponding HP instrumentation exists; 0 is returned.
udpInErrors	Yes	See Table 8-13.
udpOutDatagrams	Yes	See Table 8-13.
udpLocalAddress	Yes	See Table 8-13.
udpLocalPort	Yes	See Table 8-13.

udpTable Maintenance

The TCP/IP Subagent builds the udpTable by sending a STATUS SPI command to the PROCESS object, using the name of the TCP/IP subsystem being managed (*ztsaTcpIpProcess*). It then maps the values found in the ZTCI-DDL-STATUS-PROTO structure token into the appropriate MIB objects. The ZTCI-DDL-STATUS-PROTO structure returns connection information for both TCP and UDP. The subagent ensures that only the UDP connections are selected. Instances of object types that represent information about a particular UDP connection are transient. They persist only as long as the connection in question.

Rebuilding the udpTable also causes the indexes in the tcpConnTable to be updated.

Traps Generated by the TCP/IP Subagent

The TCP/IP Subagent generates traps to inform SNMP managers of the status of the TCP/IP subnets. The SNMP managers to which the TCP/IP subagent sends traps are identified by the set of trap destination definitions configured for the SNMP agent with which the TCP/IP subagent is communicating. See Section 2, Installing and Configuring the SNMP Agent, for information on configuring trap destinations.

The TCP/IP Subagent monitors the status of the TCP/IP subnets at the time interval specified in its ztsaStatusCache object. Whenever the status of a subnet changes, the TCP/IP Subagent sends a trap. The TCP/IP Subagent sends traps only to broadcast type trap destinations.

Table 8-15 describes traps the TCP/IP Subagent generates. The traps are a subset of the generic SNMP traps prescribed in RFC 1157.

Note. Currently, when managing TCP/IP resources that use an underlying ServerNet LAN Systems Access (SLSA) subsystem (G-series environment only), the TCP/IP Subagent does not generate traps.

Table 8-15. Traps Generated by TCP/IP Subagent

Generic Trap Type	Reason for Generating Trap
snDown	The state of a TCP/IP subnet and the state of the TLAM port used by the TCP/IP subnet has gone from an operational to a disabled state (ZCOM-VAL-SUMSTATE-STOPPED).
snUp	The state of a TCP/IP subnet and the state of the TLAM port used by the TCP/IP subnet has gone from a disabled to an operational state (ZCOM-VAL-SUMSTATE-STARTED).

EMS Support

This subsection describes the Event Management Service (EMS) events, listed in Table 8-16, generated by the TCP/IP Subagent (subsystem abbreviation ZTSA).

Table 8-16. TCP/IP Subagent Event Messages (page 1 of 3)

Number	ZTSA-EVT-	Standard Event and Description
1001	SUBAGENT-AVAIL	Object Available The TCP/IP Subagent primary process has been initialized. This event is generated at initial startup and whenever a takeover by the backup occurs.
1002	SUBAGENT-UNAVAIL	Object Unavailable The TCP/IP Subagent process is terminating for reasons other than an internal fault or invalid startup configuration.
1003	AGENT-OBJ-AVAIL	Object Available The interface to the SNMP agent configured for SNMP messaging has transitioned into the “connected” state.
1004	AGENT-OBJ-UNAVAIL	Object Unavailable The interface to the SNMP agent configured for SNMP messaging has transitioned into the “disconnected” state.
1005	BACKUP-OBJ-AVAIL	Object Available The backup process has transitioned into the “connected” state.
1006	BACKUP-OBJ-UNAVAIL	Object Unavailable The backup process has transitioned into the “disconnected” state.
1007	EMSCOLL-AVAIL	Object Available The EMS collector process has transitioned into the “connected” state.
1008	EMSCOLL-UNAVAIL	Object Unavailable The EMS collector process has transitioned into the “disconnected” state.
1009	TCPIP-OBJ-AVAIL	Object Available The interface to the TCP/IP resource being monitored has transitioned into a “connected” state.
1010	TCPIP-OBJ-UNAVAIL	Object Unavailable The interface to the TCP/IP resource being monitored has transitioned into a “disconnected” state.

Table 8-16. TCP/IP Subagent Event Messages (page 2 of 3)

Number	ZTSA-EVT-	Standard Event and Description
1011	OUT-OF-MEMORY	Transient Fault The subagent could not allocate memory needed for application execution. The subagent's ability to build MIB tables and/or reply to SNMP requests might be compromised by a shortage of memory.
1012	INTERNAL-FAULT	Object Unavailable An internal program logic fault has been detected in the subagent. The primary process abends after sending this event.
1013	CONFIGURATION-INVALID	Object Unavailable The startup configuration is invalid. The subagent process abends after sending this event.
1014	TAKEOVER-BY-BACKUP	Transient Fault The TCP/IP Subagent primary process has stopped, and the backup process is taking over the primary role.
1015	LANMON-OBJ-UNAVAIL	Transient Fault The LANMON process did not respond to the LM_Get_Version_ function. The ifDescr values will not contain the version string.
1022	PIFGETSTATUS-ERROR	Transient Fault The LM_Get_Status_ function returned an error. The ifTable values derived through the STATUS command are not current until the timer expires, the same ZLANMSRL function is retried, and a response obtained.

Table 8-16. TCP/IP Subagent Event Messages (page 3 of 3)

Number	ZTSA-EVT-	Standard Event and Description
1023	PIFGETATTR-ERROR	Transient Fault The LM_Get_Attributes_ function returned an error. The ifTable values derived through the INFO command are not current until the timer expires, the same ZLANMSRL function is retried, and a response obtained.
1039	LIFGETATTR-ERROR	Transient Fault The LM_Get_Attributes_ function returned an error. The ifTable values are not current until the timer expires, the same ZLANMSRL function is retried, and a response obtained.
1041	PIFGETSTATS-ERROR	Transient Fault The LM_Get_Statistics_ function returned an error. The ifTable values obtained through the STATS command are not current until the timer expires, the same ZLANMSRL function is retried, and a response obtained

Data Definitions

Because they are standard events, most of the tokens, structures, and values appearing in ZTSA events are defined in the ZSPI and ZEMS data definition files ZSPIDDL and ZEMSDDL and their associated language-specific files. Data elements defined by the TCP/IP Subagent are in the file ZTSADDL and associated language-specific files:

ZTSADDL	Data Definition Language (DDL) definitions from which the language-specific definitions are derived
ZTSAC	Definitions for C programs
ZTSACOB	Definitions for COBOL programs
ZTSAPAS	Definitions for Pascal programs
ZTSATACL	Definitions for TACL programs
ZTSATAL	Definitions for TAL programs

The complete set of SPI definition files is usually located in the ZSPIDEF subvolume of the NonStop Kernel installation volume. See the *SPI Programming Manual* for information on the ZSPI data definitions and data definition files in general. See the *EMS Manual* for information on ZEMS data definitions.

Subsystem ID

The subsystem ID that the TCP/IP Subagent uses to identify itself as the source of event messages is:

```
CONSTANT ZTSA-VAL-OWNER                    VALUE  "TANDEM".
CONSTANT ZTSA-VAL-NUMBER                   VALUE  202.
CONSTANT ZTSA-VAL-VERSION                  VALUE  VERSION "D20".
CONSTANT ZTSA-VAL-EXTERNAL-SSID            VALUE  "TANDEM.202.D20".

DEFINITION ZTSA-VAL-SSID                   TACL   SSID.
  02 z-filler  TYPE character 8      VALUE IS ZTSA-VAL-OWNER.
  02 z-owner   TYPE ZSPI-DDL-CHAR8   REDEFINES z-filler.
  02 z-number  TYPE ZSPI-DDL-INT     VALUE IS ZTSA-VAL-NUMBER.
  02 z-version TYPE ZSPI-DDL-UINT    VALUE IS ZTSA-VAL-
VERSION.
END
```

Tokens in ZTSA Event Messages

This subsection lists the tokens that the subagent includes in the event messages it generates.

TCP/IP Subagent (ZTSA) Tokens

Table 8-17 lists all the tokens defined by the subagent. These tokens are defined in the ZTSADDL definition file and are described where they appear in the event message descriptions later in this section.

Table 8-17. ZTSA Tokens in ZTSA Event Messages

Token	Contents
ZTSA-TKN-SUBAGENT	Identifies the subject of the event as being the TCP/IP Subagent process
ZTSA-TKN-AGENT	Identifies the subject of the event as being the interface between the TCP/IP Subagent and the SNMP agent with which it communicates
ZTSA-TKN-BACKUP	Identifies the subject of the event as being the TCP/IP Subagent backup process
ZTSA-TKN-EMSCOLLECTOR	Identifies the subject of the event as being the interface between the TCP/IP Subagent and the EMS collector process to which it sends its event
ZTSA-TKN-TCPIP	Identifies the subject of the event as being the interface between the TCP/IP Subagent and the TCP/IP process it is monitoring
ZTSA-TKN-SCP	Identifies the subject of the event as being the interface between the TCP/IP Subagent and the SCP process used for SPI requests

Standard EMS Tokens

Table 8-18 lists the standard EMS tokens that the TCP/IP Subagent includes in the event messages it generates. These tokens are defined in the ZEMSDDL definition file. See the *EMS Manual* for more information on these tokens and related data definitions.

Table 8-18. ZEMS Tokens in ZTSA Event Messages

Token	Contents
ZEMS-TKN-CONTENT-STANDARD	Type of standard event.
ZEMS-TKN-CONTENT-USER	Type of user-defined event. (For the TCP/IP Subagent, the value is always ZEMS-VAL-NULL.)
ZEMS-TKN-EVENTNUMBER	Event number.
ZEMS-TKN-SUBJECT-MARK	Event subject marker.
ZEMS-TKN-SUPPRESS-DISPLAY	Display/do not display event flag.
ZEMS-TKN-CHANGE-REASON	Reason for object state change.
ZEMS-TKN-STATE-CURRENT	Current object state; private enumerations: ZTSA-VAL-STATE-CONNECTED ZTSA-VAL-STATE-DISCONNECTED ZTSA-VAL-STATE-ENABLED ZTSA-VAL-STATE-DISABLED
ZEMS-TKN-STATE-PREVIOUS	Previous object state; private enumerations: ZTSA-VAL-STATE-CONNECTED ZTSA-VAL-STATE-DISCONNECTED ZTSA-VAL-STATE-ENABLED ZTSA-VAL-STATE-DISABLED
ZEMS-TKN-TXFAULT-TYPE	Reason for transient fault; private enumerations: ZTSA-VAL-TF-MEM ZTSA-VAL-TF-TAKEOVER ZTSA-VAL-TF-LANMON ZTSA-VAL-TF-GETNEXCOMP ZTSA-VAL-TF-SMQGETID ZTSA-VAL-TF-SMGQCREATE ZTSA-VAL-TF-QIONOTINIT ZTSA-VAL-TF-SMSEGATT ZTSA-VAL-TF-MODIDCREATE ZTSA-VAL-TF-PIFGETSTATUS ZTSA-VAL-TF-PIFGETATTR ZTSA-VAL-TF-LIFGETATTR

Event Message Descriptions

ZTSA event messages are described in order by event number. Each description includes the following:

- Token lists. Tokens listed as “unconditional” always appear in the event message. Tokens listed as “conditional” are included only under described conditions.

  Tokens not defined by ZTSA are listed only if they contain information that appears in the printed message text or if they contain ZTSA-defined values.

- Event message text. The event message text illustrates what is generated when the contents of the event message are displayed according to the message template defined in the file ZTSATMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the message template source file STSATMPL. The message templates for standard formatted messages are defined in the *EMS Manual*.

- Descriptions of listed tokens or values.
- The cause of the event, the conditions that prompted the TCP/IP Subagent to generate the event message.
- The effects associated with or resulting from the cause.
- Recovery procedures you can follow to solve the problem reported by the event message.
- An example of the formatted message.

1001: ZTSA-EVT-SUBAGENT-AVAIL

The TCP/IP Subagent process has completed process initialization after initial startup or after a takeover by the backup process.

Unconditional Tokens		**Value**
`ZEMS-TKN-SUBJECT-MARK`	`<1>`	`ZTSA-TKN-SUBAGENT`
`ZTSA-TKN-SUBAGENT`	`<2>`	`ztsaProcessName`
`ZEMS-TKN-EVENTNUMBER`	`<3>`	`ZTSA-EVT-SUBAGENT-OBJ-AVAIL`
`ZEMS-TKN-CHANGE-REASON`	`<4>`	`ZEMS-VAL-`*`reason`*
`ZEMS-TKN-STATE-PREVIOUS`	`<5>`	`ZTSA-VAL-STATE-DISCONNECTED`
`ZEMS-TKN-STATE-CURRENT`	`<6>`	`ZTSA-VAL-STATE-CONNECTED`
`ZEMS-TKN-CONTENT-USER`	`<7>`	`ZEMS-VAL-NULL`

Conditional Tokens

`None`

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state: <6>, user content: <7>
```

`ZTSA-TKN-SUBAGENT`

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Object available.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-SUBAGENT-OBJ-AVAIL`

is the event number (1001). The DDL AS clause of this value (“subagent-process-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP Subagent process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the TCP/IP Subagent process has come up because an underlying resource upon which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the TCP/IP Subagent process up.

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the previous state of the TCP/IP Subagent process. The DDL AS clause of this value (“disconnected”) appears in the message text following “previous state,” indicating that the subagent was in a nonoperational state.

`ZTSA-VAL-STATE-CONNECTED`

indicates the new state of the TCP/IP Subagent process. The DDL AS clause of this value (“connected”) appears in the message text following “current state,” indicating that the subagent is in a normal operational state.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The TCP/IP Subagent process completed process initialization after initial startup or after a takeover by the backup process.

Effect. The startup configuration (or last saved configuration in the case of a takeover by the backup) is used to establish connections with SNMP agent specified for the subagent's private `ztsaAgentName` object and to build MIB-II values derived from the TCP/IP process specified for the subagent's private `ztsaTcpIpProcess object`.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-02-08 12:21:51 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001001

                                   Object available  TCP/IP-SNMP-subagent -
                                   \NOVA.$ZTSA:384314581,
                                   event number: subagent-process-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined
```

1002: ZTSA-EVT-SUBAGENT-UNAVAIL

The TCP/IP Subagent process is terminating for reasons other than an internal fault or invalid startup configuration.

Unconditional Tokens	**Value**
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-SUBAGENT-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5> ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7> *code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8> ZEMS-VAL-NULL
Conditional Tokens	
ZEMS-TKN-UNDERLYING-OBJ-NAME	<9> ztsaProcessName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>, user content:
<8>
```

`ZTSA-TKN-SUBAGENT`

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-SUBAGENT-OBJ-UNAVAIL`

is the event number (1002). The DDL AS clause of this value (“subagent-process-unavailable”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP Subagent process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following “cause.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	“underlying-serv-failed” indicates that an underlying service the TCP/IP Subagent depends on has failed.
ZEMS-VAL-INTERNAL-FAILED	“internal-failed” indicates that an internal subagent error in program logic or data was encountered.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the TCP/IP Subagent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a running state, after initialization is complete.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates startup initialization state.

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the current state of the TCP/IP Subagent process. The DDL AS clause of this value (“disconnected,” defined in the preceding table) appears in the message text following “previous state.”

`ztsaProcessName`

is the name of the underlying process whose failure caused the TCP/IP Subagent process to go out of service. The name of the TCP/IP Subagent process specified for the TCP/IP Subagent’s private ztsaProcessName MIB object is inserted in the message text following “underlying object.”

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The TCP/IP Subagent process terminated for reasons other than an internal fault or invalid startup configuration.

Effect. The TCP/IP Subagent process is unavailable.

Recovery. Contact your HP service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- The saveabend file (if generated)
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products.

Sample Message

```
96-02-08 11:53:20 \NOVA.$ZTSA    TANDEM.TCPIPSA.D20    001002

                                 Object unavailable  TCP/IP-SNMP-subagent -
                                 \NOVA.$ZTSA:384126165,
                                 event number: 1002,
                                 cause: internal-failed,
                                 previous state: connected,
                                 current state disconnected, symptom string:
                                 T7862D20_01JAN96_TCPIPSA: AWAITIOX error -1,
                                 user content: undefined
```

1003: ZTSA-EVT-AGENT-OBJ-AVAIL

Unconditional Tokens **Value**

Unconditional Tokens	Value
`ZEMS-TKN-SUBJECT-MARK`	`<1> ZTSA-TKN-AGENT`
`ZTSA-TKN-AGENT`	`<2> ztsaAgentName`
`ZEMS-TKN-EVENTNUMBER`	`<3> ZTSA-EVT-AGENT-OBJ-AVAIL`
`ZEMS-TKN-CHANGE-REASON`	`<4> ZEMS-VAL-UNDERLYING-UP`
`ZEMS-TKN-STATE-PREVIOUS`	`<5> ZTSA-VAL-STATE-`*`state`*
`ZEMS-TKN-STATE-CURRENT`	`<6> ZTSA-VAL-STATE-CONNECTED`
`ZEMS-TKN-USER-CONTENT`	`<7> ZEMS-VAL-NULL`

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

`ZTSA-TKN-AGENT`

identifies the subject of the event (the SNMP agent process). The DDL heading of this token (“agent-interface”) is inserted in the message text following “Object available.”

`ztsaAgentName`

is the SNMP agent process name specified for the TCP/IP Subagent’s private ztsaAgentName MIB object.

`ZTSA-EVT-AGENT-OBJ-AVAIL`

is the event number (1003). The DDL AS clause of this value (“agent-interface-available”) appears in the message text following “event number.”

`ZEMS-VAL-UNDERLYING-UP`

indicates the reason the SNMP agent process has become available. The DDL AS clause associated with this value (“underlying-serv-up”) is inserted in the message text following “reason,” indicating that the interface with the SNMP agent process is available because an underlying resource upon which it depends has become available.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the SNMP agent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent functions dependent on the interface are not available.

`ZTSA-VAL-STATE-CONNECTED`

indicates the new state of the SNMP agent process. The DDL AS clause of this value (“connected,” defined in the preceding table) appears in the message text following “current state.”

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The TCP/IP Subagent reacquired its connection to the configured SNMP agent process.

Effect. Requests can now be made against the MIB-II and private MIB objects supported by the TCP/IP Subagent.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-02-08 12:21:51 \NOVA.$ZTSA       TANDEM.TCPIPSA.D20    001003
                                    Object available  agent-interface -
                                    \NOVA.$ZSNMP,
                                    event number: agent-interface-available,
                                    reason: underlying-serv-up,
                                    previous state: disconnected,
                                    current state: connected,
                                    user content: undefined
```

1004: ZTSA-EVT-AGENT-OBJ-UNAVAIL

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-AGENT
ZTSA-TKN-AGENT	<2> ztsaAgentName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-AGENT-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-UNDERLYING-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5> ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-UNDERLYING-OBJ-NAME	<7> ztsaAgentName
ZEMS-TKN-SYMPTOM-STRING	<8> *code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<9> ZEMS-VAL-NUL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
underlying object: <7>, symptom string: <8>, user content:
<9>
```

ZTSA-TKN-AGENT

identifies the subject of the event (the SNMP agent process). The DDL heading of this token (“agent-interface”) is inserted in the message text following “Object unavailable.”

ztsaAgentName

is the SNMP agent process name specified for the TCP/IP Subagent’s private ztsaAgentName MIB object.

ZTSA-EVT-AGENT-OBJ-UNAVAIL

is the event number (1004). The DDL AS clause of this value (“agent-interface-unavailable”) appears in the message text following “event number.”

ZEMS-VAL-UNDERLYING-FAILED

indicates the reason the SNMP agent process has become unavailable. The DDL AS clause “underlying-serv-failed” is inserted in the message text following “cause,” indicating that an underlying service the SNMP agent depends on has failed.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the SNMP agent process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZTSA-VAL-STATE-CONNECTED	"connected" indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state where subagent functions dependent on the interface are not available.

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the new state of the SNMP agent process. The DDL AS clause of this value ("disconnected," defined in the preceding table) appears in the message text following "current state."

`ztsaAgentName`

is the name of the underlying process whose failure caused the interface between the TCP/IP Subagent and the SNMP agent to go out of service. The name of the SNMP agent process specified for the TCP/IP Subagent's private ztsaAgentName MIB object is inserted in the message text following "underlying object."

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The TCP/IP Subagent lost its connection to the configured SNMP agent process.

Effect. SNMP requests cannot be made against the MIB-II or private MIB objects supported by the TCP/IP Subagent.

Recovery. Check the operating state of the SNMP agent process. If necessary, restart the SNMP agent. The TCP/IP Subagent attempts to reconnect to the SNMP agent at the rate indicated by its ztsaKeepAliveTimer object.

Sample Message

```
96-02-08 12:18:39 \NOVA.$ZTSA    TANDEM.TCPIPSA.D20    001004
                                 Object unavailable  agent-interface -
                                 \NOVA.$ZSNMP,
                                 event number: agent-interface-unavailable,
                                 cause: underlying-serv-failed,
                                 previous state: connected,
                                 current state: disconnected,
                                 underlying object: \NOVA.$ZSNMP,
                                 symptom string: T7862D20_01JAN96_TCPIPSA:
                                 updateAgent.2, agent interface error 201 on
                                 poll request,
                                 user content: undefined
```

1005: ZTSA-EVT-BACKUP-OBJ-AVAIL

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZTSA-TKN-BACKUP
ZTSA-TKN-BACKUP	<2>	ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZTSA-EVT-BACKUP-OBJ-AVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZTSA-VAL-STATE-CONNECTED
ZEMS-TKN-USER-CONTENT	<7>	ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

ZTSA-TKN-BACKUP

identifies the subject of the event (the backup TCP/IP Subagent process). The DDL heading of this token (“backup-process”) is inserted in the message text following “Object available.”

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

ZTSA-EVT-BACKUP-OBJ-AVAIL

is the event number (1005). The DDL AS clause of this value (“backup-process-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the backup TCP/IP Subagent process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the backup TCP/IP Subagent process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the backup TCP/IP Subagent process up by setting the ztsaBackupState MIB object to “enabled.”

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the backup TCP/IP Subagent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent function dependent on the interface was not available.
ZTSA-VAL-STATE-ENABLED	“enabled” indicates that the process was set by an operator such that the subagent would immediately attempt to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	“disabled” indicates that the process had been taken out of service explicitly by an operator setting the ztsaBackupState MIB object to “disabled.”

`ZTSA-VAL-STATE-CONNECTED`

indicates the current state of the backup TCP/IP Subagent process. The DDL AS clause of this value (“connected,” defined in the preceding table) appears in the message text following “current state.”

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The TCP/IP Subagent successfully started a backup process identified by `ztsaBackupCPU:ztsaBackupPIN.`

Effect. TCP/IP Subagent configuration changes are now saved to the backup. In case of a primary failure, the backup resumes subagent processing using the last successfully checkpointed configuration.

Recovery. Informational message only; no corrective action is needed.

Sample Messages

```
96-02-08 11:31:36 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001005

                                   Object available  backup-process -
                                   \NOVA.$ZTSA:383996885,
                                   event number: backup-process-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined
96-02-08 11:30:55 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001005

                                   Object available  backup-process -
                                   \NOVA.$ZTSA:383996885,
                                   event number: backup-process-available,
                                   reason: operator-initiated,
                                   previous state: enabled,
                                   current state: connected,
                                   user content: undefined
```

1006: ZTSA-EVT-BACKUP-OBJ-UNAVAIL

Unconditional Tokens **Value**

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZTSA-TKN-BACKUP
ZTSA-TKN-BACKUP	<2>	ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZTSA-EVT-BACKUP-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

Conditional Tokens		Value
ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	ztsaProcessName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content:  <8>
```

ZTSA-TKN-BACKUP

identifies the subject of the event (the backup TCP/IP Subagent process). The DDL heading of this token ("backup-process") is inserted in the message text following "Object unavailable."

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent's private ztsaProcessName MIB object.

ZTSA-EVT-SUBAGENT-OBJ-UNAVAIL

is the event number (1006). The DDL AS clause of this value ("backup-process-unavailable") appears in the message text following "event number."

`ZEMS-VAL-`*`reason`*

indicates the reason the backup TCP/IP Subagent process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following “cause.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	“underlying-serv-failed” indicates that an underlying service the backup TCP/IP Subagent process depends on has failed.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has taken the backup TCP/IP Subagent process out of service.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the backup TCP/IP Subagent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent function is not available.
ZTSA-VAL-STATE-ENABLED	“enabled” indicates that the process has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	“disabled” indicates that the process has been taken out of service explicitly by an operator setting the ztsaBackupState MIB object to “disabled.”

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the current state of the backup TCP/IP Subagent process. The DDL AS clause of this value (“disconnected,” described above) appears in the message text following “previous state.”

`ztsaProcessName`

is the name of the underlying process whose failure caused the backup TCP/IP Subagent process to go out of service. The name of the TCP/IP Subagent process specified for the subagent’s private ztsaProcessName MIB object is inserted in the message text following “underlying object.”

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The backup TCP/IP Subagent process stopped or failed to start.

Effect. The TCP/IP Subagent process continues to run without a backup. Configuration changes are not saved if a primary process failure occurs.

Recovery. Check the status of the configured backup processor (ztsaBackupCPU). If it is not available or there are insufficient resources to run a backup process, disable the backup (set ztsaBackupState “Disabled (3)”), set the ztsaBackupCPU to an available processor, and reenable the backup (set ztsaBackupState “Enabled(4).”

Sample Message

```
96-02-08 10:54:34 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001006

                                  Object unavailable  backup-process -
                                  \NOVA.$ZTSA:382971349,
                                  event number: backup-process-unavailable,
                                  cause: underlying-serv-failed,
                                  previous state: connected,
                                  current state: disconnected,
                                  underlying object: \NOVA.$ZTSA:382971349,
                                  symptom string: T7862D20_01JAN96_TCPIPSA:,
                                  user content: undefined


96-02-08 11:12:19 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001006

                                  Object unavailable  backup-process -
                                  \NOVA.$ZTSA:382971349,
                                  event number: backup-process-unavailable,
                                  cause: operator-initiated,
                                  previous state: connected,
                                  current state: disconnected,
                                  user content: undefined
```

1007: ZTSA-EVT-EMSCOLL-OBJ-AVAIL

Unconditional Tokens **Value**

Unconditional Tokens	Value
`ZEMS-TKN-SUBJECT-MARK`	`<1> ZTSA-TKN-EMSCOLLECTOR`
`ZTSA-TKN-EMSCOLLECTOR`	`<2> ztsaEmsCollectorName`
`ZEMS-TKN-EVENTNUMBER`	`<3> ZTSA-EVT-EMSCOLL-OBJ-AVAIL`
`ZEMS-TKN-CHANGE-REASON`	`<4> ZEMS-VAL-`*`reason`*
`ZEMS-TKN-STATE-PREVIOUS`	`<5> ZTSA-VAL-STATE-`*`state`*
`ZEMS-TKN-STATE-CURRENT`	`<6> ZTSA-VAL-STATE-CONNECTED`
`ZEMS-TKN-USER-CONTENT`	`<7> ZEMS-VAL-NULL`

Conditional Tokens

`None`

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

`ZTSA-TKN-EMSCOLLECTOR`

identifies the subject of the event (the EMS collector process). The DDL heading of this token (“emscollector-interface”) is inserted in the message text following “Object available.”

`ztsaEmsCollectorName`

is the EMS collector process name specified for the TCP/IP Subagent’s private ztsaEmsCollectorName MIB object.

`ZTSA-EVT-EMSCOLL-OBJ-AVAIL`

is the event number (1007). The DDL AS clause of this value (“emscollector-interface-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the interface with the EMS collector process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	Associated Text (DDL AS Clause) and Meaning
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the interface with the EMS collector process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the EMS collector process up.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the interface to the EMS collector process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	Associated Text (DDL AS Clause)
ZTSA-VAL-STATE-CONNECTED	"connected" indicates a normal operational interface between the EMS collector and the subagent.
ZTSA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state such that subagent function dependent on the interface is not available.
ZTSA-VAL-STATE-ENABLED	"enabled" indicates that the interface had been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	"disabled" indicates that the interface had been taken out of service explicitly by an operator setting the ztsaEmsCollectorState MIB object to "disabled."

`ZTSA-VAL-STATE-CONNECTED`

Indicates the current state of the interface between the EMS collector process and the TCP/IP Subagent. The DDL AS clause of this value ("connected," defined in the preceding table) appears in the message text following "current state."

`ZEMS-VAL-NULL`

Indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. `The subagent acquired or reacquired its connection to the configured EMS collector process (ztsaEmsCollectorName).`

Effect. Subsequent EMS events generated by the subagent are sent to ztsaEmsCollectorName.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-02-08 12:21:51 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001007
                                  Object available  emscollector-interface -
                                  \NOVA.$p0, event number: emscollector-
                                  interface-available,
                                  reason: operator-initiated,
                                  previous state: enabled,
                                  current state: connected,
                                  user content: undefined
```

1008: ZTSA-EVT-EMSCOLL-OBJ-UNAVAIL

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZTSA-TKN-EMSCOLLECTOR
ZTSA-TKN-EMSCOLLECTOR	<2>	ztsaEmsCollectorName
ZEMS-TKN-EVENTNUMBER	<3>	ZTSA-EVT-EMSCOLL-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL
Conditional Tokens		
ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	ztsaEmsCollectorName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content: <8>
```

`ZTSA-TKN-EMSCOLLECTOR`

identifies the subject of the event (the EMS collector process). The DDL heading of this token (“emscollector-interface”) is inserted in the message text following “Object unavailable.”

`ztsaEmsCollectorName`

is the EMS collector process name specified for the TCP/IP Subagent’s private ztsaEmsCollectorName MIB object.

`ZTSA-EVT-EMSCOLL-OBJ-UNAVAIL`

is the event number (1008). The DDL AS clause of this value (“emscollector-interface-unavailable”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the interface with the EMS collector process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following "cause."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	"underlying-serv-failed" indicates that an underlying service the interface with the EMS collector process depends on has failed.
ZEMS-VAL-OPERATOR-INITIATED	"operator-initiated" indicates that an operator has taken the EMS collector process out of service.

`ZTSA-VAL-STATE-`*`state`*

Indicates the previous state of the interface with the EMS collector process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZTSA-VAL-STATE-CONNECTED	"connected" indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state where subagent function dependent on the interface is not available.
ZTSA-VAL-STATE-ENABLED	"enabled" indicates that the interface has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	"disabled" indicates that the interface has been taken out of service explicitly by an operator setting the ztsaEmsCollectorState MIB object to "disabled."

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the current state of the interface with the EMS collector process. The DDL AS clause of this value ("disconnected," described in the preceding table) appears in the message text following "current state."

`ztsaEmsCollectorName`

is the name of the underlying process whose failure caused the interface between the TCP/IP Subagent and the EMS collector to go out of service. The name of the EMS collector process specified for the subagent’s private ztsaEmsCollectorName MIB object is inserted in the message text following “underlying object.”

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

Indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

Indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The subagent lost its connection to the configured EMS collector process (ztsaEmsCollectorName).

Effect. Events generated by the subagent are discarded until the EMS collector interface is returned to a connected state. The subagent attempts to reacquire the EMS collector connection whenever the subagent attempts to generate an event message.

Recovery. Check the operating state of the EMS collector process (ztsaEmsCollectorName).

Sample Message

```
96-02-08 11:31:35 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001008

                                  Object unavailable emscollector-interface -
                                  \NOVA.$p0, event number: emscollector-
                                  interface-unavailable,
                                  cause: underlying-serv-failed,
                                  previous state: connected,
                                  current state: disconnected,
                                  underlying object: \NOVA.$p0,
                                  symptom string: T7862D20_01JAN96_TCPIPSA:,
                                  user content: undefined


96-02-08 12:03:00 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001008

                                  Object unavailable emscollector-interface -
                                  \NOVA.$p0, event number: emscollector-
                                  interface-unavailable,
                                  cause: operator-initiated,
                                  previous state: connected,
                                  current state: disconnected,
                                  user content: undefined
```

1009: ZTSA-EVT-TCPIP-OBJ-AVAIL

Unconditional Tokens	Value
`ZEMS-TKN-SUBJECT-MARK`	`<1> ZTSA-TKN-TCPIP`
`ZTSA-TKN-TCPIP`	`<2> ztsaTcpIpProcess`
`ZEMS-TKN-EVENTNUMBER`	`<3> ZTSA-EVT-TCPIP-OBJ-AVAIL`
`ZEMS-TKN-CHANGE-REASON`	`<4> ZEMS-VAL-`*`reason`*
`ZEMS-TKN-STATE-PREVIOUS`	`<5> ZTSA-VAL-STATE-`*`state`*
`ZEMS-TKN-STATE-CURRENT`	`<6> ZTSA-VAL-STATE-CONNECTED`
`ZEMS-TKN-USER-CONTENT`	`<7> ZEMS-VAL-NULL`

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

`ZTSA-TKN-TCPIP`

identifies the subject of the event (the TCP/IP process). The DDL heading of this token (“tcpip-resource”) is inserted in the message text following “Object available.”

`ztsaTcpIpProcess`

is the TCP/IP process name specified for the TCP/IP Subagent’s private ztsaTcpIpProcess MIB object.

`ZTSA-EVT-TCPIP-OBJ-AVAIL`

is the event number (1009). The DDL AS clause of this value (“tcpip-resource-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	Associated Text (DDL AS Clause) and Meaning
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the interface with the TCP/IP process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the TCP/IP process up.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the TCP/IP process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a normal operational interface between the TCP/IP Subagent and the TCP/IP process it is managing.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state such that subagent function dependent on the interface is not available.
ZTSA-VAL-STATE-ENABLED	“enabled” indicates that the interface to the TCP/IP resource has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	“disabled” indicates that the interface to TCP/IP resource has been taken out of service explicitly by an operator setting the ztsaTcpIpState MIB object to “disabled.”

`ZTSA-VAL-STATE-CONNECTED`

indicates the current state of the TCP/IP process. The DDL AS clause of this value (“connected,” defined in the preceding table) appears in the message text following “current state.”

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The subagent acquired or reacquired its connection to the configured TCP/IP process (ztsaTcpIpProcess).

Effect. The MIB-II object values are rebuilt using SPI responses from the TCP/IP process (ztsaTcpIpProcess). When the update sequence is complete (as indicated by ztsaRefreshNow value = "autoDynamicRefresh (0)"), SNMP requests made against the MIB-II objects return current values.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-02-08 12:08:44 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001009

                                   Object available  tcpip-resource -
                                   \NOVA.$ZTC0
                                   event number: tcpip-resource-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined


96-02-08 12:21:51 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001009

                                   Object available  tcpip-resource -
                                   \NOVA.$foo,
                                   event number: tcpip-resource-available,
                                   reason: operator-initiated,
                                   previous state: enabled,
                                   current state: connected,
                                   user content: undefined
```

1010: ZTSA-EVT-TCPIP-OBJ-UNAVAIL

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZTSA-TKN-TCPIP
ZTSA-TKN-TCPIP	<2>	ztsaTcpIpProcess
ZEMS-TKN-EVENTNUMBER	<3>	ZTSA-EVT-TCPIP-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	*underlying-process*

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content: <8>
```

ZTSA-TKN-TCPIP

identifies the subject of the event (the TCP/IP process). The DDL heading of this token (“tcpip-resource”) is inserted in the message text following “Object unavailable.”

`ztsaTcpIpProcess`

is the TCP/IP process name specified for the TCP/IP Subagent's private ztsaTcpIpProcess MIB object.

`ZTSA-EVT-TCPIP-OBJ-UNAVAIL`

is the event number (1010). The DDL AS clause of this value ("tcpip-resource-unavailable") appears in the message text following "event number."

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following "cause."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	"underlying-serv-failed" indicates that an underlying service the TCP/IP process depends on has failed.
ZEMS-VAL-OPERATOR-INITIATED	"operator-initiated" indicates that an operator has taken the TCP/IP process out of service.

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the TCP/IP process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZTSA-VAL-STATE-CONNECTED	"connected" indicates a normal operational state.
ZTSA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state such that subagent function dependent on the interface is not available.
ZTSA-VAL-STATE-ENABLED	"enabled" indicates that the interface to the TCP/IP resource has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZTSA-VAL-STATE-DISABLED	"disabled" indicates that the interface to TCP/IP resource has been taken out of service explicitly by an operator setting the ztsaTcpIpState MIB object to "disabled."

ZTSA-VAL-STATE-DISCONNECTED

indicates the current state of the TCP/IP process. The DDL AS clause of this value ("disconnected," defined in the preceding table) appears in the message text following "current state."

underlying-process

is the name of the underlying process whose failure caused the interface between the TCP/IP Subagent and the TCP/IP process to go out of service. The name of one of the following processes is inserted in the message text following “underlying object”:

- The name of the TCP/IP process specified for the subagent’s private ztsaTcpIpProcess MIB object
- The name of the SCP process specified for the subagent’s private ztsaScpProcess MIB object
- The name of the SNMP agent process specified for the subagent’s private ztsaAgentName MIB object

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

code-location/internal-context-text

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The TCP/IP Subagent process lost its connection to the configured TCP/IP process (ztsaTcpIpProcess), the SCP SPI interface process (ztsaScpProcess), or the SNMP agent process (ztsaAgentName).

If the underlying object is the SNMP agent process (ztsaAgentName), the subagent has lost its connection to the agent process and has temporarily disconnected the TCP/IP interface to conserve resources.

Effect. Values for the MIB-II objects supported by the TCP/IP Subagent are not computed. SNMP requests for these MIB-II objects receive an SNMP-ERR-GEN-ERROR response. Private MIB objects supported by the TCP/IP Subagent can still be accessed.

Recovery. If the underlying object is the TCP/IP process or the SCP process, use SCF to check the operating state of the TCP/IP or SCP process. The subagent attempts to reacquire the TCP/IP connection according to the value of its ztsaKeepAliveTimer object.

If no error conditions are found in the TCP/IP and SCP processes, or if the subagent fails to reacquire the SCP interface, stop and restart the subagent.

If the underlying object is the SNMP agent process, check the operating state of the SNMP agent process. If necessary, restart the SNMP agent. The TCP/IP Subagent attempts to reconnect to the SNMP agent at the rate indicated by its ztsaKeepAliveTimer object.

Sample Message

```
96-02-08 12:21:51 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001010

                                   Object unavailable tcpip-resource -
                                   \NOVA.$foo,
                                   event number: tcpip-resource-unavailable,
                                   cause: underlying-serv-failed,
                                   previous state: connected,
                                   current state: disconnected,
                                   underlying object: \NOVA.$ztcx,
                                   symptom string: T7862D20_01JAN96_TCPIPSA:
                                   updateSCP.6 retcode (-10),
                                   user content: undefined


96-02-08 12:18:39 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001010

                                   Object unavailable tcpip-resource -
                                   \NOVA.$ZTC0,
                                   event number: tcpip-resource-
                                   unavailable, cause: underlying-serv-
                                   failed, previous state: connected,
                                   current state: disconnected,
                                   underlying object: \NOVA.$ZSNMP,
                                   symptom string: T7862D20_01JAN96_TCPIPSA:,
                                   user content: undefined
```

1011: ZTSA-EVT-OUT-OF-MEMORY

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-OUT-OF-MEMORY
ZEMS-TKN-TXFAULT-TYPE	<4> ZTSA-VAL-TF-MEM
ZEMS-TKN-USER-CONTENT	<5> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number <3>, fault type: <4>
user content: <5>
```

ZTSA-TKN-SUBAGENT

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-OUT-OF-MEMORY`

is the event number (1011). The DDL AS clause of this value (“out-of-memory”) appears in the message text following “event number.”

`ZTSA-VAL-TF-MEMORY`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Memory-allocation”) is inserted in the message text following “fault type.”

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The extended memory segment of the TCP/IP Subagent process is full and could not be extended to satisfy a memory allocation request.

Effect. The operation that was in progress is aborted. This might result in objects transitioning into the “disconnected” state to be later reconnected. The subagent functionality will be impaired as long as the memory shortage continues.

Recovery. Free swap space on the subagent process swap volume, or stop and restart the subagent with a different swap volume specified on the run line. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products.

Sample Message

```
96-02-08 10:51:02 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001011

                                   Transient Fault  TCP/IP-SNMP-subagent -
                                   \NOVA.$ZTSA:382971349,
                                   event number: out-of-memory,
                                   fault type: Memory-allocation,
                                   user content: undefined
```

1012: ZTSA-EVT-INTERNAL-FAULT

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> *ztsaProcessName*
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-INTERNAL-FAULT
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-INTERNAL-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5> ZTSA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7> *code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>, symptom string: <7>
user content: <8>
```

ZTSA-TKN-SUBAGENT

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

ZTSA-EVT-INTERNAL-FAULT

is the event number (1012). The DDL AS clause of this value (“internal-fault”) appears in the message text following “event number.”

ZEMS-VAL-INTERNAL-FAILED

indicates that an internal error was encountered. The DDL AS clause associated with this value (“internal-failed”) is inserted in the message text following “cause.”

`ZTSA-VAL-STATE-`*`state`*

indicates the previous state of the TCP/IP Subagent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZTSA-VAL-STATE-CONNECTED	“connected” indicates a subagent in a running state, having completed its startup initialization.
ZTSA-VAL-STATE-DISCONNECTED	“disconnected” indicates a subagent performing startup initialization.

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the current state of the TCP/IP Subagent process. The DDL AS clause of this value (“disconnected,” defined in the preceding table) appears in the message text following “current state.”

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The TCP/IP Subagent internal consistency-checking logic has detected a program logic error.

Effect. The TCP/IP Subagent terminates.

Recovery. Contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- The saveabend file (if generated)
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products.

Sample Message

```
96-02-08 11:49:38 \NOVA.$ZTSA      TANDEM.TCPIPSA.D20    001012

                                   Object unavailable  TCP/IP-SNMP-
                                   subagent - \NOVA.$ZTSA:384117973,
                                   event number: internal-fault,
                                   cause: internal-failed,
                                   previous state: connected,
                                   current state: disconnected,
                                   symptom string:T7862D20_01JAN96_TCPIPSA:
                                   AWAITIOX error -1,
                                   user content: undefined
```

1013: ZTSA-EVT-CONFIGURATION-INVALID

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-CONFIGURATION-INVALID
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-INTERNAL-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5> ZTSA-VAL-STATE-ENABLED
ZEMS-TKN-STATE-CURRENT	<6> ZTSA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7> *configuration-fault-detail*
ZEMS-TKN-USER-CONTENT	<8> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>, symptom string: <7>
user content: <8>
```

`ZTSA-TKN-SUBAGENT`

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-CONFIGURATION-INVALID`

is the event number (1013). The DDL AS clause of this value (“startup-configuration-invalid”) appears in the message text following “event number.”

`ZEMS-VAL-INTERNAL-FAILED`

indicates that an internal error was encountered. The DDL AS clause associated with this value (“internal-failed”) is inserted in the message text following “cause.”

`ZTSA-VAL-STATE-ENABLED`

indicates the previous state of the TCP/IP Subagent process. The DDL AS clause of this value (“enabled”) appears in the message text following “previous state.”

`ZTSA-VAL-STATE-DISCONNECTED`

indicates the current state of the TCP/IP Subagent process. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state.”

configuration-fault-detail

provides additional information about the configuration in the following format:

```
Agent process name parameter bad-value invalid
TCP/IP process name parameter bad-value invalid
Non-local TCP/IP process name parameter bad-value
Collector process name parameter bad-value invalid
Info cache value bad-value, out of range
Stats cache value bad-value, out of range
Status cache value bad-value, out of range
Keep alive timer value bad-value, out of range
Backup CPU value bad-value, out of range
SCP process name parameter bad-value invalid
Non-local SCP process name parameter bad-value
Unrecognized parameter type bad-value
Incomplete parameter specification bad-value, missing
value
Parameter bad-value, specified multiple times
```

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The startup parameters passed to the subagent process contain invalid definitions or the specified agent process was not found.

Effect. The TCP/IP Subagent terminates.

Recovery. Check the startup parameter string passed to subagent. Check the operating state of the specified SNMP agent process to ensure that it is running and ready to accept subagent connections.

Note. Event 1013 is sent to the EMS collector that is in effect when the parsing error is detected. If the -c startup parameter is specified later in the startup string, after the error is detected, the event is sent to the default collector $0. If $0 is not available, no event is generated.

Sample Message

```
96-02-08 11:24:51 \NOVA.$ZTSA    TANDEM.TCPIPSA.D20    001013

                                 Object unavailable  TCP/IP-SNMP-subagent -
                                 \NOVA.$ZTSA:383971797,
                                 event number: startup-configuration-invalid,
                                 cause: internal-failed,
                                 previous state: enabled,
                                 current state: disconnected,
                                 symptom string: T7862D20_01JAN96_TCPIPSA:
                                 Agent process name parameter <ZSNMP1>
                                 invalid,
                                 user content: undefined
```

1014: ZTSA-EVT-TAKEOVER-BY-BACKUP

Unconditional Tokens	Value
ZEMS-TKN-EVENTNUMBER	<1> ZTSA-EVT-TAKEOVER-BY-BACKUP
ZEMS-TKN-TXFAULT-TYPE	<2> ZTSA-VAL-TF-TAKEOVER

Conditional Tokens

None

Message Text

<1>, <2>

ZTSA-EVT-TAKEOVER-BY-BACKUP

is the event number (1014). The DDL AS clause of this value (“Takeover-by-backup”) appears in the message text.

ZTSA-VAL-TF-TAKEOVER

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Primary-stopped”) appears in the message text.

Cause. The subagent primary process terminated. The recognized causes are either process termination or primary processor failure.

Effect. The backup process takes over the subagent primary role using the last known configuration parameters. Any transactions that were in progress at the time of the primary failure are lost.

The new primary attempts to restart a backup process in the former primary processor according to the ztsaKeepAliveTimer interval.

Recovery. If the cause was a processor failure, the backup processor (ztsaBackupCPU) should be changed to a currently active processor.

If the cause was a primary process failure, the event log should be examined to determine the underlying cause for the process abend.

Sample Message

```
96-02-08 11:31:35 \NOVA.$ZTSA     TANDEM.TCPIPSA.D20    001014
                                  Takeover by backup process, Primary-stopped
```

1015: ZTSA-EVT-LANMON-OBJ-UNAVAIL

Unconditional Tokens **Value**

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-LANMON-OBJ-UNAVAIL
ZEMS-TKN-TXFAULT-TYPE	<4> ZTSA-VAL-TF-LANMON
ZEMS-TKN-USER-CONTENT	<5> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number: <3>, fault type: <4>,
user content: <5>
```

ZTSA-TKN-SUBAGENT

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

ZTSA-EVT-LANMON-OBJ-UNAVAIL

is the event number (1015). The DDL AS clause of this value (“lanmon-resource-unavailable”) appears in the message text following “event number.”

ZTSA-VAL-TF-LANMON

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“lanmon-not-responding”) appears in the message text following “fault type.”

ZEMS-VAL-NULL

indicates the type of subsystem defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The LANMON process did not respond to the LM_Get_Version_ function.

Effect. The ifDescr object values will not contain the LANMON version string.

Recovery. Generally, this condition is transient, and no corrective action is necessary. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
98-04-08 11:31:35 \NOVA.$ZTSA    TANDEM.TCPIPSA.G05   001015  Transient
                                 Fault  TCP/IP-SNMP-Subagent -
                                 \NOVA.$ZTSA:3693561,
                                 event number: lanmon-resource-unavailable,
                                 fault type: lanmon-not-responding,
                                 user content: undefined
```

1022: ZTSA-EVT-PIFGETSTATUS-ERROR

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2>	ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZTSA-EVT-PIFGETSTATUS-ERROR
ZEMS-TKN-TXFAULT-TYPE	<4>	ZTSA-VAL-TF-PIFGETSTATUS
ZEMS-TKN-USER-CONTENT	<5>	ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number: <3>, fault type: <4>,
user content: <5>
```

`ZTSA-TKN-SUBAGENT`

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-PIFGETSTATUS-ERROR`

is the event number (1022). The DDL AS clause of this value ("LM_Get_Status_error-on-PIF") appears in the message text following "event number."

`ZTSA-VAL-TF-PIFGETSTATUS`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value ("PIF-status-cmd-not-completed") appears in the message text following "fault type."

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The LM_Get_Status_ function returned an error for the PIF.

Effect. The ifTable values obtained through the STATUS command are not current until the timer expires, the same ZLANMSRL call is retried, and a response obtained.

Recovery. Generally, this condition is transient, and no corrective action is necessary. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
98-04-08 11:31:35 \NOVA.$ZTSA    TANDEM.TCPIPSA.G05   001022  Transient
                                 Fault  TCP/IP-SNMP-Subagent -
                                 \NOVA.$ZTSA:3693561,
                                 event number: LM_Get_Status_error-on-PIF,
                                 fault type: PIF-status-cmd-not-completed,
                                 user content: undefined
```

1023: ZTSA-EVT-PIFGETATTR-ERROR

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-PIFGETATTR-ERROR
ZEMS-TKN-TXFAULT-TYPE	<4> ZTSA-VAL-TF-PIFGETATTR
ZEMS-TKN-USER-CONTENT	<5> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number: <3>, fault type: <4>,
user content: <5>
```

`ZTSA-TKN-SUBAGENT`

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

`ztsaProcessName`

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-PIFGETATTR-ERROR`

is the event number (1023). The DDL AS clause of this value (“LM_Get_attributes_error-on-PIF”) appears in the message text following “event number.”

`ZTSA-VAL-TF-PIFGETATTR`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“PIF-attr-cmd-not-completed”) appears in the message text following “fault type.”

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The LM_Get_Attributes_ function returned an error on the PIF.

Effect. The ifTable values obtained through the INFO command are not current until the timer expires, the same ZLANMSRL call is retried, and a response obtained.

Recovery. Generally, this condition is transient, and no corrective action is necessary. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
98-04-08 11:31:35 \NOVA.$ZTSA   TANDEM.TCPIPSA.G05   001023  Transient Fault
                                TCP/IP-SNMP-Subagent-
                                \NOVA.$ZTSA:3693561, event number:
                                LM_Get_Attributes-error-on-PIF, fault type:
                                PIF-attr-cmd-not-completed, user content:
                                undefined
```

1039: ZTSA-EVT-LIFGETATTR-ERROR

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-LIFGETATTR-ERROR
ZEMS-TKN-TXFAULT-TYPE	<4> ZTSA-VAL-TF-LIFGETATTR
ZEMS-TKN-USER-CONTENT	<5> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number: <3>, fault type: <4>,
user content: <5>
```

ZTSA-TKN-SUBAGENT

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

`ZTSA-EVT-LIFGETATTR-ERROR`

is the event number (1039). The DDL AS clause of this value (“LM_Get_Attributes_error-on-LIF”) appears in the message text following “event number.”

`ZTSA-VAL-TF-LIFGETATTR`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“LIF-attr-cmd-not-completed”) appears in the message text following “fault type.”

`ZEMS-VAL-NULL`

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The LM_Get_Attributes_ function returned an error for the LIF.

Effect. The ifTable values are not current until the timer expires, the same ZLANMSRL call is retried, and a response obtained.

Recovery. Generally, this condition is transient, and no corrective action is necessary. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
98-04-08 11:31:35 \NOVA.$ZTSA   TANDEM.TCPIPSA.G05   001039  Transient
                                Fault  TCP/IP-SNMP-Subagent -
                                \NOVA.$ZTSA:3693561,
                                event number: LM_Get_Attributes_error-on-LIF,
                                fault type: LIF-attr-cmd-not-completed,
                                user content: undefined
```

1041: ZTSA-EVT-PIFGETSTATS-ERROR

Unconditional Tokens **Value**

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZTSA-TKN-SUBAGENT
ZTSA-TKN-SUBAGENT	<2> ztsaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZTSA-EVT-PIFGETSTATS-ERROR
ZEMS-TKN-TXFAULT-TYPE	<4> ZTSA-VAL-TF-PIFGETSTATS
ZEMS-TKN-USER-CONTENT	<5> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number: <3>, fault type: <4>,
user content: <5>
```

ZTSA-TKN-SUBAGENT

identifies the subject of the event (the TCP/IP Subagent process). The DDL heading of this token (“TCP/IP-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

ztsaProcessName

is the TCP/IP Subagent process name specified for the TCP/IP Subagent’s private ztsaProcessName MIB object.

ZTSA-EVT-PIFGETSTATS-ERROR

is the event number (1041). The DDL AS clause of this value (“lmPIFgetstats-error”) appears in the message text following “event number.”

ZTSA-VAL-TF-PIFGETSTATS

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“PIF-stats-cmd-not-completed”) appears in the message text following “fault type.”

ZEMS-VAL-NULL

indicates the type of subsystem-defined event. Because the TCP/IP Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The LM_Get_Statistics_ function returned an error.

Effect. The ifTable values obtained through the STATS command are not current until the timer expires, the same ZLANMSRL call is retried, and a response obtained.

Recovery. Generally, this condition is transient, and no corrective action is necessary. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
97-04-08 11:31:35 \NOVA.$ZTSA    TANDEM.TCPIPSA.G02   001041  Transient
                                 Fault  TCP/IP-SNMP-Subagent -
                                 \NOVA.$ZTSA:3693561,
                                 event number: lmPIFgetstats-error,
                                 fault type: PIF-stats-cmd-not-completed,
                                 user content: undefined
```

Converting Events to Traps

Any process that generates events, including the TCP/IP Subagent, can have its events translated into traps by the EMS Trap Subagent. The EMS Subagent uses an event filter known as a “routing distributor” to determine where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert into traps. Refer to Section 9, EMS Trap Subagent, for more information.

9 EMS Trap Subagent

The EMS Trap Subagent translates Event Management Service (EMS) events into SNMP traps whose objects are defined in the EMS Trap MIB. This section describes the EMS Trap Subagent and its MIB.

Architectural Overview

EMS is a set of processes that collect event messages and then distribute them to various destinations, such as viewing consoles and printers. Event messages, generated by most HP subsystems and by many customer applications that run on NonStop systems, describe significant situations, such as loss of a function or need for operator action. Many of these events are candidates for traps for SNMP managers.

Figure 9-1 identifies some of the EMS processes and the objects they use:

- Event messages are sent by subsystems and applications to one or more **collectors**. A collector known as $0 is configured at system generation and this is the primary collector. Other (alternate) collectors can be started after the system load.
- Each collector writes the events it receives to a **log file.**
- **Distributors** read events from log files and route them to specific destinations. Different kinds of distributors support different destinations, as the figure illustrates.
- Distributors can be configured to use a **filter** to select events to route.
- Events that need to be read as text can be formatted by using a **template**. A template provides instructions for how information should be displayed. Templates can be used by subsystems and applications, as well as by printing and compatibility distributors.

As Figure 9-2 indicates, the EMS Trap Subagent is implemented within EMS as a special kind of printing distributor known as a **routing distributor**. Routing distributors use destination information encoded in filters to determine where to send selected events. To optimize fault tolerance for the EMS Trap Subagent, the routing distributor can be run as a process pair.

You define a filter to identify which event messages to convert into trap messages and to identify the agent process to which to forward them. The EMS Trap Subagent constructs trap messages by using an EMS template in ZSMPTMPL and then forwards the traps to the SNMP agent. The SNMP agent sends trap messages to SNMP managers configured to receive them. These SNMP managers can interpret the traps after the EMS Trap MIB definition has been installed.

Figure 9-1. EMS Components

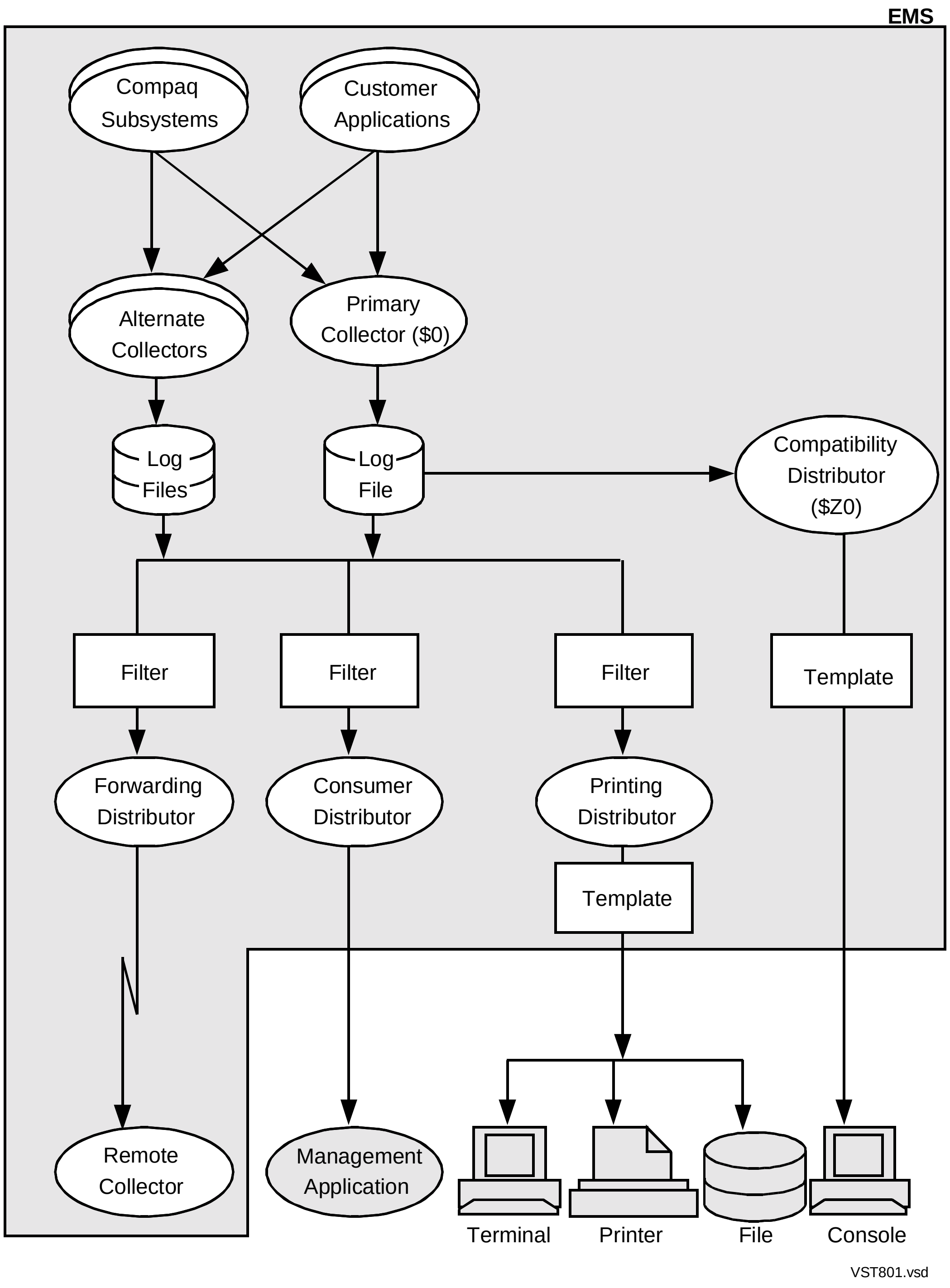
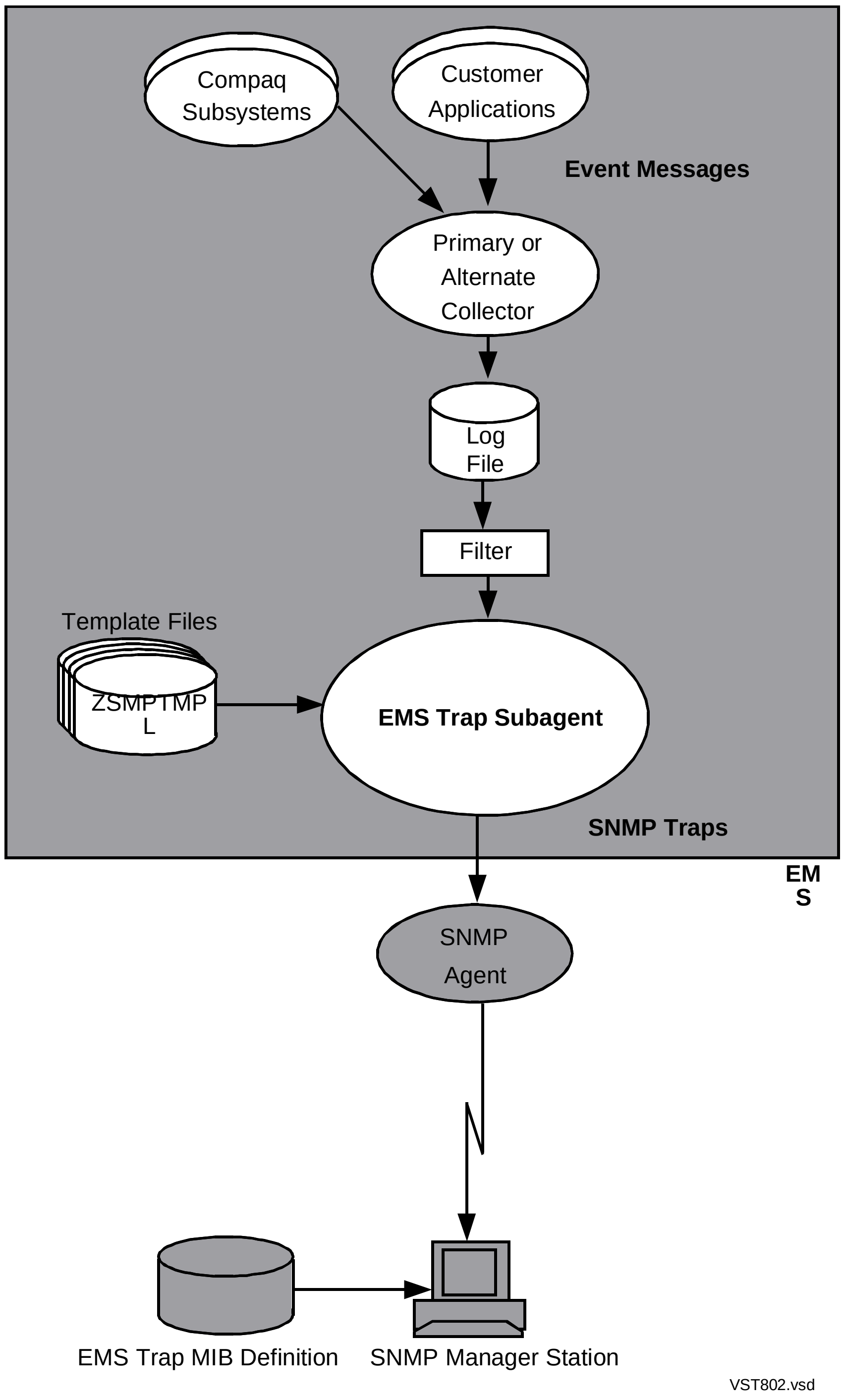


Figure 9-2. EMS Trap Subagent Components



RFC Compliance

The EMS Trap MIB complies with RFC 1215, *A Convention for Defining Traps for Use with the SNMP*.

Installation

This subsection describes how to install the EMS Trap Subagent.

Dependencies

The following products must be configured and running for the EMS Trap Subagent to operate properly:

- SNMP agent (D23 or later). In addition to the SNMP agent, the subagent needs these files, distributed with the SNMP agent:
  - SSMPTMPL: Template Language source code for a trap message DSM template.
  - ZSMPTMPL: The TEMPL-compiled version of SSMPTMPL.
- EMS (D21 or later). The subagent uses EMSGET.
- EMSDIST (D22 or later). This is the object code for the EMS distributor, which contains the event-to-trap translation functions. Also shipped with EMSDIST is the ASN.1 source code file (EMSMIBA) for the EMS trap MIB.
- DSM Template Services (D21 or later).
- At least one SNMP manager communicating with the SNMP agent that is capable of displaying values for the variable bindings of enterprise-specific traps.

Installation Steps

Run the Distributed Systems Management/Software Configuration Manager (DSM/SCM) to install EMS, DSM Template Services, and the SNMP agent. Refer to the *DSM/SCM User’s Guide* for complete software installation information.

The EMS Trap Subagent is installed automatically when you install EMS. Ensure that the EMS file named ZSMPTMPL is installed in the system template file. When you use the SYSGEN utility to generate your system, ZSMPTMPL is automatically installed in the system template file. Otherwise, you must manually install it. Refer to the *EMS Manual* for more information.

Then load the ASN.1 source code for the EMS Trap MIB, installed in a file named EMSMIBA and shown in Example 9-1, onto any SNMP manager you want to receive traps from the subagent. Compile the MIB as described in the documentation provided with your SNMP manager. Compiling the MIB makes it possible for the SNMP manager to display the names (rather than only numeric object identifiers) of MIB objects.

The subagent can then be configured.

Example 9-1. ASN.1 Source Code for EMS Trap MIB (page 1 of 3)

```
--
-- This MIB defines Tandem EMS event messages as traps
--
--  The MIB source is set for installation on HP OpenView for
--  Windows and Unix.  Installation on other platforms may
require
--  editing:

--
--  Sun NetManager:  comment out entire IMPORTS section;
--                   uncomment iso, dod, etc.
--

Tandem-EMS-MIB DEFINITIONS ::= BEGIN

   IMPORTS
     enterprises FROM RFC1155-SMI;
--   DisplayString, OBJECT-TYPE FROM RFC-1212
--   TRAP-TYPE FROM RFC-1215;

--
-- EMS event MIB
--

--  iso             OBJECT IDENTIFIER ::= { 1 }
--  org             OBJECT IDENTIFIER ::= { iso 3 }
--  dod             OBJECT IDENTIFIER ::= { org 6 }
--  internet        OBJECT IDENTIFIER ::= { dod 1 }
--  directory       OBJECT IDENTIFIER ::= { internet 1 }
--  mgmt            OBJECT IDENTIFIER ::= { internet 2 }
--  experimental    OBJECT IDENTIFIER ::= { internet 3 }
--  private         OBJECT IDENTIFIER ::= { internet 4 }
--  enterprises     OBJECT IDENTIFIER ::= { private 1 }


  tandem           OBJECT IDENTIFIER ::= { enterprises 169 }
  nonstopsystems   OBJECT IDENTIFIER ::= { tandem 3 }
  ems              OBJECT IDENTIFIER ::= { nonstopsystems 12 }
  zsnmp            OBJECT IDENTIFIER ::= { nonstopsystems 155 }
  snmpagent        OBJECT IDENTIFIER ::= { zsnmp 1 }
```

Example 9-1. ASN.1 Source Code for EMS Trap MIB (page 2 of 3)

```
--
-- items in each event
--


  TrapSSIDtype            ::= DisplayString (SIZE (5..24))
  TrapEventNumbertype     ::= DisplayString (SIZE (1..6))
  TrapContentStandardtype ::= DisplayString (SIZE (1..24))
  TrapSubjecttype         ::= DisplayString (SIZE (1..50))
  TrapProcesstype         ::= DisplayString (SIZE (1..50))
  TrapGenerationtimetype  ::= DisplayString (SIZE (19))
  TrapCriticaltype        ::= DisplayString (SIZE (1..14))
  TrapTexttype            ::= DisplayString (SIZE (1..1000))



--
-- item definitions
--

  trapSSID OBJECT-TYPE
      SYNTAX   TrapSSIDtype               --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "subsystem ID that generated the EMS event"
      ::= { ems 1}

  trapEventNumber  OBJECT-TYPE
      SYNTAX   TrapEventNumbertype        --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "Event number of event"
      ::= { ems 2}

  trapContentStandard OBJECT-TYPE
      SYNTAX   TrapContentStandardtype    --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "standard content of event"
      ::= { ems 3}

  trapSubject OBJECT-TYPE
      SYNTAX   TrapSubjecttype            --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The value of the first subject token in
                    the event "
      ::= { ems 4}
```

Example 9-1. ASN.1 Source Code for EMS Trap MIB (page 3 of 3)

```
  trapProcess OBJECT-TYPE
      SYNTAX   TrapProcesstype             --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The process name that generated the event "
      ::= { ems 5}

  trapGenerationtime OBJECT-TYPE
      SYNTAX   TrapGenerationtimetype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " GMT the event was generated, rendered
                    yyyy/mm/dd hh:mm:ss.  Timestamp is GMT. "
      ::= { ems 6}

  trapCritical OBJECT-TYPE                 --
      SYNTAX   TrapCriticaltype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " zems-tkn-critical == zspi-val-false ->
                    then '(not critical)'
                    else '(CRITICAL)    ' "
      ::= { ems 7}

  trapText OBJECT-TYPE                     --
      SYNTAX   TrapTexttype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The text expansion of the EMS event "
      ::= { ems 8}


emsMessage TRAP-TYPE
  ENTERPRISE      snmpagent
  VARIABLES       { trapSSID,
                  trapEventNumber,
                  trapContentStandard,
                  trapSubject,
                  trapProcess,
                  trapGenerationtime,
                  trapCritical,
                  trapText }
  DESCRIPTION "A generic format for an EMS event."
::= 0

END -- TANDEM-EMS-MIB
```

Configuration

Before using the EMS Trap Subagent, you need to create an event filter and might need to configure trap connections. This subsection describes these operations.

Event Filter

The EMS Trap Subagent uses an event filter known as a **routing distributor filter** to determine where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert into traps. Refer to the *EMS Manual* for information on how to encode and install routing distributor filters.

Example 9-2 shows the source code for an event filter that selects and passes all events generated by HP subsystems. This example filter also provides, as part of the routing destination statement, a DEFINE name (=_SNMP_AGENT) to identify the SNMP agent. The DEFINE name, which circumvents hard coding of an agent process name, is resolved at run time, as in Starting and Stopping the Subagent on page 9-9. The lines shown in bold type contain the event-selection specifications.

Note. See the *Operator Messages Manual* for the names, numbers, and descriptions of supported subsystems.

Example 9-2. Example Routing Distributor Filter

```
filter MYFILTER;
begin
  destination
    rid 1,
    type process,
    name =_snmp_agent,
    format off;

  [#def ssid^structure struct begin
    char z^owner(0:7);
    int  z^number;
    int  z^version;
  end;]

   == only pass events from these subsystems:
   == CPU=15 IPB=16 DISK=31 STORAGE=189
   if zspi^tkn^ssid.ssid^structure:z^owner = "TANDEM" and
     (zspi^tkn^ssid.ssid^structure:z^number =  15 or
      zspi^tkn^ssid.ssid^structure:z^number =  16 or
      zspi^tkn^ssid.ssid^structure:z^number =  31 or
      zspi^tkn^ssid.ssid^structure:z^number = 189)

    then pass ,1;
end;
```

You compile the filter with EMF after loading the Data Definition Language (DDL) definitions used by the filter:

```
#PUSH DUMMY
#LOAD /KEEP 1, LOADED DUMMY/ $SYSTEM.ZSPIDEF.ZSPITACL
#POP DUMMY
EMF /IN source-filter/ object-filter
```

The source code for the event filter is in the file identified by *`source-filter`*, and the compiled version is in the file identified by *`object-filter`*.

Trap Connections

Traps from the EMS Trap Subagent are routed to SNMP managers identified in the SNMP agent's TRAPDEST objects. Refer to Configuring Trap Destinations on page 2-38 for more information on this subject.

Starting and Stopping the Subagent

With the SNMP agent running, start EMSDIST:

```
EMSDIST /NAME, NOWAIT/ TYPE PRINTING, FILTER object-filter, &
COLLECTOR $collector-process
```

The EMF-compiled version of the routing distributor filter is in the file identified by *`object-filter`*. The collector from which events are to be taken is identified by *`collector-process`*.

If the destination statement in the filter uses a DEFINE name, you must resolve the name before starting EMSDIST as follows:

```
ADD DEFINE define-name, FILE $agent-process.#TRAP
```

where the qualifier .#TRAP is required.

In the following example, the DEFINE name (=_SNMP_AGENT) is in a routing distributor filter that was compiled into a file named MYFILTER. The agent process is named $ZSNMP, and events are to be taken from the primary EMS collector, $0.

```
ADD DEFINE =_SNMP_AGENT, FILE $ZSNMP.#TRAP
```

```
EMSDIST /NAME, NOWAIT/ TYPE PRINTING, FILTER MYFILTER, &
COLLECTOR $0
```

You can set up a TACL macro to simplify subagent invocation. Example 9-3 shows a TACL macro that accomplishes the same thing as the two commands just shown. It also contains an output statement that displays the name of the subagent process started. To start the EMS Trap Subagent, simply start the macro and identify the agent process:

```
RUN MYMACRO $ZSNMP
```

To stop the subagent process, provide its name in the STOP command:

```
STOP $ZSNMP
```

The Trap PDU

The trap PDU contains these fields:

enterprise	The object identifier for the SNMP agent, indicating the origin of the trap: 1.3.6.1.4.1.169.3.155.1.
agent-address	The Internet address of the system on which the SNMP agent forwarding the trap is installed.
generic-trap	A 16-bit number set to 6 to signify that the trap is enterpriseSpecific.
specific-trap	A 16-bit number set to 0.
time-stamp	A 32-bit number indicating how much time has passed since the SNMP agent was last started. The subagent inserts 0, and the SNMP agent inserts the value of sysUpTime, which is defined in Table 3-1, System Group Objects Supported by SNMP Agent, on page 3-4.
variable-bindings	The objects defined in the EMS Trap MIB, described in the following subsection.

Example 9-3. Example TACL Macro for Starting EMS Trap Subagent

```
?tacl routine
==
== routine: MYMACRO
==
== synopsis:   MYMACRO <SNMP-agent-process>
==
== example: MYMACRO $ZSNMP
==
== description:  MYMACRO opens the SNMP agent process and
forwards
== from $0 as SNMP traps all EMS events that pass the filter
== MYFILTER
==
#frame
#push :agent^process
[#if [#argument /value :agent^process/ processname]]
[#if "[#deviceinfo /devicetype, subtype/ [:agent^process]]"
     '<>' "31 30" |then|
    #output expected the process name of an SNMP agent
    #unframe
    #return
]

[#if [#definedelete =_snmp_agent]]
add define =_snmp_agent, file [:agent^process].#trap

emsdist /nowait, name/ type printing, filter myfilter, &
                   collector $0, gmt on
#output EMS event subagent process started; process name is
[#process].
#output Type STOP [#process] to stop sending EMS events as
traps.
#pop :agent^process
[#if [#definedelete =_snmp_agent]]
#unframe
```

The EMS Trap MIB

The EMS Trap MIB is a collection of scalar objects, identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          private (4)
            enterprises (1)
              tandem (169)
                nonstopsystems (3)
                  ems (12)
                    trapSSID (1) √
                    trapEventNumber (2) √
                    trapContentStandard (3) √
                    trapSubject (4) √
                    trapProcess (5) √
                    trapGenerationtime (6) √
                    trapCritical (7) √
                    trapText (8) √
```

Table 9-1 describes the objects in the EMS Trap MIB.

Table 9-1. Objects in the EMS Trap MIB (page 1 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapSSID 1.3.6.1.4.1.169.3.12.1 not-accessible DisplayString (SIZE (5..24))	The identifier of the subsystem or application that generated the event.	*owner.ss.version*, *owner* is the organization providing the software that generated the trap. For HP subsystems, the value is TANDEM. *ss* is the subsystem name or number. Refer to the *Operator Messages Manual* to interpret HP values. *version* identifies the software release version of the subsystem. For HP subsystems, this value matches the three-character release ID in the product release version (for example, D23).	ZSPI-TKN-SSID token value.
trapEventNumber 1.3.6.1.4.1.169.3.12.2 not-accessible DisplayString (SIZE (1..6))	An integer that uniquely identifies an event from a subsystem or application. The *Operator Messages Manual* provides cause, effect, and recovery information for events generated by HP subsystems. Use the trapSSID and the trapEventNumber to locate explanations in that manual.	Integer without leading 0s.	ZSPI-TKN-EVENTNUMBER value.

Table 9-1. Objects in the EMS Trap MIB (page 2 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapContentStandard 1.3.6.1.4.1.169.3.12.3) not-accessible DisplayString (SIZE (1..24))	An indication of whether the event is a standard event and, if so, what type of standard event. Programmers are encouraged to use standard events to improve the consistency among messages in certain categories and to promote system management automation.	One of these values:	The value of trapContentStandard corresponds to ZEMS-TKN-CONTENT-STANDARD token values as follows:
		not-specified: event is not a standard event.	not-specified (ZEMS-VAL-NULL)
		transient-fault: an error has been corrected automatically.	transient-fault (ZEMS-VAL-TRANSIENT
		object-unavailable: an object or the services it depends on are not supplying the service they were designed to provide.	object-unavailable (ZEMS-VAL-OBJECT-UNAVAILABLE)
		object-available: an object and all services it depends on are supplying the service they were designed to provide.	object-available (ZEMS-VAL-OBJECT-AVAILABLE)
		other-state-change: an object needs an operator to change or notice its state.	other-state-change (ZEMS-VAL-OTHER-STATE-CHANGE)
		attention-needed: an application cannot continue until an operator takes some action.	attention-needed (ZEMS-VAL-ATTN-NEEDED)
		attention-completed: an action described by an earlier event having the value attention-needed has been completed.	attention-completed (ZEMS-VAL-ATTN-COMPLETED)
		usage-threshold: the usage level of an object or resource exceeds the configured level.	usage-threshold (ZEMS-VAL-USAGE-THRESHOLD)
		unknown: returned for any event not associated with the values above.	

Table 9-1. Objects in the EMS Trap MIB (page 3 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapSubject 1.3.6.1.4.1.169.3.12.4) not-accessible DisplayString (SIZE (1..50))	The hardware or software component most directly involved in the event.	All values except floating-point values and values stored as structured tokens can be formatted. A question mark (?) appears when the value of the subject cannot be formatted.	The value of the first event subject token.

trapProcess
1.3.6.1.4.1.169.3.12.5
not-accessible
DisplayString (SIZE (1..50))
	The process descriptor of the process reporting the event.	For a named process:
		`node-name.process-name.sequence-number`
		
		For an unnamed process:
		`node-name.`
		
		`$:cpu:pin:sequence-number`
		
		`node-name` identifies the system on which the process resides.
		
		`process-name` is the name assigned at startup time, of the form $*`name`*, where *`name`* has from one through five characters.
		
		`sequence-number` is an integer used to distinguish among different instances of a process.
		
		`cpu:pin` is a process ID consisting of a processor module number (*`cpu`*) and a processor identification number (*`pin`*).
			ZEMS-TKN-PROC-DESC token value.

trapGenerationtime 1.3.6.1.4.1.169.3.12.6 not-accessible DisplayString (SIZE 19)	The time when the event was generated.	YYYY/MM/DD HH:MM:SS.	ZEMS-TKN-GENTIME token value.

Table 9-1. Objects in the EMS Trap MIB (page 4 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapCritical 1.3.6.1.4.1.169.3.12.7 not-accessible DisplayString (SIZE (1..14))	An indication of whether the event is critical or noncritical. Critical events have consequences that might be severe, as when they indicate potential or actual loss of data, loss of a major subsystem function, loss of fault-tolerance capability, or loss of subsystem integrity.	One of these enumerated values: (CRITICAL) (not critical)	ZEMS-TKN-EMPHASIS token value. If the value of this token is false (0), the value passed is (not critical). If the value is non-zero, the value passed is (CRITICAL).
trapText 1.3.6.1.4.1.169.3.12.8 not-accessible DisplayString (SIZE (1..1000))	Text describing the event.	ASCII text.	EMSTEXT procedure call.

10 NonStop NET/MASTER Trap Subagent

The NonStop NET/MASTER Trap Subagent translates Event Management Service (EMS) events routed to NonStop NET/MASTER MS into SNMP traps whose objects are defined in the EMS Trap MIB. It also gives you the ability to define your own trap objects, for use with an enterprise-specific trap MIB.

This section describes the NonStop NET/MASTER Trap Subagent and its MIB.

Architectural Overview

EMS is a set of processes that collect event messages and then distribute them to various destinations, including NonStop NET/MASTER. Event messages, generated by most HP subsystems and by many customer applications that run on NonStop systems, describe significant situations, such as loss of a function or need for operator action. Many of these events are candidates for traps for SNMP managers.

Figure 10-1 illustrates the relationship between EMS and NonStop NET/MASTER:

- Event messages are sent by subsystems and applications to one or more **collectors**. A collector known as $0 is configured at system generation and this collector is the primary collector. Other (secondary) collectors can be started after system load.
- Each collector writes the events it receives to a **log file**.
- **Distributors** read events from log files and route them to specific destinations. A distributor known as a **consumer distributor** is used to forward messages to the NonStop NET/MASTER environment.
- Distributors can be configured to use a **filter** to select events to route.
- Additional message filtering, and optional message modification, can be done within the NonStop NET/MASTER environment. Procedures known as **EMSPROCs** or **RMS message handlers** are used to conduct automated procedures and distribute and present messages. The **LOGPROC** process manages messages logged to a file, and the **MSGPROC** process manages those meant for viewing at an Operator Control Services (**OCS**) window. **NCL (Network Control Language) procedures** can be used to customize the presentation of messages.

Figure 10-1. EMS and NonStop NET/MASTER Components

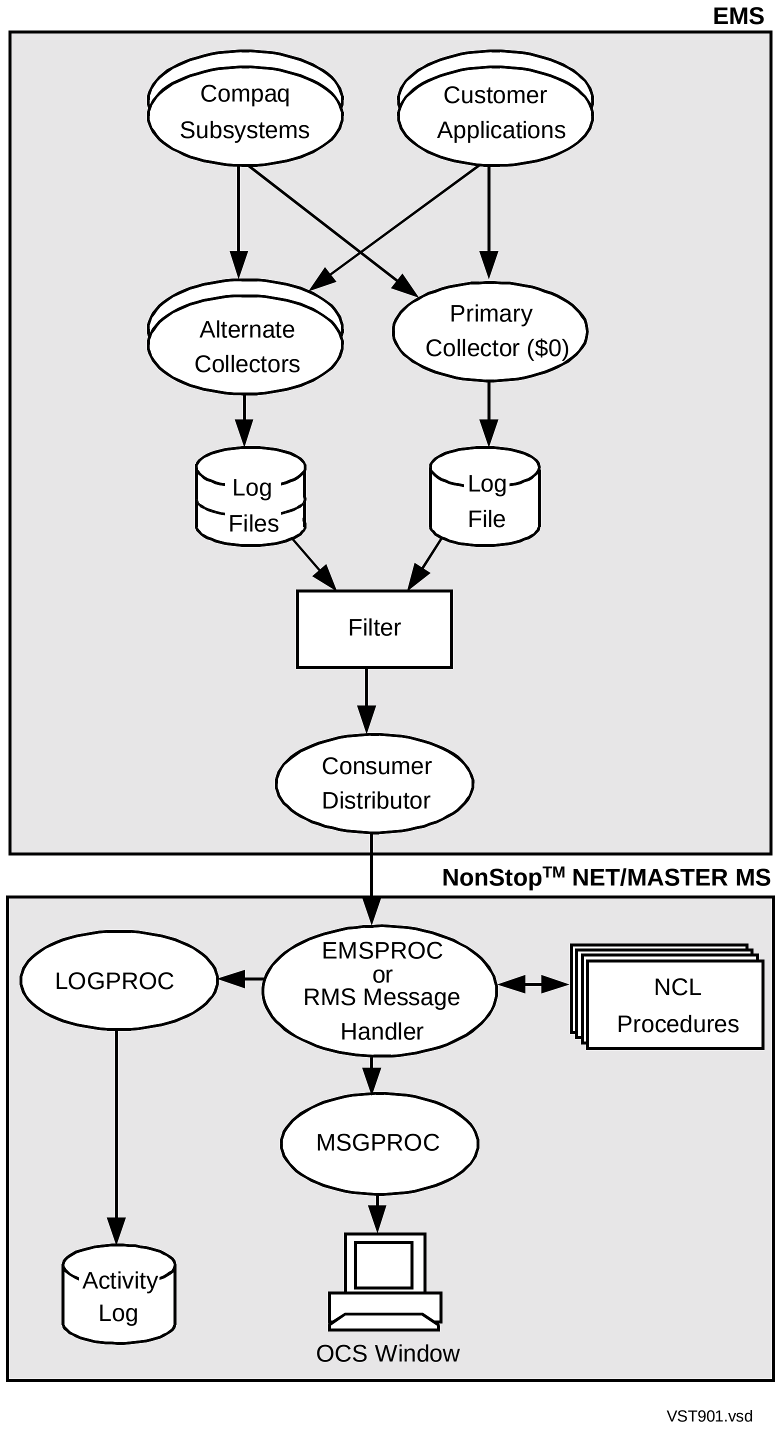


Figure 10-2 shows how the NonStop NET/MASTER Trap Subagent fits into this environment. The subagent consists of two NCL procedures, GENTRAP and NMMTRAP, which are invoked from EMSPROC or an RMS message handler:

- **GENTRAP** assembles an SNMP trap from an EMS event and calls NMMTRAP for handling communications with the SNMP agent. If you are fluent in NCL and ASN.1, you can define your own trap MIB and then modify GENTRAP to correspond. The MIB that corresponds to the trap defined in GENTRAP initially is the EMS Trap MIB.
- **NMMTRAP** contains procedures for opening and closing sessions with the SNMP agent and for forwarding traps to the SNMP agent. Do not modify this NCL code.

The subagent sends trap messages to the SNMP agent for forwarding to SNMP managers, which can interpret the traps after the EMS Trap MIB has been installed.

RFC Compliance

The EMS Trap MIB complies with RFC 1215, *A Convention for Defining Traps for Use with the SNMP*.

Installation

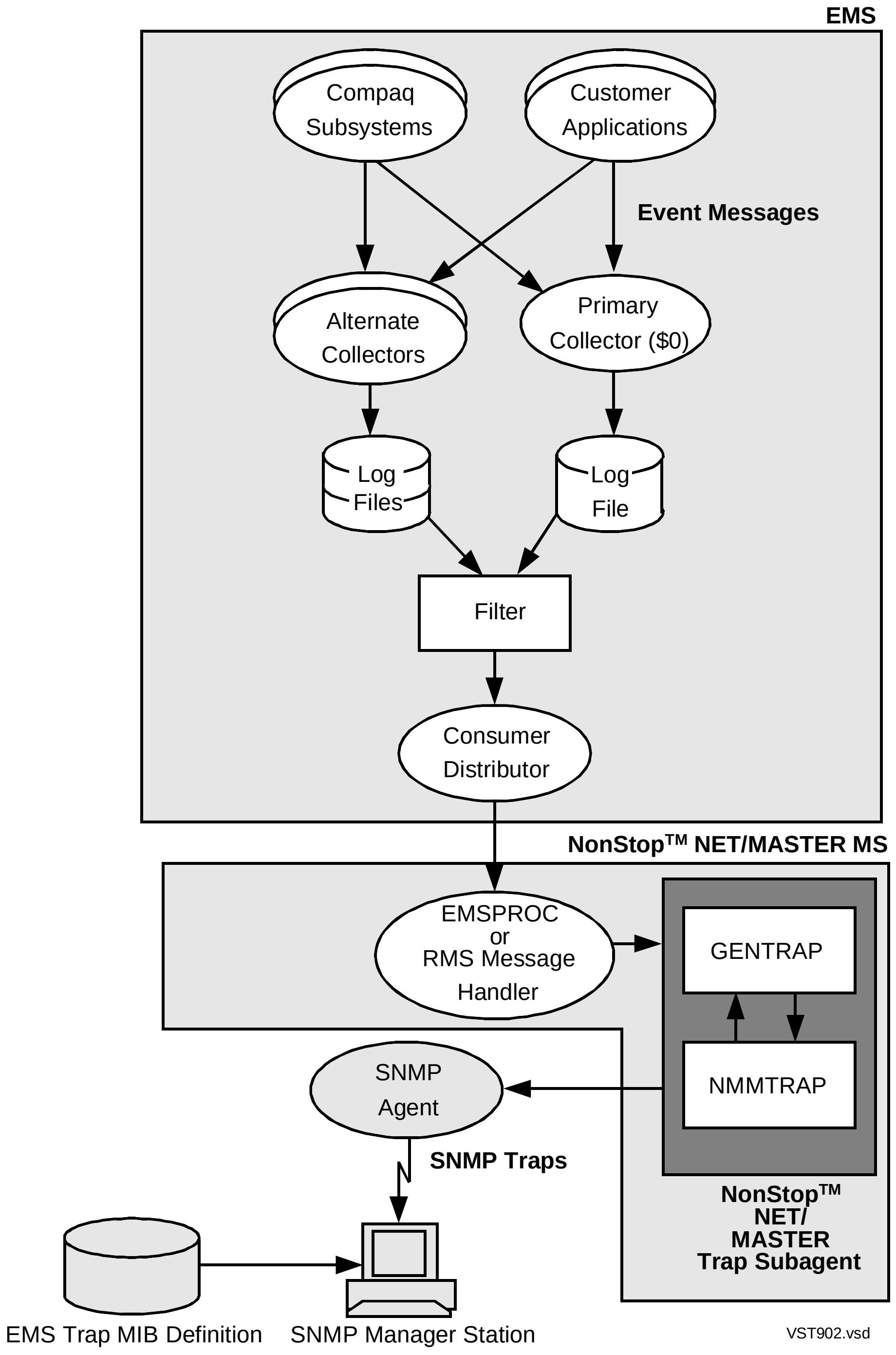
This subsection describes how to install the NonStop NET/MASTER Trap Subagent.

Dependencies

The following products must be configured and running for the NonStop NET/MASTER Subagent to operate properly:

- SNMP agent (D21 or later).
- EMS (D21 or later). The ASN.1 source code file (EMSMIBA) containing the EMS Trap MIB definition is shipped with EMS.
- NonStop NET/MASTER (C30 or later). These NonStop NET/MASTER Trap Subagent files are distributed with NonStop NET/MASTER:
    - GENTRAP: NCL procedure for translating EMS events into SNMP traps.
    - NMMTRAP: NCL library that handles communications between the subagent and the SNMP agent.
    - TESTSEND: NCL procedure that generates an example trap.
- At least one SNMP manager communicating with the SNMP agent that is capable of displaying values for the variable bindings of enterprise-specific traps.

Figure 10-2. NonStop NET/MASTER Trap Subagent Components



Installation Steps

Run the Distributed Systems Management/Software Configuration Manager (DSM/SCM) or the Install program to install EMS, NonStop NET/MASTER MS, and RMS. Refer to the *DSM/SCM User’s Guide* or the *INSTALL User’s Guide* for complete software installation information.

The NonStop NET/MASTER Trap Subagent is installed automatically when you install NonStop NET/MASTER MS. The subagent files are installed in $*dsv*.ZNNMNDS. If you will be modifying GENTRAP, first copy it to your ZNNMNCS subvolume and modify the ZSNMMCS file. The distributed file will still be available, but NonStop NET/MASTER MS will execute your custom GENTRAP procedure.

Next load the ASN.1 source code for the EMS Trap MIB, installed in a file named EMSMIBA and shown in Example 10-1, onto any SNMP manager you want to receive traps from the subagent. Compile the MIB as described in the documentation provided with your SNMP manager. Compiling the MIB allows the SNMP manager to display the names (rather than only numeric object identifiers) of MIB objects.

The subagent can then be configured.

To view an example trap produced by TESTSEND, start the SNMP agent as described in Section 2, Installing and Configuring the SNMP Agent. Then log on to OCS and start TESTSEND:

```
TESTSEND $agent-process
```

agent-process is the name assigned to the SNMP agent process when it was started.

TESTSEND is an NCL file that sends a predefined trap to your SNMP manager. The trap has values for all eight objects defined in the EMS Trap MIB.

Example 10-1. ASN.1 Source Code for EMS Trap MIB (page 1 of 3)

```
--
-- This MIB defines Tandem EMS event messages as traps.
--
--  The MIB source is set for installation on HP OpenView for
--  Windows and Unix.  Installation on other platforms may
require
--  editing:

--
--  Sun NetManager:  comment out entire IMPORTS section;
--                   uncomment iso, dod, etc.
--

TANDEM-EMS-MIB DEFINITIONS ::= BEGIN

   IMPORTS
     enterprises FROM RFC1155-SMI;
--   DisplayString, OBJECT-TYPE FROM RFC-1212
--   TRAP-TYPE FROM RFC-1215;


--
-- EMS event MIB
--

--  iso             OBJECT IDENTIFIER ::= { 1 }
--  org             OBJECT IDENTIFIER ::= { iso 3 }
--  dod             OBJECT IDENTIFIER ::= { org 6 }
--  internet        OBJECT IDENTIFIER ::= { dod 1 }
--  directory       OBJECT IDENTIFIER ::= { internet 1 }
--  mgmt            OBJECT IDENTIFIER ::= { internet 2 }
--  experimental    OBJECT IDENTIFIER ::= { internet 3 }
--  private         OBJECT IDENTIFIER ::= { internet 4 }
--  enterprises     OBJECT IDENTIFIER ::= { private 1 }


  tandem           OBJECT IDENTIFIER ::= { enterprises 169 }
  nonstopsystems   OBJECT IDENTIFIER ::= { tandem 3 }
  ems              OBJECT IDENTIFIER ::= { nonstopsystems 12 }
  snmp             OBJECT IDENTIFIER ::= { nonstopsystems 155 }
  snmpagent        OBJECT IDENTIFIER ::= { snmp 1 }

--
```

Example 10-1. ASN.1 Source Code for EMS Trap MIB (page 2 of 3)

```
-- items in each event
--


  TrapSSIDtype             ::= DisplayString (SIZE (5..24))
  TrapEventNumbertype      ::= DisplayString (SIZE (1..6))
  TrapContentStandardtype ::= DisplayString (SIZE (1..24))
  TrapSubjecttype          ::= DisplayString (SIZE (1..50))
  TrapProcesstype          ::= DisplayString (SIZE (1..50))
  TrapGenerationtimetype  ::= DisplayString (SIZE (19))
  TrapCriticaltype         ::= DisplayString (SIZE (1..14))
  TrapTexttype             ::= DisplayString (SIZE (1..1000))



--
-- item definitions
--

  trapSSID OBJECT-TYPE
      SYNTAX   TrapSSIDtype                --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "subsystem ID that generated the EMS event"
      ::= { ems 1}

  trapEventNumber  OBJECT-TYPE
      SYNTAX   TrapEventNumbertype         --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "Event number of event"
      ::= { ems 2}

  trapContentStandard OBJECT-TYPE
      SYNTAX   TrapContentStandardtype     --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION "standard content of event"
      ::= { ems 3}

  trapSubject OBJECT-TYPE
      SYNTAX   TrapSubjecttype             --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The value of the first subject token in
                    the event "
      ::= { ems 4}
```

Example 10-1. ASN.1 Source Code for EMS Trap MIB (page 3 of 3)

```
  trapProcess OBJECT-TYPE
      SYNTAX   TrapProcesstype             --
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The process name that generated the event "
      ::= { ems 5}

  trapGenerationtime OBJECT-TYPE
      SYNTAX   TrapGenerationtimetype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " GMT the event was generated, rendered
                    yyyy/mm/dd hh:mm:ss.  Timestamp is GMT. "
      ::= { ems 6}

  trapCritical OBJECT-TYPE                 --
      SYNTAX   TrapCriticaltype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " zems-tkn-critical == zspi-val-false ->
                    then '(not critical)'
                    else '(CRITICAL)    ' "
      ::= { ems 7}

  trapText OBJECT-TYPE                     --
      SYNTAX   TrapTexttype
      ACCESS   not-accessible
      STATUS   mandatory
      DESCRIPTION " The text expansion of the EMS event "
      ::= { ems 8}


emsMessage TRAP-TYPE
  ENTERPRISE      snmpagent
  VARIABLES       { trapSSID,
                  trapEventNumber,
                  trapContentStandard,
                  trapSubject,
                  trapProcess,
                  trapGenerationtime,
                  trapCritical,
                  trapText }
  DESCRIPTION "A generic format for an EMS event."
::= 0

END -- TANDEM-EMS-MIB
```

Configuration

You might need to perform one or both of these operations before using the subagent:

- Modify GENTRAP to change the SNMP trap definition or the way the subagent communicates with the SNMP agent.
- Configure trap connections.

This subsection describes these operations.

Modifying GENTRAP

When first installed, this file looks like Example 10-2. The statements highlighted in bold type can be modified to change the trap definition or subagent-SNMP agent communication behavior.

Changing the Trap Definition

To define a trap that differs from the one defined by the EMS Trap MIB, you need to create a MIB definition in ASN.1 and modify GENTRAP to correspond. The trap built by GENTRAP contains items corresponding to the objects in the EMS Trap MIB, described later in this section. Trap object definitions in GENTRAP are shown in bold italic type in Example 10-2.

Changing Subagent-SNMP Agent Communication

Three procedures in NMMTRAP are called from GENTRAP to handle subagent-SNMP agent communications:

- The **openagent** function initiates communication with the SNMP agent.
- The **sendtrap** function builds and sends a trap to the SNMP agent.
- The **closeagent** function terminates communication with the SNMP agent.

This subsection briefly describes these functions and the way GENTRAP uses them initially. The function calls in GENTRAP are highlighted in bold type in Example 10-2.

Example 10-2. Initial Contents of GENTRAP (page 1 of 2)

```
gentrap: procedure nofold

/*
procedure:  gentrap

synopsis:   gentrap <snmp-agent-processname>

description:

    This procedure is intended to be invoked from RMS.  GENTRAP
    assembles an SNMP trap from the EMS event and writes it to
    the #TRAP interface of the specified SNMP agent.

    The agent is specified as &1 on the "NCL Proc Name" field of
    page 5 of RMS : System Action screen.

    This procedure opens and closes the agent each time it is called.
    To improve efficiency, the procedure could open the agent and set
    a global flag first time it is called, then check the flag each
    time to decide whether to open the agent or not.

    This procedure assumes that the NCL variables for the event exist
    and are set correctly.  Do not run this procedure from an
    environment that does not have the event (OCS, for example).
*/

%%include nnmtrap

  &err = openagent(&1, myagt);

  if &err \= 0 then do
    say "error "&err" from openagent call.  &sys.file.rc = ",
      &sys.file.rc
    say "  &sys.file.error = "&sys.file.error
    say "  &sysmsg = "&sysmsg
    flush;
  end;

  control shrvars=&trap.*

  &trap.oid.1 = 1.3.6.1.4.1.169.3.12.1 ; /* SSID */
  &trap.value.1 = &zzzmssid;

  &trap.oid.2 = 1.3.6.1.4.1.169.3.12.2 ; /* event number */
  &trap.value.2 = d2c(&$ems.spi.tandem.zems_tkn_eventnumber) ;
```

Example 10-2. Initial Contents of GENTRAP (page 2 of 2)

```
  &trap.oid.3 = 1.3.6.1.4.1.169.3.12.3 ; /* standard content */
  &trap.value.3 = d2c(&$ems.tandem.spi.zems_tkn_content_standard) ;

  &trap.oid.4 = 1.3.6.1.4.1.169.3.12.4 ; /* value of subject token */
  &trap.value.4 = &zzzmsubject;

  &trap.oid.5 = 1.3.6.1.4.1.169.3.12.5 ; /* generating process */
  &trap.value.5 = &$ems.tandem.spi.zems_tkn_zems_tkn_proc_desc;

  &trap.oid.6 = 1.3.6.1.4.1.169.3.12.6 ; /* date & time */
  &trap.value.6 = &$ems.tandem.spi.zems_tkn_gentime ;
  /*              YYYY/MM/DD         HH:MM:SS.mmmmmm */

  &trap.oid.7 = 1.3.6.1.4.1.169.3.12.7 ; /* emphasis token */
  if &$ems.tandem.spi.zems_tkn_emphasis \= Y then
    &trap.value.7 = "(not critical)"
  else
    &trap.value.7 = "(CRITICAL)" ;


  &trap.oid.8 = 1.3.6.1.4.1.169.3.12.8 ; /* event text */
  &trap.value.8 = &zzzmsgtext ;

  &err = sendtrap(myagt, 6, 0, 8)
                  /*zsmp-val-trap-enterpriseSpfc = 6
                    trap type                    = 0
                    OID count                    = 8   */

  if &err \= 0 then do
    say "error "&err" from sendtrap procedure.  &sys.file.rc = ",
      &sys.file.rc
    say "  &sys.file.error = "&sys.file.error
    say "  &sysmsg = "&sysmsg
  end;

  &err = closeagent(myagt);

  if &err \= 0 then do
    say "error "&err" from closeagent procedure.  &sys.file.rc = ",
      &sys.file.rc
    say "  &sys.file.error = "&sys.file.error
    say "  &sysmsg = "&sysmsg
  end;

end gentrap
```

The **openagent function** sets up a user database (UDB) for the agent process. It accepts two arguments and returns an integer describing the outcome of the function call. The openagent function is invoked as follows:

```
return-code = openagent (SNMP-agent-process, file-id)
```

SNMP-agent-process

is the name assigned to the agent process when it was started. GENTRAP passes the agent process name as a parameter (&1), assigned a value when GENTRAP starts.

file-id

is the file identifier you associate with the open operation. GENTRAP assigns the name myagt.

return-code

is an integer describing the outcome of the function call. GENTRAP uses a variable named &err to hold this value. The openagent return codes, their meanings, and the actions taken by GENTRAP when they occur are described in Messages on page 10-18.

The **sendtrap function** assembles the trap and writes it to the SNMP agent. In this function, all trap MIB values are assumed to be strings. The function accepts four arguments and returns an integer describing the outcome of the function call. The sendtrap function is invoked as follows:

```
return-code = sendtrap (file-id, generic-trap, specific-trap,
   OID-count)
```

file-id

is the file identifier you associated with the openagent operation. GENTRAP assigns the name myagt.

generic-trap

is an integer identifying one of the standard trap types. Valid values are:

2 (linkDown)
3 (linkUp)
4 (authenticationFailure)
5 (egpNeighborLoss)
6 (enterpriseSpecific)

GENTRAP passes the value 6 to identify the trap as an enterprise-specific trap.

specific-trap

is a code indicating more specifically the nature of the trap. GENTRAP assigns the value 0.

OID-count

is an integer specifying the number of object identifiers to append. GENTRAP assigns the value 8 because there are 8 objects in the EMS Trap MIB. The size of the (non-BER-encoded) trap is limited to 1500 bytes.

return-code

is an integer describing the outcome of the function call. GENTRAP uses a variable named &err to hold this value. The sendtrap return codes, their meanings, and the actions taken by GENTRAP when they occur are described in Messages on page 10-18.

The **closeagent function** terminates communication with the SNMP agent. It closes an open instance created by openagent. This function accepts one argument and returns an integer describing the outcome of the function call. The closeagent function is invoked as follows:

```
return-code = closeagent (file-id)
```

file-id

is the file identifier you associated with the openagent operation. GENTRAP assigns the name myagt.

return-code

is an integer describing the outcome of the function call. GENTRAP uses a variable named &err to hold this value. The closeagent return codes, their meanings, and the actions taken by GENTRAP when they occur are described in Messages on page 10-18.

Trap Connections

Traps from the NonStop NET/MASTER Trap Subagent are routed to SNMP managers identified in the SNMP agent's TRAPDEST objects. Refer to Configuring Trap Destinations on page 2-38 for more information on this subject.

Starting and Stopping the Subagent

To start the subagent, first start the SNMP agent as described in Section 2, Installing and Configuring the SNMP Agent. Then invoke GENTRAP from EMSPROC or an RMS message handler:

```
GENTRAP $agent-process
```

agent-process

is the name assigned to the agent process when it was started.

GENTRAP initiates communication with the SNMP agent by calling the openagent function in **NMMTRAP**. After sending a trap, GENTRAP terminates communication with the SNMP agent by calling the closeagent function in NMMTRAP.

The Trap PDU

The trap PDU contains these fields:

enterprise	The object identifier for the SNMP agent, indicating the origin of the trap: 1.3.6.1.4.1.169.3.155.1.
agent-address	The Internet address of the system on which the SNMP agent forwarding the trap is installed.
generic-trap	A 16-bit number set to 6 to signify that the trap is enterpriseSpecific.
specific-trap	A 16-bit number set to 0.
time-stamp	A 32-bit number indicating how much time has passed since the SNMP agent was last started. The subagent inserts 0, and the SNMP agent inserts the value of sysUpTime, defined in Section 8, TCP/IP Subagent.
variable-bindings	The objects defined in the EMS Trap MIB, described in the following subsection.

The EMS Trap MIB

The EMS Trap MIB is a collection of scalar objects, identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          private (4)
            enterprises (1)
              tandem (169)
                nonstopsystems (3)
                  ems (12)
                    trapSSID (1) √
                    trapEventNumber (2) √
                    trapContentStandard (3) √
                    trapSubject (4) √
                    trapProcess (5) √
                    trapGenerationtime (6) √
                    trapCritical (7) √
                    trapText (8) √
```

Table 10-1 describes the objects in the EMS Trap MIB.

Table 10-1. Objects in the EMS Trap MIB (page 1 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapSSID 1.3.6.1.4.1.169.3.12.1 not-accessible DisplayString (SIZE (5..24))	The identifier of the subsystem or application that generated the event.	*owner.ss.version*, *owner* is the organization providing the software that generated the trap. For HP subsystems, the owner value is TANDEM. *ss* is the subsystem name or number. Refer to the *Operator Messages Manua* to interpret HP values. *version* identifies the software release version of the subsystem. For HP subsystems, this value matches the three-character release ID in the product release version (for example, D31).	ZSPI-TKN-SSID token value.
trapEventNumber 1.3.6.1.4.1.169.3.12.2 not-accessible DisplayString (SIZE (1..6))	An integer that uniquely identifies an event from a subsystem or application.	Integer without leading 0s.	ZSPI-TKN-EVENTNUMBER value.

Table 10-1. Objects in the EMS Trap MIB (page 2 of 4)

Object and Attributes	**Definition**	**Format of Value**	**Derivation of Value**
trapContentStandard 1.3.6.1.4.1.169.3.12.3) not-accessible DisplayString (SIZE (1..24))	An indication of whether the event is a standard event and, if so, what type of standard event. Programmers are encouraged to use standard events to improve the consistency among messages in certain categories and to promote system management automation.	One of these values: not-specified (Event is not a standard event.) transient-fault (An error has been corrected automatically.) object-unavailable (An object or the services it depends on are not supplying the service they were designed to provide.) object-available (An object and all services it depends on are supplying the service they were designed to provide.) other-state-change (An object needs an operator to change or notice its state.) attention-needed (An application cannot continue until an operator takes some action.) attention-completed (An action described by an earlier event having the value attention-needed has been completed.) usage-threshold (The usage level of an object or resource exceeds the configured level.) unknown (Returned for any event not associated with the preceding values.)	The value of trapContentStandard corresponds to ZEMS-TKN-CONTENT-STANDARD token values as follows: not-specified (ZEMS-VAL-NULL) transient-fault (ZEMS-VAL-TRANSIENT-FAULT) object-unavailable (ZEMS-VAL-OBJECT-UNAVAILABLE) object-available (ZEMS-VAL-OBJECT-AVAILABLE) other-state-change (ZEMS-VAL-OTHER-STATE-CHANGE) attention-needed (ZEMS-VAL- ATTN-NEEDED) attention-completed (ZEMS-VAL- ATTN-COMPLETED) usage-threshold (ZEMS-VAL-USAGE-THRESHOLD)

Table 10-1. Objects in the EMS Trap MIB (page 3 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapSubject 1.3.6.1.4.1.169.3.12.4) not-accessible DisplayString (SIZE (1..50))	The hardware or software component most directly involved in the event.	All values except floating-point values and values stored as structured tokens can be formatted. A question mark (?) appears when the value of the subject cannot be formatted.	The value of the first event subject token.
trapProcess 1.3.6.1.4.1.169.3.12.5 not-accessible DisplayString (SIZE (1..50))	The process descriptor of the process reporting the event.	For a named process: *`node-name`*.*`process-name`*.*`sequence-number`* For an unnamed process: *`node-name.`* $*`cpu:pin`*:*`sequence-number`* *`node-name`* identifies the system on which the process resides. *`process-name`* is the name assigned at startup time, of the form $*`name`*, where *`name`* has from one through five characters. *`sequence-number`* is an integer used to distinguish among different instances of a process. *`cpu:pin`* is a process ID consisting of a processor module number (*`cpu`*) and a processor identification number (*`pin`*).	ZEMS-TKN-PROC-DESC token value.
trapGenerationtime 1.3.6.1.4.1.169.3.12.6 not-accessible DisplayString (SIZE 19)	The time when the event was generated.	YYYY/MM/DD HH:MM:SS.	ZEMS-TKN-GENTIME token value.

Table 10-1. Objects in the EMS Trap MIB (page 4 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
trapCritical 1.3.6.1.4.1.169.3.12.7 not-accessible DisplayString (SIZE (1..14))	An indication of whether the event is critical or noncritical. Critical events have consequences that might be severe, as when they indicate potential or actual loss of data, loss of a major subsystem function, loss of fault-tolerance capability, or loss of subsystem integrity.	One of these enumerated values: CRITICAL not critical	ZEMS-TKN-EMPHASIS token value. If the value of this token is false (0), the value passed is (not critical). If the value is non-zero, the value passed is (CRITICAL).
trapText 1.3.6.1.4.1.169.3.12.8 not-accessible DisplayString (SIZE (1..1000))	Text describing the event.	ASCII text.	EMSTEXT procedure call.

Messages

GENTRAP sends messages to an Operator Control Services (OCS) window when the openagent, sendtrap, or close agent functions fail.

openagent Failures

Table 10-2 describes the actions GENTRAP takes when openagent fails.

Table 10-2. openagent Return Codes and GENTRAP Actions

Return Code	Meaning	GENTRAP Actions
0	The function completed successfully.	Processing continues.
1	A UDB problem occurred.	Processing stops after a message is displayed at an OCS window.
2	Communication with the agent process could not be started.	Processing stops after a message is displayed at an OCS window.

The format of the message is as follows:

```
error return-code from openagent call.
&sys.file.rc = &sys.file.rc-value
&sys.file.error = &sys.file.error-value
&sysmsg = &sysmsg-value
```

return-code

has the value 1 or 2, as described in Table 10-2.

&sys.file.rc-value

contains an integer if the FILE OPEN verb in the openagent function failed:

12 (UDB is inaccessible.)
16 (UDB identifier is unavailable or not found. See &sysmsg for more information.)

&sys.file.error-value

is the Guardian file error number corresponding to the value of &sys.file.rc.

&sysmsg

is a text message.

sendtrap Failures

Table 10-3 describes the actions GENTRAP takes when sendtrap fails.

Table 10-3. sendtrap Return Codes and GENTRAP Actions

Return Code	Meaning	GENTRAP Actions
0	The function completed successfully.	Processing continues.
1	The OID count is less than 0.	Processing continues after a message is displayed at an OCS window.
2	The total trap size exceeds 1500 bytes.	Processing continues after a message is displayed at an OCS window.
3	Communication with the SNMP agent failed.	Processing continues after a message is displayed at an OCS window.

The format of the message is as follows:

```
error return-code from sendtrap procedure.
&sys.file.rc = &sys.file.rc-value
&sys.file.error = &sys.file.error-value
&sysmsg = &sysmsg-value
```

return-code

has the value 1 through 3, as described in Table 10-3.

&sys.file.rc-value

contains an integer if the FILE PUTGET verb in the sendtrap function failed:

8 (Error occurred. See &sys.file.error value for more information.)
16 (Error occurred. &sysmsg might contain more information.)

&sys.file.error-value

is the Guardian error number corresponding to the value of &sys.file.rc.

&sysmsg

is a text message.

closeagent Failures

Table 10-4 describes the actions GENTRAP takes when openagent fails.

Table 10-4. closeagent Return Codes and GENTRAP Actions

Return Code	Meaning	GENTRAP Actions
0	The function completed successfully.	GENTRAP processing is complete.
1	Communication with the SNMP agent could not be terminated.	Processing stops after a message is displayed at an OCS window.

The format of the message is as follows:

```
error return-code from closeagent procedure.
&sys.file.rc = &sys.file.rc-value
&sys.file.error = &sys.file.error-value
&sysmsg = &sysmsg-value
```

return-code

has the value 1, as described in Table 10-4.

&sys.file.rc-value

contains an integer if the FILE CLOSE verb in the closeagent function failed:

16 (UBD closed.)

&sys.file.error-value

is the Guardian error number corresponding to the value of &sys.file.rc.

&sysmsg

is a text message.

11 Host Resources Subagent

The Host Resources Subagent supports all the mandatory and two of the optional groups of the standard Host Resources MIB. It also implements several groups defined by HP that maintain information about hardware resources that might require operator attention. These MIB extensions also let you manage the subagent and control the MIB values it maintains. This section describes the Host Resources Subagent and its MIB.

Architectural Overview

The Host Resources Subagent facilitates management of system resources in several ways:

- Its MIB implements five of the MIB groups defined in RFC 1514, *Host Resources MIB*. These groups describe hardware and software resources on the NonStop system where the subagent is installed.
- Several groups defined by HP extend the standard Host Resources MIB definition to support proactive hardware management.
- Other additional groups defined by HP support management of the subagent and control over the objects it maintains in its MIB.

The Host Resources Subagent can be configured to monitor both NonStop Kernel Guardian and Open System Services (OSS) personalities.

The remainder of this subsection provides more information about the features of the Host Resources Subagent.

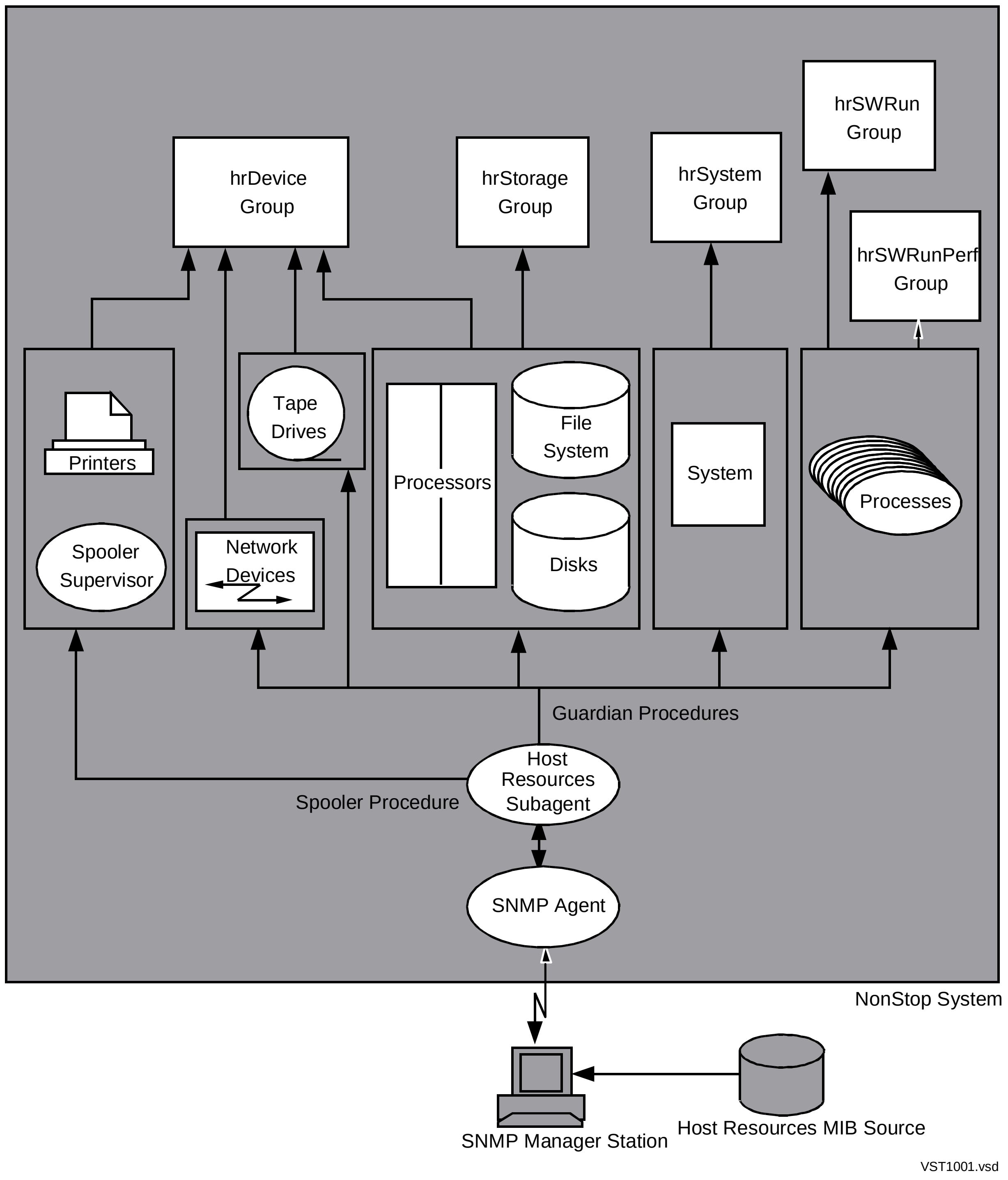
Standard MIB Groups

The Host Resources Subagent's MIB supports the following groups defined in RFC 1514, *Host Resources MIB*:

hrSystem group	This mandatory group describes general host characteristics, such as the number of current users.
hrStorage group	This mandatory group describes storage media attached to the host. Storage media include physical memory, virtual memory, fixed or removable disks, and so on.
hrDevice group	This mandatory group describes devices ranging from printers to tape drives.
hrSWRun group	This optional group describes the processes running on the system.
hrSWRunPerf group	This optional group describes the resource consumption of the processes in the hrSWRun group.

As Figure 11-1 indicates, the subagent uses Guardian and Spooler procedure calls to obtain information for these MIB groups.

Figure 11-1. Derivation of Information for RFC 1514 Support



The standard MIB groups, identified by a check mark in the following list, are members of MIB-II:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                mib-2 (1)
                    system (1)
                    interfaces (2)
                    at (3)
                    icmp (5)
                    ip (4)
                    tcp (6)
                    udp (7)
                    egp (8)
                    cmot (9)
                    transmission (10)
                    snmp (11)
                    host (25)
                        hrSystem (1) √
                        hrStorage (2) √
                        hrDevice (3) √
                        hrSWRun (4) √
                        hrSWRunPerf (5) √
                        hrSWInstalled (6)
```

The subagent does not support the optional hrSWInstalled group.

MIB Extensions

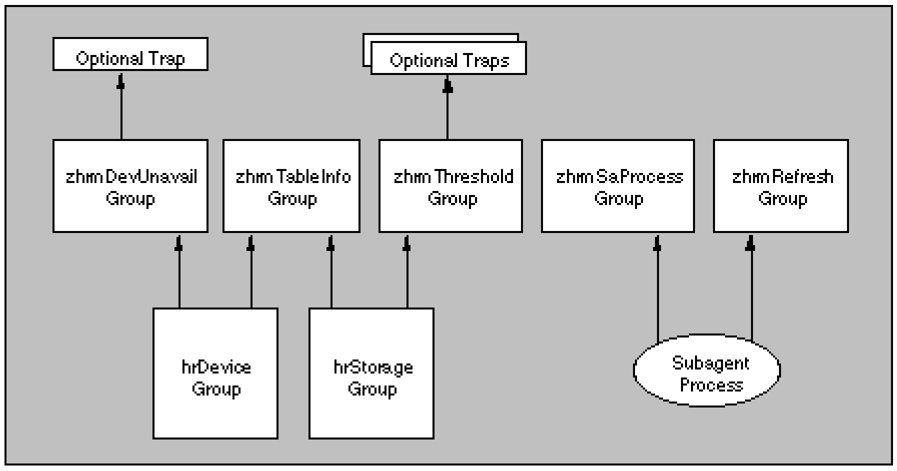
The Host Resources Subagent's MIB also supports the following groups, defined by HP:

zhrmTableInfo group	This group describes the status of and operations performed on objects in the hrStorage and hrDevice groups.
zhrmThreshold group	This group describes RAM and disk devices in the hrStorage group whose usage has reached a critically high level.

zhrmDevUnavail group	This group describes hrDevice group devices that are in a state that may require operator attention.
zhrmSaProcess group	This group provides information on the subagent process. This group also lets you change the priority of the subagent process, cause the subagent's backup process to take over and a new backup process to be created, control which NonStop Kernel personality (Guardian or OSS) is to be monitored, and control whether the subagent maintains hrSWRun and hrSWRunPerf group values.
zhrmRefresh group	This group provides information about MIB value updates by the subagent and lets you request that the subagent refresh MIB values on demand.

Figure Figure 11-2 illustrates the source of some of the information maintained in these groups. Note that the subagent can generate traps to signal certain noteworthy conditions; you can enable or disable trap generation by setting the value of objects in the zhrmDevUnavail or zhrmThreshold groups.

Figure 11-2. Derivation of Information for RFC 1514 Extensions



The group and trap definitions, identified by a check mark in the following list, reside in the zhrm subtree within the subtree registered to HP:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                zhrm (180)
                                    zhrmTableInfo (1) √
                                    zhrmThreshold (2) √
                                    zhrmDevUnavail (3) √
                                    zhrmSaProcess (4) √
                                    zhrmRefresh (5) √
                                    zhrmTraps (7) √
```

Proactive Hardware Management

You can configure the Host Resources Subagent to generate traps that describe situations that might require system operator intervention. For example:

- A particular device is no longer running.
- A particular disk is demonstrating critically high usage.
- A particular CPU is demonstrating critically high RAM usage.

Entries maintained in several MIB groups also support proactive management of hardware resources:

- The zhrmThreshold group provides information on device utilization that exceeds a settable threshold.
- The zhrmDevUnavail group provides device statistics and highlights unusual device states.

Refreshing MIB Values

MIB values are always refreshed when a subagent process starts. After startup, to control its CPU usage, the subagent uses two configurable timers for refreshing MIB values:

- A stable object timer controls the rate at which hardware configuration information is refreshed. Configuration information is relatively static, changing mainly when devices are added to or removed from the system.

- A dynamic object timer controls the rate at which hardware status and statistical information is refreshed. Status information describes object states, such as whether a device is up or down. Statistical information includes information such as the current number of system users.

These two timers also control the rate at which software data is refreshed. Whenever one of the timers runs down, values in the hrSWRun and hrSWRunPerf groups are refreshed.

You set refresh timer values when you start the subagent, as describedStarting and Stopping the Subagent on page 11-12. After the subagent is running, you can refresh values on demand by setting the value of objects in the zhrmRefresh group from an SNMP manager.

Although these counters control the rate at which most MIB values are refreshed, the subagent also:

- Uses an internal timer to refresh the average 60-second CPU load every 60 seconds
- Updates the values of hrSystemDate and hrSystemUpTime in the hrSystem group whenever these objects are accessed by a manager

MIB Value Management

To minimize the size of the subagent's MIB, reduce the overhead of refreshing data, and highlight certain critical resources, you can:

- Limit entries in the hrSWRun and hrSWRunPerf groups to specific processes
- Limit entries in the hrDevice group to specific devices
- Suppress creation and maintenance of values in the hrSWRun and hrSWRunPerf groups

The values maintained can be both preconfigured and dynamically altered. This ability to maintain MIB values for a subset of hardware and software resources is particularly useful for large hosts with many resources. It is also useful when you need to focus on particularly critical resources.

Refer to Starting and Stopping the Subagent on page 11-12 for more information on this topic.

Obtaining Information About the Subagent

Objects in the zhrmSaProcess group provide information about the subagent process. Examples of information available are the name of the subagent process and statistics about Get, GetNext, and Set requests the subagent has handled since starting.

Initiating Backup Process Takeover

The Host Resources Subagent can be run as a process pair to achieve a basic fault-tolerant level. The support includes a persistent subagent process and checkpointing of the startup parameters.

To run the subagent as a process pair and have the backup process take over if the primary process fails, use the -b startup parameter:

```
RUN HMSAX /NAME $HMSA, NOWAIT/ -b
```

You can specify a backup cpu number during starup to create a backup process on the specified cpu. To start the backup process on a particular cpu, invoke HRSA with -b [backup_cpu_num] option.

If the specified backup cpu is not up and running, backup process will start on any available cpu.

If the specified backup_cpu_num is not a valid cpu number, then an EMS event message will be generated and HRSA process will stop.

You can force the backup process to take over and a new backup process to be created by setting the value of an object in the zhrmSaProcess group.

Monitoring the Open System Services (OSS) File System

To start the Host Resources Subagent to an HP NonStop Kernel Open System Services (OSS) environment, use the -u startup parameter:

```
RUN HMSAX /NAME $HRMIB, NOWAIT/ -u
```

To monitor both Guardian and OSS environments, enter:

```
RUN HMSAX /NAME $HRMIB, NOWAIT/ -t -u
```

If you specifiy neither the -u or -t startup parameters, monitoring of the Guardian personality is enabled, and monitoring of the OSS personality is disabled.

In the NonStop Kernel Guardian environment, there is a one-to-one mapping between a file system (fileset) and a physical disk partition.

In the OSS environment, a fileset can span multiple physical disks. To provide monitorability of OSS filesets, the Host Resources MIB gives information about the filesets in the hrFSTable and in the hrDiskStorageTable. Each disk in an OSS file system's disk pool is treated as an HRMIB partition. Detailed information for each Guardian disk in the pool is available in the hrPartitionTable, hrDeviceTable, and hrStorageTable.

The file-system size is calculated as the sum of the component disk sizes. If a disk is in more than one file system, the disk size will be accounted for twice (once in each file system).

Related Documents

The following documents provide information related to the Host Resources Subagent:

- The *EMS Manual* describes the Event Management Service (EMS). EMS is a collection of processes, tools, and interfaces that provide event-message collection and distribution in the Distributed Systems Management environment. The subagent generates EMS event messages when noteworthy situations arise.
- The *Guardian Procedure Calls Reference Manual* describes the Guardian procedure calls the subagent uses to obtain values for most MIB objects.
- RFC 1514, *Host Resources MIB*, specifies a MIB for use when managing host systems. This MIB is referred to as the Host Resources MIB.
- The *Spooler Programmer's Guide* describes Spooler procedure SPOOLERSTATUS2, which the subagent uses to obtain values for MIB objects that are printers.

RFC Compliance

The Host Resources Subagent supports objects defined in the mandatory groups and two of the optional groups of the Host Resources MIB defined in RFC 1514:

hrSystem group
hrStorage group
hrDevice group
hrSWRun group
hrSWRunPerf group

For details about how support for each group complies with RFC 1514:

- Table 11-3 describes compliance of hrSystem group objects.
- Table 11-5 describes compliance of hrStorage group objects.
- Table 11-7 describes compliance of hrDevice group objects.
- Table 11-9 describes compliance of hrSWRun group objects.
- Table 11-10 describes compliance of hrSWRunPerf group objects.

Installation

This subsection describes how to install the Host Resources Subagent.

Dependencies

The following products must be configured and running for the Host Resources Subagent to operate properly:

- SNMP agent (D23 or later)
- EMS (D21 or later)

- DSM Template Services (D21 or later)
- At least one SNMP manager that can send requests to the SNMP agent for processing by the Host Resources Subagent
- If the NonStop Kernel OSS environment is to be monitored, the OSS Monitor (OSSMON) must be running.

Installation Steps

Run the Distributed Systems Management /Software Configuration Manager (DSM/SCM) to install EMS, DSM Template Services, and the Host Resources Subagent. Refer to the *DSM/SCM User's Guide* for complete software installation information.

Ensure that the EMS file named ZHRMTMPL is installed in the system template file. When you use the SYSGEN utility to generate your system, this file is automatically installed in the system template file. Otherwise, you must manually install it. Refer to the *EMS Manual* for more information.

Load the ASN.1 source code for the Host Resources Subagent's MIB onto any SNMP manager you want to communicate with the subagent. The MIB definitions are contained in three files:

- Standard MIB definitions. Two files are provided:
    - The **RFC1514** file contains definitions identical to those in RFC 1514, with two exceptions. InternationalDisplayString objects and hrPartitionID have been redefined as DisplayString objects so that managers can display their values in a readable format instead of hexadecimal format. Additionally, the two Boolean objects have been redefined as INTEGER objects with values of TRUE and FALSE, displayable by managers.
    - The **RFC1514A** file contains the same definitions as RFC 1514 with several exceptions. DESCRIPTION clauses have been modified to provide information specific to HP systems. Several read-write objects have been redefined as read-only objects to reflect the Host Resource Subagent's implementation. And several comments have been added to provide a summary of conformance with RFC 1514.
- MIB extensions. The **ZHRMMIB** file contains definitions for groups and traps defined by HP.

You can load definitions for both a standard MIB and the MIB extensions, only the standard definitions, or only the extended definitions.

The Host Resources Subagent provides information for resources on the NonStop system where it is installed. The subagent can communicate with a SNMP agent on either the same system or a remote system, but only one subagent process can communicate with any particular SNMP agent process.

Configuration

You can configure the behavior of the Host Resources Subagent at startup time or from an SNMP manager after startup. Table 11-1 summarizes the available options.

All subagent attributes that can be controlled by setting MIB object values are reinitialized when the subagent is restarted to default values or values specified in startup parameters.

Refer to Starting and Stopping the Subagent on page 11-12 for information on configuration using RUN startup parameters. Refer to subsequent subsections on individual MIB groups for information on settable MIB objects.

Table 11-1. Host Resources Subagent Configuration Options (page 1 of 3)

Subagent Attribute	**RUN Startup Para-meter**	**Settable MIB Object Name (Group Name)**	**Default Behavior**
SNMP agent process name	`-a`		$ZSNMP on the same system as the subagent specified at startup.
Subagent process			
Priority	`-n`	zhrmSaPri (zhrmSaProcess group)	Priority is 145.
Backup process creation	`-b`		No backup process.
Backup process takeover		zhrmSwitchToBackupNow (zhrmSaProcess group)	Backup process takes over if primary process fails.
MIB value refresh rate			
Hardware configuration objects	`-s`	zhrmStableRefreshNow (zhrmRefresh group)	Values refreshed every 24 hours.
Hardware status and statistics objects	`-d`	zhrmDynamRefreshNow (zhrm Refresh group)	Values refreshed every 10 minutes.
Software objects	`-s` `-d`	zhrmStableRefreshNow (zhrmRefresh group) zhrmDynamRefreshNow (zhrm Refresh group)	Values refreshed whenever hardware object values are refreshed.
MIB values maintained			

Table 11-1. Host Resources Subagent Configuration Options (page 2 of 3)

Subagent Attribute	RUN Startup Para-meter	Settable MIB Object Name (Group Name)	Default Behavior
Hardware and software objects	`-e`		Values maintained for all devices and all processes except those associated with products listed under "Starting and Stopping the Subagent."
Software objects	`-w`	zhrmEnableSoftwareGroup (zhrmSaProcess group)	hrSWRun and hrSWRunPerf group values are maintained.
Network interface file	`-f`		IFTBL in the current volume and subvolume of the user who starts the subagent.
Spooler process controlling printers in MIB	`-p`		$SPLS, the default Spooler supervisor process.
Tracing			
To a terminal or Spooler destination	`-o`		No tracing.
To a disk file	`-l`		No tracing.
Trap generation		zhrmDevUnTrapEnable (zhrmDevUnavail group) zhrmThrDiskTrapEnable (zhrmThreshold group) zhrmThrRAMTrapEnable (zhrmThreshold group)	Traps enabled.
Thresholds related to trap generation and zhrmThreshold group MIB objects maintained			
Low threshold	`-i`	zhrmThrDiskLowValue (zhrmThreshold group) zhrmThrRAMLowValue (zhrmThreshold group)	Trap generated when RAM or disk usage reaches 45 percent.
High threshold	`-h`	zhrmThrDiskHighValue (zhrmThreshold group) zhrmThrRAMHighValue (zhrmThreshold group)	Trap generated when RAM or disk usage reaches 90 percent.

Table 11-1. Host Resources Subagent Configuration Options (page 3 of 3)

Subagent Attribute	RUN Startup Para-meter	Settable MIB Object Name (Group Name)	Default Behavior
NonStop Kernel personality to be monitored			
Guardian	`-t`	zhrmExamineGuardian (zhrmSaProcess group)	Guardian file system and processes are monitored.
OSS	`-u`	zhrmExamineOSS (zhrmSaProcess group)	OSS file system and processes are monitored.

Starting and Stopping the Subagent

The SNMP agent must be running before the Host Resources Subagent is started. If the agent is not running, start it with the RUN command:

```
RUN SNMPAGT /NAME $agent-process, NOWAIT/
```

For complete information on starting the SNMP agent, refer to Section 2, Installing and Configuring the SNMP Agent.

Start the subagent by entering the RUN command at a terminal that is running the TACL program. The subagent can be run by any user (you do not need to belong to a super user group), but it must be started as a named process. Assuming the name of the SNMP agent is $ZSNMP, the following is an example of starting the subagent to monitor a NonStop Kernel Guardian environment:

```
RUN HMSAX /NAME $HRMIB, NOWAIT/
```

Note. You could also include the -t option. By default, if neither the -t or -u options are specified, monitoring of the Guardian personality is enabled, and monitoring of the OSS personality is disabled.

The next example starts the subagent monitoring only the OSS personality. The Guardian personality is not monitored.

```
RUN HMSAX /NAME $HRMIB, NOWAIT/ -u
```

To monitor both Guardian and OSS environments:

```
RUN HMSAX /NAME $HRMIB, NOWAIT/ -t -u
```

Syntax of the startup options follows established UNIX conventions.

```
[RUN] [[$volume.]subvolume.]HMSAX
   / NAME $subagent-process [,other-run-option]... /
   [ ? | startup-parameter [startup-parameter]...]
```

volume

identifies the volume on which HMSAX resides. You can omit it if HMSAX resides on your current subvolume. By default, the installation program used to put HMSAX into $SYSTEM.

subvolume

identifies the subvolume on which HMSAX resides. You can omit it if it is named in your TACL #PMSEARCHLIST. By default, the installation program used to put HMSAX into SYS*nn.*

subagent-process

identifies the subagent process. You can specify from one through five alphanumeric characters, but the first character must be alphabetic.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the NOWAIT option so that you can resume TACL operations after subagent startup.

?

displays help information.

startup-parameter

is one of the following parameters that control attributes of the subagent process:

```
-a $agent-process                          |
-b backup-cpu                              |
-d dynamic-object-refresh-rate             |
-e initialization-file                     |
-f iffile                                  |
-h high-value-threshold                    |
-i low-value-threshold                     |
-n process-priority                        |
-p $spooler-process                        |
-s stable-object-refresh-rate              |
-t                                         |
-u                                         |
-w                                         |
-o                                         |
-l log-file
```

-a *$agent-process*

is the name of the SNMP agent process from which you want the subagent to receive SNMP manager requests. If not specified, $ZSNMP on the same system as the subagent you specify is assumed.

-b *backup-cpu*

specifies that the subagent is to run as a process pair. If the primary process fails, the backup process automatically takes over. You can force the backup process to take over and a new backup process to be created by setting the value of zhrmSwitchToBackupNow in the zhrmSaProcess group from an SNMP manager after the subagent is started.

You can specify a backup cpu number during starup to create a backup process on the specified cpu.

If the specified backup-cpu is not up and running, backup process will start on any available cpu.

If the specified backup-cpu is not a valid cpu number, then the following EMS event message will be generated and HRSA process will stop. "Invalid CPU number.Program terminated."

Note. When the subagent switches to a backup process, SNMP managers interacting with the subagent might experience a timeout because the subagent refreshes all its MIB values when switching. Simply resubmit requests after waiting a brief period. Large MIBs might require more than a minute to be refreshed.

-d *dynamic-object-refresh-rate*

specifies, in one-minute intervals, the rate at which the subagent refreshes values describing the status of and statistics about hardware resources for the following MIB objects:

Status objects	**Statistics objects**
hrDeviceStatus	hrSystemUpTime
hrPrinterStatus	hrSystemNumUsers
hrPrinterDetectedErrorState	hrSystemProcesses
	hrMemorySize
	hrStorageSize
	hrStorageUsed
	hrStorageAllocationFailures
	hrDeviceErrors

Status objects	**Statistics objects**
	hrProcessorLoad
	hrDiskStorageCapacity
	hrPartitionSize

The default dynamic refresh rate is every 10 minutes. A value of 0 or less for the dynamic refresh rate suppresses refreshing these values.

The dynamic refresh rate is the slowest rate at which values are refreshed. Because of the internal resource access mechanism, the refresh can occur before the configured time has elapsed.

If a status or statistics object is accessed from an SNMP manager within 30 seconds before the dynamic object timer expires, the subagent postpones refreshing values until another cycle has passed. This minimizes SNMP manager timeouts.

Whenever the values of status and statistics objects are refreshed, the values for MIB objects in the hrSWRun and hrSWRunPerf groups are also refreshed.

You can refresh the values of these hardware and software objects on demand from an SNMP manager after the subagent is running by setting the value of zhrmDynamRefreshNow in the zhrmRefresh group.

Note. While the subagent refreshes status and statistics objects, SNMP managers interacting with the subagent might experience a timeout. Simply resubmit requests after waiting a brief period. Large MIBs might require more than one minute to be refreshed.

`-e` *`initialization-file`*

specifies the name of a file that identifies processes and devices whose values you want represented in the subagent's MIB. If this startup parameter is not used, the subagent looks for a file named HMSAINI in the current volume and subvolume of the user who starts the subagent. If a specified file or HMSAINI cannot be found, values for all devices are maintained, and values for all processes except those associated with the following products are maintained:

BIND	FUP	PRS	TAL
BINSERV	GUIMAIL	PSMAIL	TEDIT
C	IMON	PUP	TELE
CFRONT	Inspect	QBMAKE	Telnet
COBOL	LINE25	REMIND	Telserv
CPREP	MAKE	SCF	TFORM
CSPOOL	M6530	SCOBOL	TGAL
DMON	NETGAL	SeeView	TISERV

DDL	NetView	SENTINEL	TWORK
EDIT	OSIMAGE	SPOOL	VS
EMSDIST	PATHCOM	SPOOLCOM	YOUMAIL
FILTER	PERUSE	SYMSERV	
FTP	PERUSESP	TACL	

Devices excluded are not represented in the hrDevice group. Processes associated with program file names excluded are not represented in the hrSWRun or hrSWRunPerf groups.

Encode entries in HMSAINI as shown in the following syntax diagram, using plus (+) and minus (-) signs to include and exclude devices and processes. MIB values for any devices or processes not excluded in HMSAINI are maintained. Invalid device or program file names or other invalid entries are ignored. After startup, you might want to query the MIB to ensure that values are present for objects you are interested in.

```
[ "["DeviceTable"]"
   [-*] | [-device-name]...
   [+device-name]... ]
[ "["SWRunTable"]"
   [-*] | [-program-file-name]...
   [+program-file-name]... ]
```

`[DeviceTable]`

precedes a list of one or more device name specifications.

`[SWRunTable]`

precedes a list of one or more program file name specifications. The -w startup parameter or a value of swValuesDisabled for zhrmEnableSoftwareGroup in the zhrmSaProcess group overrides specifications in this list.

`-*`

excludes all devices or all program files.

`-device-name` `|` *`-program-file-name`*

excludes the device or program file named.

`+device-name` `|` *`+program-file-name`*

includes the device or program file named.

device-name

identifies a device:

CPU*nn*	identifies a CPU.
$*printer-name*	identifies a printer.
$*volume-name*	identifies a disk.
$*tape-drive-name*	identifies a tape drive.
$*network-device-name*	identifies a network device.

program-file-name

is an unqualified object file name. To refer to operating system entries (one per CPU), use OS. To refer to operating system-level processes (for example, memory manager, I/O, and disk processes), use OSIMAGE.

In an OSS environment, *program-file-name* is case sensitive.

Note. In OSS, file names can be specified in both uppercase and lowercase characters. In a Guardian environment, they must be specified in uppercase.

You can also embed comments in the initialization file. Text following a semicolon (;) on a line is ignored.

The following entries instruct the Host Resources Subagent to exclude MIB values for all programs except two and to exclude MIB values for one disk:

```
;exclude all programs except tacl and operating
;system entries (one per CPU):
[SWRunTable]
-*
+OS
+TACL
;exclude one disk:
[DeviceTable]
-$scratch            ;not important enough
```

-f *iffile*

specifies the name of a network interface file. The subagent uses this file to maintain values for MIB objects in the hrNetworkTable. If this parameter is not specified, the file name IFTBL in the current volume and subvolume of the user who starts the subagent is assumed.

Note. The subagent simply inserts the interfaces from the file into the MIB table with a state value as STARTED. No monitoring of the interfaces is performed.

You create the network interface file initially by using a Subsystem Control Facility (SCF) INFO SUBNET command. The command should retrieve information for the subnets associated with the TCP/IP subsystem whose

MIB-II data is maintained by the SNMP agent you specify when starting the subagent. For example, if SNMP agent process $ZSNMP is supporting MIB objects for the network under the control of TCP/IP process $ZTC0, you build the network interface file and invoke the subagent as follows:

```
SCF /OUT $DATA.MYSUB.IFTBL/ INFO SUBNET $ZTC0.*
RUN HMSAX /NAME $HMSA, NOWAIT/ -f $DATA.MYSUB.IFTBL
```

If the subagent cannot open the network interface file, no hrNetworkTable information is available. After the subagent is started, however, you can make this table available by setting up the network interface file in the expected location. When the subagent refreshes configuration data, the file is opened and the data processed.

`-h` *`high-value-threshold`*

designates a percent utilization of any RAM storage area or disk device being monitored that constitutes a high value threshold. When an OSS environment is being monitored, the high threshold is checked for the file system.

When utilization of a device reaches or exceeds the high value threshold, the subagent creates an entry in a table in its zhrmThreshold group to describe the device. The default high value threshold is 90%. By default, the subagent sends a trap when utilization of a RAM storage area or disk device has reached or exceeds the high threshold.

After the subagent is started, you can separately manage the high value thresholds for RAM and for disk devices by setting the values of zhrmThrRAMHighValue and zhrmThrDiskHighValue in the zhrmThreshold group.

You can suppress traps announcing high device utilization by specifying a *`high-value-threshold`* of 100 and a *`low-value-threshold`* of 0. After the subagent is started, you can suppress these traps by setting the values of zhrmThrRAMTrapEnable or zhrmThrDiskTrapEnable in the zhrmThreshold group.

Refer to the discussion under Thresholds and Traps on page 11-100 for more information about the relationship between high and low threshold values and trap generation.

`-i` *`low-value-threshold`*

designates a percent utilization of any RAM storage area or disk device being monitored that constitutes a low value threshold. When an OSS environment is being monitored, the low threshold is checked for the file system.

The default low value threshold is 45%. By default, the subagent sends a trap when utilization of a RAM storage area or disk device reaches or falls below the low threshold.

After the subagent is started, you can separately manage the low value thresholds for RAM and for disk devices by setting the values of zhrmThrRAMLowValue and zhrmThrDiskLowValue in the zhrmThreshold group.

Refer to the discussion under Thresholds and Traps on page 11-100 for more information about the relationship between high and low threshold values and trap generation.

`-n` *`process-priority`*

sets the initial process priority of the subagent. If this parameter is not specified, the default initial priority is 145. If the subagent is run as a process pair, this priority applies to both primary and backup processes.

You can change the subagent process priority after the subagent is started by setting the value of zhrmSaPri in the zhrmSaProcess group.

`-p $`*`spooler-process`*

identifies the supervisor process for the Spooler controlling printers whose MIB object values you are interested in. Every Spooler installed on your system has a supervisor process. If you omit the name, the default supervisor process ($SPLS) is presumed.

`-s` *`stable-object-refresh-rate`*

specifies, in 1-minute intervals, the rate at which the subagent refreshes values for the following MIB objects describing the configuration of hardware resources:

hrSystemDate
hrSystemInitialLoadDevice
hrSystemInitialLoadParameters
hrSystemMaxProcesses
hrStorageIndex
hrStorageType
hrStorageDescr
hrStorageAllocationUnits
hrDeviceIndex
hrDeviceType
hrDeviceDescr
hrDeviceID
hrProcessorFrwID
hrNetworkIfIndex
hrDiskStorageAccess
hrDiskStorageMedia
hrDiskStorageRemoveble
hrPartitionIndex
hrPartitionLabel

hrPartitionID
hrPartitionFSIndex
hrFSIndex
hrFSMountPoint
hrFSRemoteMountPoint
hrFSType
hrFSAccess
hrFSBootable
hrFSStorageIndex
hrFSLastFullBackupDate
hrFSLastPartialBackupDate

Because these values are relatively stable, the default stable refresh rate is every 1440 minutes (24 hours). A value of 0 or less suppresses refreshing these values. If the values are not refreshed, devices added after the subagent completes its initialization are not recognized. Similarly, if a device is removed from the system, an SNMP manager user is unaware of its removal because the device is still represented in the subagent's MIB.

If a configuration object is accessed from an SNMP manager within 30 seconds before the stable object timer expires, the subagent postpones refreshing values until another cycle has passed. This minimizes SNMP manager timeouts.

Whenever the values of configuration objects are refreshed, the values for MIB objects in the hrSWRun and hrSWRunPerf groups are also refreshed.

The values of these hardware and software objects can be refreshed on demand from an SNMP manager after the subagent is running by setting the value of the object zhrmStableRefreshNow in the zhrmRefresh group.

Note. While the subagent refreshes configuration objects, SNMP managers interacting with the subagent might experience a timeout. Simply resubmit these requests after waiting a brief period. Large MIBs might require over a minute to be refreshed.

`-t`

causes the subagent to enable Guardian file system and process monitoring. If neither -t or -u is specified, monitoring of the Guardian personality is enabled, and monitoring of the OSS personality is disabled.

`-u`

causes the subagent to enable NonStop Kernel Open System Services (OSS) file system and process monitoring. By default, this option is disabled. If neither -t or -u is specified, monitoring of the Guardian personality is enabled, and monitoring of the OSS personality is disabled.

`-w`

disables creation and maintenance of hrSWRun and hrSWRunPerf group values. This is true for both the Guardian and OSS personalities. If this parameter is not specified, the subagent creates and maintains values in the hrSWRun and hrSWRunPerf groups for processes as described for the -e startup parameter.

Creation and maintenance of the software groups can be enabled or disabled after the subagent is running by setting the value of the object zhrmEnableSoftwareGroup in the zhrmSaProcess group.

`-o`

causes the subagent to write a formatted trace of its interaction with the SNMP agent to the terminal or to a Spooler destination. By default, trace information is sent to the terminal from which the subagent was started. To send the information to a Spooler destination, specify the destination as a TACL RUN option:

```
RUN HMSAX /NAME $HMSA, OUT $S.#OUT, NOWAIT/ -o
```

Spooler information is formatted as a C language-compatible file (file code 180), which you can convert to an edit file (file code 101) or use directly from any utility program.

If you want a second copy of the log file, you can also use the -l startup parameter. However, use of this startup parameter slows subagent processing time because a high volume of trace information is produced.

`-l` *`log-file`*

causes the subagent to write a formatted trace of its interaction with the SNMP agent to a C language-compatible disk file (file code 180), which you can convert to an edit file (file code 101) or use directly from any utility program.

If you want a second copy of the log file, you can also use the -o startup parameter. However, use of this startup parameter slows subagent processing time because a high volume of trace information is produced.

To stop the subagent process, provide its name in the TACL STOP command:

```
STOP $HRMIB
```

Troubleshooting the Subagent

This subsection contains several suggestions for handling problems you might encounter while using the Host Resources Subagent.

EMS Event Messages

Whenever the subagent behaves unexpectedly or an error condition arises, be sure to check for EMS events that have been generated by the subagent. Described under EMS Support on page 11-104, event messages provide information to help you diagnose and fix problems.

Startup Problems

If the subagent stops running shortly after startup, check the EMS event log for unusual conditions. If no EMS events explain the startup difficulty, investigate subagent-agent communication problems.

The SNMP agent specified at subagent startup needs to be running before you start the subagent. If the agent is not running, try to determine why. Common agent startup problems involve security and port conflicts. To monitor port 161 (the standard SNMP port for receiving SNMP manager requests) or any other port less than or equal to 1023, the SNMP agent must run as a process associated with the super user group (user ID 255,*n*). Additionally, the port number assigned at SNMP agent startup must not be in use by some other process, such as another SNMP agent process. Section 2, Installing and Configuring the SNMP Agent provides complete information on agent startup options.

You can use SCF to determine whether a SNMP agent has been started by checking the status of the agent’s ENDPOINT OBJECT, as in the following example, where the name of the SNMP agent process is $ZSNMP:

```
SCF STATUS ENDPOINT $ZSNMP.*
```

A properly started subagent has ENDPOINT objects with a status of STARTED.

Another common cause for startup problems is that the SNMP agent process specified at subagent startup is already in use by another Host Resources Subagent process. Only one Host Resources Subagent process can communicate with a particular SNMP agent process at any time.

To determine whether a SNMP agent process name is correct, retrieve the value of sysDescr from a manager communicating with the agent. This value, one of the System group objects in MIB-II, includes the SNMP agent process name.

Manager Timeouts

Managers interacting with the Host Resources Subagent might experience a timeout when:

- The subagent cannot complete an internal system call in time.
- The size of the MIB being maintained requires prolonged value refresh times.

System Call Suspension

If the subagent does not respond in a timely fashion to a manager request but appears to be running, the subagent might be suspended because it cannot complete a system call needed to process the request or refresh MIB values. This situation can occur when the subagent attempts to retrieve MIB values for remote printers if the connection to the remote system, the remote system itself, or a remote printer becomes unstable.

Remote printers are known as “pseudo” printers. They are actually Spooler collector processes running on a remote system but are configured in the Spooler system as printers. You can use the SPOOLCOM DEV command to display the names of pseudo printers. Their names consist of a node name and a process name; for example, \mysys.$s. To determine the status of a remote printer, use the SPOOLCOM DEV command, specifying a particular remote printer name:

```
DEV \MYSYS.$S, STATUS DETAIL
```

You can also use the TACL STATUS command:

```
STATUS \MYSYS $S, DETAIL
```

To determine the status of the subagent process, use the TACL STATUS command:

```
STATUS $ZHMSA,DETAIL
```

If the WAIT state is %004 (LDONE) and the process time does not change (increase), the subagent is most likely waiting for an internal call to complete.

In many cases, this condition is transient. The subagent becomes responsive again as soon as the communications line or remote system become available. If you experience this problem frequently, you can modify the subagent’s initialization file to exclude offending devices. Refer to information about the -e startup parameter under Starting and Stopping the Subagent on page 11-12 for complete information about encoding initialization file entries.

Large MIBs

The larger the MIB being maintained, the longer it takes the subagent to refresh the values of MIB objects. You can minimize data refresh wait time in several ways:

- The -d and -s startup parameters control the frequency at which the subagent refreshes MIB values. You can specify long refresh intervals at startup and then force refreshes after startup when needed by setting the values of two objects in the zhrmRefresh group: zhrmDynamRefreshNow and zhrmStableRefreshNow.
- The -e startup parameter specifies an initialization file that defines devices and processes to exclude from the MIB. You can use this parameter to limit the resources monitored by the subagent to only those of particular interest at any time.

- The -w startup parameter excludes all software processes from the MIB. You can use this startup parameter, but enable or disable software value maintenance as needed by setting the value of zhrmEnableSoftwareGroup in the zhrmSaProcess group.
- The -t and -u startup parameters can be used to exclude or include monitoring of the Guardian personality or the OSS personality of the SNMP Kernel. You can use these startup parameters, but enable or disable the Guardian or OSS personality maintenance as needed by setting the value of zhrmExamineGuardian or zhrmExamineOSS is zhrmSaProcess group.

Refer to Starting and Stopping the Subagent on page 11-12 for information about startup parameters. Refer to zhrmRefresh Group on page 11-96 and zhrmSaProcess Group on page 11-91 for information about MIB objects you can use to minimize manager timeouts.

hrSystem Group

The hrSystem group is a collection of scalar objects describing general attributes of the host. The Host Resources Subagent's MIB supports the objects identified by a check mark in the following list:dod (6)

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                    mib-2 (1)
                        host (25)
                            hrSystem (1) √
                                hrSystemUpTime (1) √
                                hrSystemDate (2) √
                                hrSystemInitialLoadDevice (3) √
                                hrSystemInitialLoadParameters (4) √
                                hrSystemNumUsers (5) √
                                hrSystemProcesses (6) √
                                hrSystemMaxProcesses (7) √
```

MIB Objects

Table 11-2 describes how the Host Resources Subagent supports objects in the hrSystem group.

Table 11-2. hrSystem Group Objects Supported by Host Resources Subagent's MIB (page 1 of 2)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSystemUpTime 1.3.6.1.2.1.25.1.1 read-only TimeTicks	Number of microseconds since system load.	Integer without leading 0s.	Guardian procedure JULIANTIMESTAMP.
hrSystemDate 1.3.6.1.2.1.25.1.2 read-only DateAndTime	Current Greenwich mean time.	Refer to RFC1514 for an explanation of how to interpret the octet string used to format DateAndTime values.	Guardian procedure JULIANTIMESTAMP.
hrSystemInitialLoadDevice 1.3.6.1.2.1.25.1.3 read-only INTEGER (1..2147483647)	Index of hrDeviceEntry for system load subvolume. hrDeviceEntry is a row in the hrDeviceTable, described under “hrDevice Group.”	Integer without leading 0s.	Guardian procedure NODE_GETCOLDLOADINFO_.
hrSystemInitialLoadParameters 1.3.6.1.2.1.25.1.4 read-only InternationalDisplayString (SIZE (0..128))	The full name of the command interpreter started from the system load subvolume.	$SYSTEM.*subvolume*.TACL.	Guardian procedure NODE_GETCOLDLOADINFO_.

Table 11-2. hrSystem Group Objects Supported by Host Resources Subagent's MIB (page 2 of 2)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSystemNumUsers 1.3.6.1.2.1.25.1.5 read-only Gauge	Number of users running a command interpreter. Users logged in through a non-TACL process such as Pathway are not counted. Users running multiple TACLs are counted only once.	Integer without leading 0s.	Guardian procedure PROCESS_GETINFOLIST_.
hrSystemProcesses 1.3.6.1.2.1.25.1.6 read-only Gauge	Number of process control blocks in both low and high process identification numbers (PINs) currently in use.	Integer without leading 0s.	Guardian procedure PROCESSOR_GETINFOLIST_.
hrSystemMaxProcesses 1.3.6.1.2.1.25.1.7 read-only INTEGER (1..2147483647)	Number of process control blocks in both low and high PINs currently in use plus the number that are not.	Integer without leading 0s.	Guardian procedure PROCESSOR_GETINFOLIST_.

Sample Data

A manager can reformat hrSystemUpTime to convert the microseconds returned by the subagent to an easy-to-interpret format. In addition, in this example and others later in this section, the first value, Time, is added by the manager. It is not an object in the subagent’s MIB.

```
Time, hrSystemUptime, hrSystemDate, hrSystemInitialLoadDevice,
hrSystemInitialLoadParameters, hrSystemNumUsers,
    hrSystemProcesses, hrSystemMaxProcesses

05/22/95 10:32:16, 2 days 17 hours 30 mins 29.42 secs., 07 CB 05 16 0A 19 19
00 , 1005, $SYSTEM.SYS02.TACL, 2,
146, 998
```

RFC Compliance

Table 11-3. Compliance With hrSystem Group Definitions in RFC 1514

Object Descriptor	Compliance	Explanation
hrSystemUpTime	Yes	See Table 11-2.
hrSystemDate	Partial	Set operation not supported.
hrSystemInitialLoadDevice	Partial	Set operation not supported.
hrSystemInitialLoadParameters	Partial	Set operation not supported.
hrSystemNumUsers	Yes	See Table 11-2.
hrSystemProcesses	Yes	See Table 11-2.
hrSystemMaxProcesses	Yes	See Table 11-2.

hrSystem Group Maintenance

The values of the hrSystem group MIB objects are initialized at subagent startup time and then refreshed at a frequency controlled by the -s and -d startup parameters:

-s	hrSystemDate hrSystemInitialLoadDevice hrSystemInitialLoadParameters hrSystemMaxProcesses
-d	hrSystemUpTime hrSystemNumUsers hrSystemProcesses

The values of these objects can be refreshed on demand from an SNMP manager after the subagent is running. Values associated with the -s startup parameter are refreshed by setting the value of zhrmStableRefreshNow in the zhrmRefresh group. Values associated with the -d startup parameter are refreshed by setting the value of zhrmDynamRefreshNow in the zhrmRefresh group.

When values of hrSystem group objects are refreshed, values for objects in the hrSWRun and hrSWRunPerf groups are also refreshed.

The values of hrSystemDate and hrSystemUpTime are also updated each time these objects are accessed in response to a manager request.

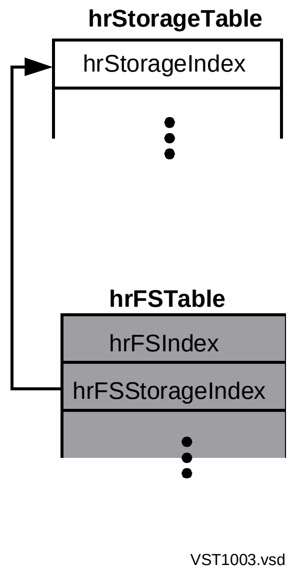
hrStorage Group

The hrStorage group consists of one scalar object and one table (the hrStorage table). The objects in this group describe RAM and disks on the host. The Host Resources Subagent's MIB supports the objects identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                    mib-2 (1)
                        host (25)
                            hrStorage (2) √
                                hrMemorySize(2) √
                                hrStorageTable (3) √
                                    hrStorageEntry (1) √
                                        hrStorageIndex (1) √
                                        hrStorageType (2) √
                                        hrStorageDescr (3) √
                                        hrStorageAllocationUnits (4) √
                                        hrStorageSize (5) √
                                        hrStorageUsed (6) √
                                        hrStorageAllocationFailure (7) √
```

Figure 11-3 illustrates how the hrStorageTable is related to the hrFSTable in the hrDevice group. The values of hrFSStorageIndex in the hrFSTable correspond to values of hrStorageIndex in the hrStorageTable.

Figure 11-3. Relationship Between hrStorageTable and hrFSTable



MIB Objects

Table 11-4. hrStorage Group Objects Supported by Host Resources Subagent's MIB (page 1 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrMemorySize 1.3.6.1.2.1.25.2.2 read-only KBytes	The total amount of physical memory in all CPUs.	Integer without leading 0s.	Guardian procedure PROCESSOR_ GETINFOLIST_ is called once for each CPU and the sum derived after pages are converted to kilobytes.
hrStorageTable Objects:	Entries describing storage areas on the system.		Refer to “hrStorage Group Maintenance” later in this section for information on how entries are maintained.
hrStorageIndex 1.3.6.1.2.1.25.2.3.1.1 read-only INTEGER (1..2147483647)	An index value having a corresponding value (hrFSStorageIndex) in the hrFSTable. A Guardian file system corresponds to a writable disk.	For CPUs: 1 through 16. For disks: 1001 through 10000. For file system: 10001 and above.	Computed by Host Resources Subagent.
hrStorageType 1.3.6.1.2.1.25.2.3.1.2 read-only OBJECT IDENTIFIER	One of the following storage types defined by RFC 1514 that characterizes the entry: hrStorageRam hrStorageFixedDisk hrStorageOther (designating file system entries)	For RAM: 1.3.6.1.2.1.25.2.1.2. For disk: 1.3.6.1.2.1.25.2.1.4. For file system: 1.3.6.1.2.1.25.2.1.1.	RFC 1514.

Table 11-4. hrStorage Group Objects Supported by Host Resources Subagent's MIB (page 2 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrStorageDescr 1.3.6.1.2.1.25.2.3.1.3 read-only DisplayString	Description of the type of storage represented by the entry.	For RAM: the number and type of a processor: CPU*nn* (*processor-type*).running D*nn.nn* For disks and file system entries: the name and type of the disk: `$volume`(*disk-type*). For OSS disks: $*disk-fileset*.	For physical memory: Guardian procedure PROCESSOR_GETINFOLIST_. For disks and file system entries: Guardian procedure FILE_ GETINFOLISTBYNAME_.
hrStorageAllocationUnits 1.3.6.1.2.1.25.2.3.1.4 read-only INTEGER (1..2147483647)	The number of bytes allocated by the entry when storage is requested.	For RAM: an integer without leading 0s, indicating the page size. For disks and file system entries: 2048, indicating one page.	For RAM: Guardian procedure PROCESSOR_ GETINFOLIST_. For disks and file system entries: constant.

Table 11-4. hrStorage Group Objects Supported by Host Resources Subagent's MIB (page 3 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrStorageSize 1.3.6.1.2.1.25.2.3.1.5 read-only INTEGER (1..2147483647)	The capacity of the entry, in number of allocation units.	Integer without leading 0s. For physical memory: if the number exceeds 2,147,483,647, the value returned is -1D. For disks: the number describes the pages formatted. For file system entries: the number describes the size of the writable disk.	For physical memory: Guardian procedure PROCESSOR_GETINFOLIST_. For disks: Guardian procedure FILE_ GETINFOLISTBYNAME_.
hrStorageUsed 1.3.6.1.2.1.25.2.3.1.6 read-only INTEGER (1..2147483647)	The amount of allocated storage associated with the entry, in allocation units.	Integer without leading 0s.	For physical memory: the number of free pages returned by Guardian procedure PROCESSOR_ GETINFOLIST_ is subtracted from the value of hrStorageSize. For disks and file system entries: the number of free pages returned by Guardian procedure FILE_ GETINFOLISTBYNAME_ is subtracted from the value of hrStorageSize.
hrStorageAllocationFailures 1.3.6.1.2.1.25.2.3.1.7 read-only Counter	The number of requests for storage by the entry that could not be honored because of insufficient space.	Integer without leading 0s. For physical memory: the number of page-fault interrupts since the processor was loaded. For disks and file system entries: this value is not available, and 0 is returned.	For physical memory: the page faults returned by Guardian procedure PROCESSOR_ GETINFOLIST_. For disks and file system entries: constant.

Sample Data

```
Time, hrMemorySize

05/22/95 10:33:10, 65536

Time, hrStorageIndex, hrStorageType, hrStorageDescr,
hrStorageAllocationUnits, hrStorageSize, hrStorageUsed,
hrStorageAllocationFailures

05/22/95 10:33:45, 1, 1.3.6.1.2.1.25.2.1.2, CPU00(NSR-L) running D20.06,
4096, 8192, 8169,
3791
05/22/95 10:33:45, 2, 1.3.6.1.2.1.25.2.1.2, CPU01(NSR-L) running D20.06,
4096, 8192, 6982,
1951
05/22/95 10:33:45, 3, 1.3.6.1.2.1.25.2.1.2, CPU02(unknown), 0, 0, 0,
0
05/22/95 10:33:45, 4, 1.3.6.1.2.1.25.2.1.2, CPU03(unknown), 0, 0, 0,
0
05/22/95 10:33:45, 1001, 1.3.6.1.2.1.25.2.1.4, $DSM-P(4230-1C), 2048, 316470,
311818,
0
05/22/95 10:33:45, 1002, 1.3.6.1.2.1.25.2.1.4, $DSM2-P(4220-1), 2048, 146726,
145449,
0
05/22/95 10:33:45, 1003, 1.3.6.1.2.1.25.2.1.4, $DSM3-P(4230-1C), 2048,
316470, 275811,
0
05/22/95 10:33:45, 1004, 1.3.6.1.2.1.25.2.1.4, $ISVD20-P(4230-3), 2048,
316470, 225707,
0
05/22/95 10:33:45, 1005, 1.3.6.1.2.1.25.2.1.4, $SYSTEM-P(4250-2), 2048,
976611, 488322,
0
05/22/95 10:33:45, 10001, 1.3.6.1.2.1.25.2.1.1, $DSM-P(4230-1C), 2048,
316470, 311818,
0
05/22/95 10:33:45, 10002, 1.3.6.1.2.1.25.2.1.1, $DSM2-P(4220-1), 2048,
146726, 145449,
0
05/22/95 10:33:45, 10003, 1.3.6.1.2.1.25.2.1.1, $DSM3-P(4230-1C), 2048,
316470, 275811,
0
05/22/95 10:33:45, 10004, 1.3.6.1.2.1.25.2.1.1, $ISVD20-P(4230-3), 2048,
316470, 225707,
0
05/22/95 10:33:45, 10005, 1.3.6.1.2.1.25.2.1.1, $SYSTEM-P(4250-2), 2048,
976611, 488322,
0
```

RFC Compliance

Table 11-5 summarizes compliance of hrStorage group support with RFC 1514.

Table 11-5. Compliance With hrStorage Group Definitions in RFC 1514

Object Descriptor	Compliance	Explanation
hrMemorySize	Yes	See Table 11-4.
hrStorageIndex	Yes	See Table 11-4.
hrStorageType	Yes	See Table 11-4.
hrStorageDescr	Yes	See Table 11-4.
hrStorageAllocationUnits	Yes	See Table 11-4.
hrStorageSize	Partial	Set operation is not supported.
hrStorageUsed	Yes	See Table 11-4.
hrStorageAllocationFailures	Partial	The value 0 is returned for disk and file system entries.

hrStorage Group Maintenance

The hrStorage group entries describe three kinds of storage available for use by applications:

- **Physical memory**, or RAM. One entry exists for each processor (CPU).
- **Logical disk devices**. One entry exists for each physical disk assigned an LDEV (logical device) number in the destination control table (DCT). Primary and mirrored physical disks are treated as two entities. Nonwriteable disks (for example, optical) are not included. Each logical device is considered to have one partition.
- **File systems**. On a NonStop system, there is one file system per writable disk and as many file system entries as the number of writable disks.
- **OSS**. For OSS, there can be one or more disks for one fileset. An entry exists for the fileset and for each disk of a fileset.

To derive entries for physical memory, the subagent calls Guardian procedure PROCESSOR_GETINFOLIST_ to obtain values for each processor. To derive logical disk device and file system entries, the subagent calls Guardian procedure DEVICE_GETINFOBYLDEV_ to identify writable disks and then calls FILE_GETINFOBYLISTNAME_ to obtain information for each disk.

hrStorage group MIB values are initialized at startup time and then refreshed at a frequency controlled by the -s and -d startup parameters:

`-s`	hrStorageIndex hrStorageType hrStorageDescr hrStorageAllocationUnits
`-d`	hrMemorySize hrStorageSize hrStorageUsed hrStorageAllocationFailures

The values of these objects can be refreshed on demand from an SNMP manager after the subagent is running. Values associated with the -s startup parameter are refreshed by setting the value of zhrmStableRefreshNow in the zhrmRefresh group. Values associated with the -d startup parameter are refreshed by setting the value of zhrmDynamRefreshNow in the zhrmRefresh group.

When values of hrStorage group objects are refreshed, values for objects in the following groups are also refreshed:

- The hrSWRun and hrSWRunPerf groups
- Three of the groups defined by HP: zhrmTableInfo, zhrmThreshold, and zhrmRefresh

hrDevice Group

The hrDevice group is a large group consisting of a main table (the hrDeviceTable), five dependent tables (hrNetworkTable, hrProcessorTable, hrPrinterTable, hrDiskStorageTable, and hrPartitionTable), and the hrFSTable. Objects in the hrDevice group describe processors, network devices, printers, disks, partitions, and file systems on the host. The Host Resources Subagent's MIB supports the objects identified by a check mark in the following list:

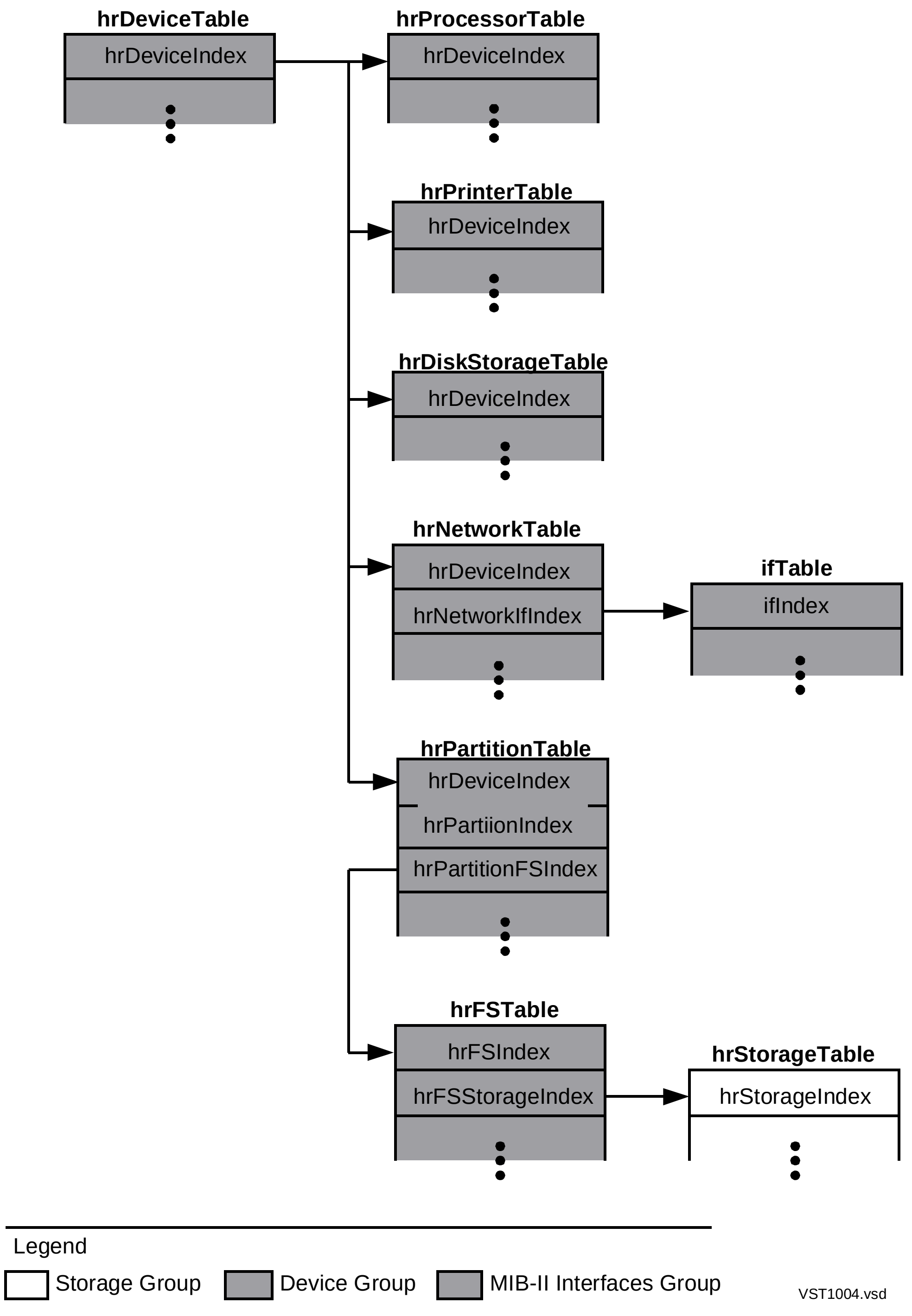
```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                    mib-2 (1)
                        host (25)
                            hrDevice (3) √
                                hrDeviceTable (2) √
                                    hrDeviceEntry (1) √
                                        hrDeviceIndex (1) √
                                        hrDeviceType (2) √
                                        hrDeviceDescr (3) √
                                        hrDeviceID (4) √
            hrDeviceStatus (5) √
            hrDeviceErrors (6) √
    hrProcessorTable (3) √
        hrProcessorEntry (1) √
            hrProcessorFrwID (1) √
            hrProcessorLoad (2) √
    hrNetworkTable (4) √
        hrNetworkEntry (1) √
            hrNetworkIfIndex (1) √
    hrPrinterTable (5) √
        hrPrinterEntry (1) √
            hrPrinterStatus (1) √
            hrPrinterDetectedErrorState (2) √
    hrDiskStorageTable (6) √
    hrDiskStorageEntry (1) √
            hrDiskStorageAccess (1) √
            hrDiskStorageMedia (2) √
            hrDiskStorageRemoveble (3) √
            hrDiskStorageCapacity (4) √
    hrPartitionTable (7) √
        hrPartitionEntry (1) √
            hrPartitionIndex (1) √
            hrPartitionLabel (2) √
            hrPartitionID (3) √
            hrPartitionSize (4) √
            hrPartitionFSIndex (5) √
    hrFSTable (8) √
        hrFSEntry (1) √
            hrFSIndex (1) √
            hrFSMountPoint (2) √
            hrFSRemoteMountPoint (3) √
            hrFSType (4) √
            hrFSAccess (5) √
            hrFSBootable (6) √
            hrFSStorageIndex (7) √
            hrFSLastFullBackupDate (8) √
            hrFSLastPartialBackupDate (9) √
```

Figure 11-4 illustrates hrDevice group table relationships. The major table in the hrDevice group is the hrDeviceTable, which contains an entry for each entry in five other tables in the hrDevice group (hrNetworkTable, hrProcessorTable, hrPrinterTable, hrDiskStorageTable, and hrPartitionTable) Each hrDeviceTable entry has an index (hrDeviceIndex) that points to the entry in those five tables. The hrDevice group also contains an hrFSTable, which contains an entry for every entry in the hrPartitionTable. Each hrPartitionTable entry has an index (hrPartitionFSIndex) that points to an entry in the hrFSTable.

Two additional tables are related to hrDevice group tables: the ifTable and the hrStorageTable. The ifTable is one of the objects in the Interfaces group, one of the MIB-II groups.

The hrStorageTable is described in hrStorage Group on page 11-29.

Figure 11-4. Relationship Among Tables in hrDevice, hrStorage, and Interfaces Groups



MIB Objects

Table 11-6 describes how the Host Resources Subagent supports objects in the hrDevice group.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 1 of 9)

Object and Attributes	**Definition**	**Format of Value**	**Derivation of Value**
hrDeviceTable Objects:	Entries describing and pointing to other entries for devices associated with the system. Entries maintained are controlled by the -e startup parameter.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.
hrDeviceIndex 1.3.6.1.2.1.25.3.2.1.1 read-only INTEGER (1..2147483647)	Index values that point to entries in hrProcessorTable, hrPrinterTable, hrDiskStorageTable hrNetworkTable, and the hrPartitionTable.	For CPUs: 1 through 16. For network devices: 101 through 500. For printers: 501 through 1000. For disks: 1001 through 10000. For tape drives: 10001 through 11000.	Computed by Host Resources Subagent.
hrDeviceType 1.3.6.1.2.1.25.3.2.1.2 read-only OBJECT IDENTIFIER	One of the following device types defined by RFC 1514 that characterizes the entry: hrDeviceProcessor hrDeviceNetwork hrDevicePrinter hrDeviceDisk Storage hrDeviceTape	For CPUs: 1.3.6.1.2.1.25.3.1.3. For printers: 1.3.6.1.2.1.25.3.1.5. For disks: 1.3.6.1.2.1.25.3.1.6. For tape drives: 1.3.6.1.2.1.25.3.1.18. For network devices: 1.3.6.1.2.1.25.3.1.4.	RFC 1514.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 2 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrDeviceDescr 1.3.6.1.2.1.25.3.2.1.3 read-only DisplayString (SIZE (0..64))	Description of the type of device represented by the entry.	CPUs: CPU*nn* (*CPU--type*). Printers: collector name of form $*printer-id*. Spooler collectors, used as pseudo printers, have name form [\\*node--name*]$*collector-process- name* [.#*group-name*]. Disks: name and type of the disk: $*volume* (*disk-type*). Tapes: $*tape-drive-name.* Network devices: *$network-devicedrive-name.* OSS:$*disk-fileset* . Loopback device: $NOIOP.	For CPUs: Guardian procedure PROCESSOR_ GETINFOLIST_. For printers: Spooler procedure SPOOLERSTATUS2. For disks and tape drives: Guardian procedures DEVICE_ GETINFOBYLDEV_ and FILE_ GETINFOBYLISTNAME_
hrDeviceID 1.3.6.1.2.1.25.3.2.1.4 read-only ProductID	An identifier characterizing the entry as a particular device type and subtype.	The root of the identifier is the same for all entries: 1.3.6.1.4.1.169.3.180. Appended to this value are the type and subtype values. For example, the hrDeviceID for a 4500 (1038-MB formatted capacity per spindle) disk would look like this: 1.3.6.1.4.1.169.3.180. 3.22. Refer to the *Guardian Procedure Calls Reference Manual* for a complete list of all device types and complete list of all device types and subtypes.	The product ID for a device is a concatenation of the product ID of the Host Resources Subagent subsystem and the device type and subtype. For processor type, the device type is set to 0.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 3 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrDeviceStatus 1.3.6.1.2.1.25.3.2.1.5 read-only INTEGER { unknown(1), running(2), warning(3), testing(4), down(5) }	The current state of the device represented by the entry.	One of the following values: unknown running warning testing down	For CPUs: Guardian procedure PROCESSORSTATUS. For printers: See hrPrinterStatus. For disks: Guardian procedure DEVICE_ GETINFOBYLDEV_. For tapes: Guardian procedure DEVICE_ GETINFOBYLDEV_ For network devices: Guardian procedure DEVICE_ GETINFOBYLDEV_. For OSS: SCF STATUS FILESET *fileset-name* Refer to "hrDeviceGroup Maintenance," later in this section, for more information.
hrDeviceErrors 1.3.6.1.2.1.25.3.2.1.6 read-only Counter	The number of errors detected on the device described by the entry.	Constant: 0 for OSS	Simulation using information about changes in device states.
hrProcessorTable Objects:	Entries describing processors associated with the system.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.
hrProcessorFrwID 1.3.6.1.2.1.25.3.3.1.1 read-only Product ID	A "dummy" object identifier because it is not accessible.	1.3.6.1.4.1.169.3.180.	Constant.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 4 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrProcessorLoad 1.3.6.1.2.1.25.3.3.1.2 read-only INTEGER (0..100)	The average percent of time a processor was not idle during the last minute.	Integer without leading 0s.	Guardian procedure PROCESSOR_ GETINFOLIST_ is called to obtain two snapshots of non-idle time since system load; then the value is calculated as follows: ((*non-idle-time-2* - *non-idle-time-1*)/*delta-time*)*100.
hrNetworkTable Object:	Entries describing network devices associated with the system.		Refer to "hrDevice Group Maintenance" later in this section for information about how network entries are maintained.
hrNetworkIfIndex 1.3.6.1.2.1.25.3.4.1.1 read-only INTEGER	Value of ifIndex that corresponds to the network device the entry represents.	Integer without leading 0s.	Computed by Host Resources Subagent.
hrPrinterTable Objects:	Entries describing printers associated with the system.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 5 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrPrinterStatus 1.3.6.1.2.1.25.3.5.1.1 read-only INTEGER { other(1), unknown(2), idle(3), printing(4), warmup(5) }	The current state of the printer represented by the entry.	One of the following values: other (in error or unavailable) idle printing unknown If a printer is defined and recorded by the Spooler supervisor, but the device does not exit in reality, the device has an entry in hrDeviceTable and hrPrinterTable. Its device status is defined as down, and the printer status is returned as unknown.	Spooler procedure SPOOLERSTATUS2. The value returned as `state` maps as follows to hrPrinterStatus and hrDeviceStatus values: State / hrPrinter Status / hrDevice Status -------- -------- --------- 1 (device waiting) 3 2 2 (device busy) 4 2 3 (device suspended) 1 5 4 (device error) 1 5 5 (device offline) 1 5 6 (printer process error 1 5
hrPrinterDetectedError State 1.3.6.1.2.1.25.3.5.1.2 read-only OCTET STRING	A description of any error condition existing for the printer.	One of these values: offline: 40 noPaper: 02 no error: 00	The last error value returned by Spooler procedure SPOOLERSTATUS2. Error codes 100 (device not ready) and 104 (device does not respond) are mapped to offline. Error code 102 (paper out) is mapped to noPaper.
hrDiskStorageTable Objects:	Entries describing long-term storage devices associated with the system.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.
hrDiskStorageAccess 1.3.6.1.2.1.25.3.6.1.1 read-only INTEGER { readWrite(1), readOnly(2) }	An indication of whether the disk represented by the entry is readable and writable, or only readable. Optical disks are always read-only.	One of these values: readWrite readOnly	Guardian procedure DEVICE_ GETINFOBYLDEV_. For OSS: RW flag from ZXCONFIG column 6.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 6 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrDiskStorageMedia 1.3.6.1.2.1.25.3.6.1.2 read-only INTEGER { other(1), unknown(2), hardDisk(3), floppyDisk(4), opticalDiskROM(5), opticalDiskWORM(6), opticalDiskRW(7), amDisk(8) }	A characterization of the type of disk represented by the entry.	One of these values: hardDisk opticalDiskWORM The value opticalDiskWORM is returned for nonwriteable disks. The value hardDisk is returned for all other disks and for OSS.	Guardian procedure DEVICE_ GETINFOBYLDEV_. For OSS: Constant
hrDiskStorageRemoveble 1.3.6.1.2.1.25.3.6.1.3 read-only INTEGER { true(1), false(2) }	An indication of whether the disk represented by the entry can be removed from the drive. Optical disks are removable. Most others are not.	One of these values: - true (Removable) - false (nonRemovable) For OSS, the value is always false (nonRemovable).	Guardian procedure DEVICE_ GETINFOBYLDEV_. For OSS: Constant
hrDiskStorageCapacity 1.3.6.1.2.1.25.3.6.1.4 read-only KBytes	The total number of kilobytes the disk represented by the entry can store.	Integer without leading 0s.	Guardian procedure FILE_ GETINFOLIST BYNAME_.e
hrPartitionTable Objects:	Entries describing partitions of long-term storage devices associated with the system.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.
hrPartitionIndex 1.3.6.1.2.1.25.3.7.1.1 read-only INTEGER (1..2147483647)	Index to each partition in a disk. On a NonStop system, each disk has only one partition. For OSS: the index represents the partition number in the fileset.	For Guardian, the value is always 1. For OSS: integer without leading 0s.	For Guardian: Constant. For OSS: computed by the subagent.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 7 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value

hrPartitionLabel
1.3.6.1.2.1.25.3.7.1.2
read-only
 	
InternationalDisplayString
(SIZE (0..128))
	The name of the disk.	For Guardian:
		`$`*`volume`*
		
		`For OSS:`
		`$`*`disk-fileset`* .
			Guardian procedure FILE_ GETINFOLISTBYNAME_.

hrPartitionID 1.3.6.1.2.1.25.3.7.1.3 read-only DisplayString	Partition identifier. On a NonStop system, each disk has only one partition.	A value of 1 is returned for all partition entries.	Constant.
hrPartitionSize 1.3.6.1.2.1.25.3.7.1.4 read-only KBytes	The total number of kilobytes the disk represented by the entry can store.	Integer without leading 0s.	Guardian procedure FILE_ GETINFOLISTBYNAME_.
hrPartitionFSIndex 1.3.6.1.2.1.25.3.7.1.5 read-only INTEGER (1..2147483647)	An index value having a corresponding value (hrFSIndex) in the hrFSTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
hrFSTable Objects:	Entries describing each file system associated with the system. For NonStop systems, there is one file system entry per writable disk.		Refer to "hrDevice Group Maintenance" later in this section for information about how entries are maintained.
hrFSIndex 1.3.6.1.2.1.25.3.8.1.1 read-only INTEGER (1..2147483647)	Index to the entry having a corresponding value (hrPartitionFS Index) in the hrPartition Table.	Integer without leading 0s.	Computed by Host Resources Subagent.
hrFSMountPoint 1.3.6.1.2.1.25.3.8.1.2 read-only InternationalDisplayString (SIZE (0..128))	The name of the disk.	For Guardian: `$`*`volume`* For OSS: Mountpoint /[*`dir`*][/*`subdir`*...] of the fileset.	Guardian procedure FILE_ GETINFOLISTBYNAME_. For OSS: from the fifth column in the ZXCONFIG file.

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 8 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrFSRemoteMountPoint 1.3.6.1.2.1.25.3.8.1.3 read-only InternationalDisplayString (SIZE (0..128))	Name of the system.	`\system name`	
hrFSType 1.3.6.1.2.1.25.3.8.1.4 read-only OBJECT IDENTIFIER	An object identifier from RFC 1514 identifying the type of file system represented by the entry.	1.3.6.1.2.1.25.3.9.1 (hrFSOther)	RFC 1514.
hrFSAccess 1.3.6.1.2.1.25.3.8.1.5 read-only INTEGER { readWrite(1), readOnly(2) }	An indication of whether the entry describes a file system that is readable and writeable, or only readable.	For Guardian: readWrite For OSS: readWrite or readOnly	For Guardian: Always readWrite. For OSS: RW flag from ZXCONFIG file.
hrFSBootable 1.3.6.1.2.1.25.3.8.1.6 read-only INTEGER { true(1), false(2) }	An indication of whether the disk represented by the entry is bootable. For NonStop systems, only $SYSTEM is bootable. For OSS, the disk entry is always nonbootable.	One of these values: true (bootable) false (nonBootable) For OSS: value is always false (nonBootable).	Bootable if hrFSMountPoint is $SYSTEM. For OSS: Constant

Table 11-6. hrDevice Group Objects Supported by Host Resources Subagent's MIB (page 9 of 9)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrFSStorageIndex 1.3.6.1.2.1.25.3.8.1.7 read-only INTEGER (1..2147483647)	An index that has a corresponding value in the hrStorageTable (hrStorageIndex).	Integer without leading 0s.	Computed by Host Resources Subagent.
hrFSLastFullBackup Date 1.3.6.1.2.1.25.3.8.1.8 read-only DateAndTime	For NonStop systems, the value for this object is unknown.	(hex)'00 00 01 01 00 00 00 00', representing the value January 1, year 0000,00:00:00.0.	Constant.
hrFSLastPartialBackup Date 1.3.6.1.2.1.25.3.8.1.9 read-only DateAndTime	For NonStop systems, the value for this object is unknown.	(hex)'00 00 01 01 00 00 00 00', representing the value January 1, year 0000,00:00:00.0.	Constant.

Sample Data

```
Time, hrDeviceIndex, hrDeviceType, hrDeviceDescr, hrDeviceID, hrDeviceStatus,
hrDeviceErrors
```

```
05/22/95 10:35:30, 1, 1.3.6.1.2.1.25.3.1.3, CPU00(NSR-L) running D20.06,
1.3.6.1.4.1.169.3.180.0.6, running, 0
05/22/95 10:35:30, 2, 1.3.6.1.2.1.25.3.1.3, CPU01(NSR-L) running D20.06,
1.3.6.1.4.1.169.3.180.0.6, running, 0
05/22/95 10:35:30, 3, 1.3.6.1.2.1.25.3.1.3, CPU02(unknown),
1.3.6.1.4.1.169.3.180.0.6, down, 1
05/22/95 10:35:30, 4, 1.3.6.1.2.1.25.3.1.3, CPU03(unknown),
1.3.6.1.4.1.169.3.180.0.6, down, 1
05/22/95 10:35:30, 101, 1.3.6.1.2.1.25.3.1.4, $LAM7,
1.3.6.1.4.1.169.3.180.56.0, running, 0
05/22/95 10:35:30, 102, 1.3.6.1.2.1.25.3.1.4, $NOIOP,
1.3.6.1.4.1.169.3.180.56.0, running, 0
05/22/95 10:35:30, 501, 1.3.6.1.2.1.25.3.1.5, $RAVEN,
1.3.6.1.4.1.169.3.180.5.7, running, 1
05/22/95 10:35:30, 502, 1.3.6.1.2.1.25.3.1.5, \DSMDEV.$S,
1.3.6.1.4.1.169.3.180.0.31, running, 0
05/22/95 10:35:30, 1001, 1.3.6.1.2.1.25.3.1.6, $DSM-P(4230-1C),
1.3.6.1.4.1.169.3.180.3.21, running, 0
05/22/95 10:35:30, 1002, 1.3.6.1.2.1.25.3.1.6, $DSM2-P(4220-1),
1.3.6.1.4.1.169.3.180.3.19, running, 0
05/22/95 10:35:30, 1003, 1.3.6.1.2.1.25.3.1.6, $DSM3-P(4230-1C),
1.3.6.1.4.1.169.3.180.3.21, running, 0
05/22/95 10:35:30, 1004, 1.3.6.1.2.1.25.3.1.6, $ISVD20-P(4230-3),
1.3.6.1.4.1.169.3.180.3.21, running, 0
05/22/95 10:35:30, 1005, 1.3.6.1.2.1.25.3.1.6, $SYSTEM-P(4250-2),
1.3.6.1.4.1.169.3.180.3.29, running, 0
05/22/95 10:35:30, 10001, 1.3.6.1.2.1.25.3.1.18, $TAPE,
1.3.6.1.4.1.169.3.180.4.5, running, 0
```

```
05/22/95 10:35:30, 10002, 1.3.6.1.2.1.25.3.1.18, $TAPE4,
1.3.6.1.4.1.169.3.180.4.6, running, 0

Time, hrProcessorFrwID, hrProcessorLoad

05/22/95 10:36:03, 1.3.6.1.4.1.169.3.180, 13
05/22/95 10:36:03, 1.3.6.1.4.1.169.3.180, 1
05/22/95 10:36:03, 1.3.6.1.4.1.169.3.180, 0
05/22/95 10:36:03, 1.3.6.1.4.1.169.3.180, 0

Time, hrNetworkIfIndex

05/22/95, 10:36:15, 2
05/22/95, 10:36:15, 1

Time, hrPrinterStatus, hrPrinterDetectedErrorState

05/22/95, 10:37:00, idle, 00
05/22/95, 10:37:00, printing, 00

Time, hrDiskStorageAccess, hrDiskStorageMedia, hrDiskStorageRemoveble,
hrDiskStorageCapacity

05/22/95 10:37:13, readWrite, hardDisk, 2, 632940
05/22/95 10:37:13, readWrite, hardDisk, 2, 293452
05/22/95 10:37:13, readWrite, hardDisk, 2, 632940
05/22/95 10:37:13, readWrite, hardDisk, 2, 632940
05/22/95 10:37:13, readWrite, hardDisk, 2, 1953222

Time, hrPartitionIndex, hrPartitionLabel, hrPartitionID, hrPartitionSize,
hrPartitionFSIndex

05/22/95 10:37:45, 1, $DSM-P, 1 , 632940, 1
05/22/95 10:37:45, 1, $DSM2-P, 1 , 293452, 2
05/22/95 10:37:45, 1, $DSM3-P, 1 , 632940, 3
05/22/95 10:37:45, 1, $ISVD20-P, 1 , 632940, 4
05/22/95 10:37:45, 1, $SYSTEM-P, 1 , 1953222, 5

Time, hrFSIndex, hrFSMountPoint, hrFSRemoteMountPoint, hrFSType, hrFSAccess,
hrFSBootable, hrFSStorageIndex,
hrFSLastFullBackupDate, hrFSLastPartialBackupDate

05/22/95 10:38:19, 1, $DSM-P, \SHERIFF, 1.3.6.1.2.1.25.3.9.1, readWrite, 0,
10001,
00 00 01 01 00 00 00 00 , 00 00 01 01 00 00 00 00
05/22/95 10:38:19, 2, $DSM2-P, \SHERIFF, 1.3.6.1.2.1.25.3.9.1, readWrite, 0,
10002,
00 00 01 01 00 00 00 00 , 00 00 01 01 00 00 00 00
05/22/95 10:38:19, 3, $DSM3-P, \SHERIFF, 1.3.6.1.2.1.25.3.9.1, readWrite, 0,
10003,
00 00 01 01 00 00 00 00 , 00 00 01 01 00 00 00 0
05/22/95 10:38:19, 4, $ISVD20-P, \SHERIFF, 1.3.6.1.2.1.25.3.9.1, readWrite,
0, 10004,
00 00 01 01 00 00 00 00 , 00 00 01 01 00 00 00 00
05/22/95 10:38:19, 5, $SYSTEM-P, \SHERIFF, 1.3.6.1.2.1.25.3.9.1, readWrite,
1, 10005,
00 00 01 01 00 00 00 00 , 00 00 01 01 00 00 00 00
```

RFC Compliance

Table 11-7 summarizes compliance of hrDevice group support with RFC 1514.

Table 11-7. Compliance With hrDevice Group Definitions in RFC 1514

Object Descriptor	Compliance	Explanation
hrDeviceIndex	Yes	See Table 11-6.
hrDeviceType	Yes	See Table 11-6.
hrDeviceDescr	Yes	See Table 11-6.
hrDeviceID	Yes	See Table 11-6.
hrDeviceStatus	Yes	See Table 11-6.
hrDeviceErrors	Yes	See Table 11-6.
hrProcessorFrwID	Partial	Value is a “dummy” object identifier.
hrProcessorLoad	Yes	See Table 11-6.
hrNetworkIfIndex	Yes	See Table 11-6.
hrPrinterStatus	Yes	See Table 11-6.
hrPrinterDetectedErrorState	Yes	See Table 11-6.
hrDiskStorageAccess	Yes	See Table 11-6.
hrDiskStorageMedia	Yes	See Table 11-6.
hrDiskStorageRemoveble	Yes	See Table 11-6.
hrDiskStorageCapacity	Yes	See Table 11-6.
hrPartitionIndex	Yes	See Table 11-6.
hrPartitionLabel	Yes	See Table 11-6.
hrPartitionID	Yes	See Table 11-6.
hrPartitionSize	Yes	See Table 11-6.
hrPartitionFSIndex	Yes	See Table 11-6.
hrFSIndex	Yes	See Table 11-6.
hrFSMountPoint	Yes	See Table 11-6.
hrFSRemoteMountPoint	Yes	See Table 11-6.
hrFSType	Yes	See Table 11-6.
hrFSAccess	Yes	See Table 11-6.
hrFSBootable	Yes	See Table 11-6.
hrFSStorageIndex	Yes	See Table 11-6.
hrFSLastFullBackupDate	Partial	Set operation is not supported. Value is set to January 1, year 0000,00:00:00.0.
hrFSLastPartialBackupDate	Partial	Set operation is not supported. Value is set to January 1, year 0000,00:00:00.0.

hrDevice Group Maintenance

Values in the hrDevice group support the following device types:

Processors	The subagent uses Guardian procedures PROCESSORSTATUS, PROCESSORTYPE, and PROCESSOR_GETINFOLIST_ to construct processor entries in the hrProcessorTable.
Network interfaces	The subagent accesses the network interface file to build hrNetworkTable entries. Refer to Starting and Stopping the Subagent on page 11-12 for a description of how the -f startup parameter is used to identify the network interface file.
Printers	The subagent uses Spooler procedure SPOOLERSTATUS2 and Guardian procedure FILE_GETINFOLISTBYNAME_ to obtain information about printers for the hrPrinterTable. The Spooler can define virtual printers, usually collector processes that forward print jobs to a remote printer. Entries are also created for virtual printers.
Disks	The subagent uses Guardian procedures DEVICE_GETINFOBYLDEV_ and FILE_GETINFOLISTBYNAME_ to construct entries in the hrDiskStorageTable for both read-only disks and other disks.
Tape devices	The subagent creates entries describing tape devices by using Guardian procedure DEVICE_GETINFOBYLDEV_.
OSS filesets	The subagent creates entries for OSS filesets by reading the ZXCONFIG file.

Some of these devices are also represented in the hrPartitionTable and the hrFSTable. The subagent assigns one partition and one file system per logical disk, using the information it maintains for writable disks.

One entry appears in the hrDeviceTable for each of the six preceding device types listed above. Indexes for each entry, derived from a range of values unique to each device type as described in Table 11-6, point to index values in five other tables: hrProcessorTable, hrNetworkTable, hrPrinterTable, hrDiskStorageTable, and hrPartitionTable. Once assigned, a device index is never reused, even when a device is removed from the system. When a new device is added to the system, it is assigned an index value that has never been used from the assigned range of index values.

The values of hrDeviceStatus in the hrDeviceTable map as follows to device states returned when Guardian and Spooler procedures are called:

Device	Device State	hrDeviceStatus
Processors, tapes, and network interfaces	up	2 (running)
	down	5 (down)
Disks	up	2 (running)
	down	5 (down)
	special	4 (testing)
	mount	4 (testing)
	revive	3 (warning)
	exercise	4 (testing)
	exclusive	4 (testing)
	harddown	5 (down)
Printers	waiting	2 (running)
	busy	2 (running)
	suspended	5 (down)
	error	5 (down)
	offline	5 (down)
	proc error	5 (down)
	paper out	5 (down)
OSS	diagnosing	4 (testing)
	started	2 (running)
	stopped	5 (down)
	unknown	1 (unknown)

The hrDevice group tables expand, contract, and change as resources change. Initialized at subagent startup time, MIB values are refreshed at a frequency controlled by the -s and -startup parameters:

`-s`	hrDeviceIndex hrDeviceType hrDeviceDescr hrDeviceID hrProcessorFrwID hrNetworkIfIndex hrDiskStorageAccess hrDiskStorageMedia hrDiskStorageRemoveble hrPartitionIndex hrPartitionLabel hrPartitionID hrPartitionFSIndex hrFSIndex hrFSMountPoint hrFSRemoteMountPoint hrFSType hrFSAccess hrFSBootable hrFSStorageIndex hrFSLastFullBackupDate hrFSLastPartialBackupDate
`-d`	hrDeviceStatus hrPrinterStatus hrPrinterDetectedErrorState hrDeviceErrors hrProcessorLoad hrDiskStorageCapacity hrPartitionSize

The values of these objects can be refreshed on demand from an SNMP manager after the subagent is running. Values associated with the -s startup parameter are refreshed by setting the value of zhrmStableRefreshNow in the zhrmRefresh group. Values associated with the -d startup parameter are refreshed by setting the value of zhrmDynamRefreshNow in the zhrmRefresh group.

When values of hrDevice group objects are refreshed, values for objects in the following groups are also refreshed:

- The hrSWRun and hrSWRunPerf groups
- Four of the groups defined by HP: zhrmTableInfo, zhrmDevUnavail, zhrmThreshold, and zhrmRefresh.

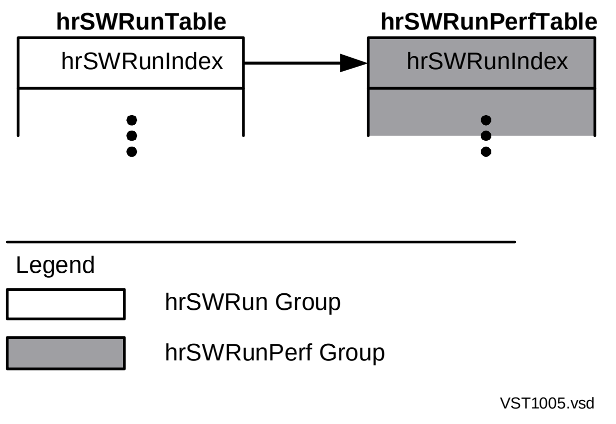
hrSWRun Group

The hrSWRun group consists of a scalar object and one table (the hrSWRunTable). The objects in this group describe processes running on the host. The Host Resources Subagent's MIB supports the objects identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                    mib-2 (1)
                        host (25)
                            hrSWRun (4) √
                                hrSWOSIndex (1) √
                                hrSWRunTable (2) √
                                    hrSWRunEntry (1) √
                                        hrSWRunIndex (1) √
                                        hrSWRunName (2) √
                                        hrSWRunID (3) √
                                        hrSWRunPath (4) √
                                        hrSWRunParameters (5) √
                                        hrSWRunType (6) √
                                        hrSWRunStatus (7) √
```

Figure 11-5 illustrates how the hrSWRunTable is related to the hrSWRunPerfTable in the hrSWRunPerf group. Each hrSWRunTable entry has an index (hrSWRunIndex) that points to an entry in hrSWRunPerfTable.

Figure 11-5. Relationship Between hrSWRunTable and hrSWRunPerfTable



MIB Objects

Table 11-8 describes how the Host Resources Subagent supports objects in the hrSWRun group.

)

Table 11-8. hrSWRun Group Objects Supported by Host Resources Subagent's MIB (page 1 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSWOSIndex 1.3.6.1.2.1.25.4.1 read-only INTEGER (1..2147483647)	An index value that points to the hrSWRunTable entry for the operating system associated with lowest numbered CPU on the system.	Integer without leading 0s.	Computed by Host Resources Subagent.
hrSWRunTable Objects:	Entries describing each process specified by the -e startup parameter that is currently running on the system. Each process running has one entry. If the process is running as a process pair, each member has a separate entry.		Refer to “hrSWRun Group Maintenance” later in this section for information on how entries are maintained.
hrSWRunIndex 1.3.6.1.2.1.25.4.2.1.1 read-only INTEGER (1..2147483647)	Index values that point to entries in the hrSWRun PerfTable.	For the Host Resources Subagent: 1 (primary process) or 2 (backup process) For the SNMP agent:11 (primary process) or 12 (backup process) For the operating system associated with each CPU: 100 through 199 For OSS processes: 20000+((CPU+1)*1000)+ PIN For other processes: ((CPU+1)*1000)+PIN	Computed by Host Resources Subagent.

Table 11-8. hrSWRun Group Objects Supported by Host Resources Subagent's MIB (page 2 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSWRunName 1.3.6.1.2.1.25.4.2.1.2 read-only InternationalDisplayString (SIZE (0..64))	The unqualified name of the program file whose process the entry represents.	For non-operating system processes: unqualified file name. For operating systems: NSK D*nn*, where D*nn* is the operating system version.	For non-operating system processes, the name is derived by using Guardian procedure PROCESS_GETINFOLIST_.
hrSWRunID 1.3.6.1.2.1.25.4.2.1.3 read-only ProductID	An identifier for the process.	1.3.6.1.4.1.169.3.180.0	For non-operating system processes, a 0 is added to the subagent's object identifier. For operating systems, 9050 is added. For OSS, 9594 is added.
hrSWRunPath 1.3.6.1.2.1.25.4.2.1.4 read-only InternationalDisplayString (SIZE (0..128))	The fully qualified name of the program file whose process the entry represents.	For non-operating system processes: \node- name.$*volume--name*.*subvolume-name*.*program-file-name* For operating systems: $SYSTEM.SYS*nn*.OS, identifying the subvolume from which the system is loaded. For OSS: */dir/*[*subdir/*..]/*object*	The name is derived by using Guardian procedure PROCESS_GETINFOLIST_.

Table 11-8. hrSWRun Group Objects Supported by Host Resources Subagent's MIB (page 3 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSWRunParameters 1.3.6.1.2.1.25.4.2.1.5 read-only InternationalDisplayString (SIZE (0..128))	Information describing the process. For OSS: Parameters used when starting the program.	A series of values describing such process attributes as CPU number, PIN, process role (primary, backup, single), process name, process start time, PAID, home terminal ID, and CPU load time. An example for a non-operating system process is: $HMSA1-P(9,87) started 1995, April 12, 1995 12:11:32.2 by 165,243 An example for an operating system process is: CPU 0 loaded with D20.06 at March 18, 1995 14:44:3.0 (running 36 days, 21 hours, 7 minutes, 3 seconds)	The information is derived by using Guardian procedure PROCESS_ GETINFOLIST_.
hrSWRunType 1.3.6.1.2.1.25.4.2.1.6 read-only INTEGER { unknown(1), operatingSystem(2), deviceDriver(3), application(4) }	The type of software the entry represents.	One of the following values: operatingSystem application The value *operatingSystem* is returned for NonStop system processes. The value *application* is returned for all other processes, including OSS.	When Guardian procedure PROCESS_ GETINFOLIST_ returns a value of 3 for process state, a process is identified as an operating system process. Any other return value indicates the process is an application process.

Table 11-8. hrSWRun Group Objects Supported by Host Resources Subagent's MIB (page 4 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSWRunStatus 1.3.6.1.2.1.25.4.2.1.7 read-only INTEGER { running(1), runnable(2), notRunnable(3), invalid(4) }	The state of the process.	One of the following values: running runnable	The process state information returned by Guardian procedure PROCESS_ GETINFOLIST_ is mapped to values as follows: 1 if bit 1 and 2 are off 2 if bit 1 or 2 is on For operating system entries, the value is always running.

Sample Data

```
Time, hrSWOSIndex

05/22/95 10:39:06, 100

Time, hrSWRunIndex, hrSWRunName, hrSWRunID, hrSWRunPath,
hrSWRunParameters,
hrSWRunType, hrSWRunStatus

05/22/95 10:39:52, 1, HMSAXR, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$DSM.VINHMS.HMSAXR,
$HMSA1-P(0,47) started May 20, 1995 17:20:20.7 by 255,255,
application, runnable
05/22/95 10:39:52, 2, HMSAXR, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$DSM.VINHMS.HMSAXR,
$HMSA1-B(1,39) started May 20, 1995 17:20:26.6 by 255,255,
application, running
05/22/95 10:39:52, 11, SNMPAGT, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$DSM.SNMP.SNMPAGT,
$ZSNMP-P(0,43) started May 20, 1995 17:19:31.9 by 255,255,
application, running
05/22/95 10:39:52, 12, SNMPAGT, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$DSM.SNMP.SNMPAGT,
$ZSNMP-B(1,38) started May 20, 1995 17:19:38.8 by 255,255,
application, running
05/22/95 10:39:52, 100, NSK D20, 1.3.6.1.4.1.169.3.180.9050,
$SYSTEM.SYS02.OS,
CPU 0 loaded with D20.06 at May 19, 1995 16:54:56.3 (running 2 days, 17
hours, 14 minutes, 54 seconds), operatingSystem, running
05/22/95 10:39:52, 101, NSK D20, 1.3.6.1.4.1.169.3.180.9050,
$SYSTEM.SYS02.OS,
CPU 1 loaded with D20.06 at May 19, 1995 16:58:15.4 (running 2 days, 17
hours, 11 minutes, 36 seconds),
operatingSystem, running
05/22/95 10:39:52, 1019, XMIOP, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.XMIOP,
$XMIOP-S(0,19) started May 19, 1995 16:55:56:42.1 by 255,255,
application, running
05/22/95 10:39:52, 1027, SYHZPER, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.SYHZPER,
$ZPER-P(0,27) started May 19, 1995 16:56:44.8 by 255,255,
application, running
05/22/95 10:39:52, 1028, SYHZNOT, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.SYHZNOT,
$ZNOT-S(0,28) started May 19, 1995 16:57:26.7 by 255,255,
application, running
05/22/95 10:39:52, 1029, SYHZSHM, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.SYHZSHM,
$ZSHM-S(0,29) started May 19, 1995 16:57:26.7 by 255,255,
application, running
05/22/95 10:39:52, 1042, MLMAN, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.MLMAN,
$ZLMG-B(0,42) started May 20, 1995 17:3:8.8 by 255,255,
application, running
05/22/95 10:39:52, 1045, LISTNER, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.ZTCPIP.LISTNER,
$ZPRT-S(0,45) started May 20, 1995 17:4:12.3 by 255,255,
application, running
05/22/95 10:39:52, 2018, SYHZPER, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.SYHZPER,
$ZPER-B(1,18) started May 19, 1995 16:58:36.8 by 255,255,
application, running
05/22/95 10:39:52, 2036, MLMAN, 1.3.6.1.4.1.169.3.180.0,
\SHERIFF.$SYSTEM.SYS02.MLMAN,
$ZLMG-P(1,36) started May 20, 1995 17:2:51.1 by 255,255,
application, running
```

RFC Compliance

Table 11-9 summarizes compliance of hrSWRun group support with RFC 1514.

Table 11-9. Compliance With hrSWRun Group Definitions in RFC 1514

Object Descriptor	Compliance	Explanation
hrSWOSIndex	Yes	See Table 11-8.
hrSWRunIndex	Yes	See Table 11-8.
hrSWRunName	Yes	See Table 11-8.
hrSWRunID	Yes	See Table 11-8.
hrSWRunPath	Yes	See Table 11-8.
hrSWRunParameters	Yes	See Table 11-8.
hrSWRunType	Yes	See Table 11-8.
hrSWRunStatus	Partial	Set operation not supported.

hrSWRun Group Maintenance

If the value of zhrmEnableSoftwareGroup in the zhrmSaProcess group is set to 1, the values of objects in this group are all derived anew, and existing values deleted whenever these events occur:

- The subagent starts.
- One of the refresh timers set at startup time (-s or -d) runs down.
- The value of one of the following objects in the zhrmRefresh group is set to 1: zhrmStableRefreshNow or zhrmDynamRefreshNow.

You control whether the subagent creates or maintains hrSWRun group objects by using the -w startup parameter or, after startup, by setting the value of zhrmEnableSoftwareGroup in the zhrmSaProcess group:

- Use the -w startup parameter to suppress creation of hrSWRun group objects. Omit the -w startup parameter to enable object creation.
- Set zhrmEnableSoftwareGroup to 0 to disable maintenance of hrSWRunGroup objects. Set it to 1 to enable object maintenance.

hrSWRunPerf Group

The hrSWRunPerf group consists of one table (the hrSWRunPerfTable), which contains one entry for each entry in the hrSWRunTable in the hrSWRun group. The objects in the hrSWRunPerf group describe the CPU and memory utilization of processes running on the host. The Host Resources Subagent's MIB supports the objects identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                mgmt (2)
                    mib-2 (1)
                        host (25)
                            hrSWRunPerf (5) √
                                hrSWRunPerfTable (1) √
                                    hrSWRunPerfEntry (1) √
                                        hrSWRunPerfCPU (1) √
                                        hrSWRunPerfMem (2) √
```

MIB Objects

Table 11-10 describes how the Host Resources Subagent supports objects in the hrSWRunPerf group.

Table 11-10. hrSWRunPerf Group Objects Supported by Host Resources Subagent's MIB

Object and Attributes	Definition	Format of Value	Derivation of Value
hrSWRunPerfTable Objects:	Entries providing performance metrics for resources having entries in the hrSWRun group.		Refer to “hrSWRunPerf Group Maintenance” later in this section for information on how entries are maintained.
hrSWRunPerfCPU 1.3.6.1.2.1.25.5.1.1.1 read-only INTEGER	The number of centiseconds of the total system's CPU resources consumed by the process described by this entry. For operating system entries, the CPU busy time.	Integer without leading 0s.	The process time is obtained by calling Guardian procedure PROCESS_GETINFOLIST_. If the time is less than 1 centisecond (tick), the value is rounded up to 1.
hrSWRunPerfMem 1.3.6.1.2.1.25.5.1.1.2 read-only KBytes	The kilobytes of memory pages that have been swapped in by the process represented by this entry and which are still resident. For operating system entries, the amount of RAM installed in the corresponding CPU.	Integer without leading 0s.	Guardian procedure FILE_GETINFOLIST_. The value for non-operating system entries is derived from the current page value returned.

Sample Data

```
Time, hrSWRunPerfCPU, hrSWRunPerfMem

05/22/95 10:40:30, 170367, 0
05/22/95 10:40:30, 3, 0
05/22/95 10:40:30, 140058, 0
05/22/95 10:40:30, 8, 0
05/22/95 10:40:30, 23489477, 32768
05/22/95 10:40:30, 23469643, 32768
05/22/95 10:40:30, 10, 0
05/22/95 10:40:30, 21, 0
05/22/95 10:40:30, 518, 0
05/22/95 10:40:30, 3527, 0
05/22/95 10:40:30, 6, 0
05/22/95 10:40:30, 64, 0
05/22/95 10:40:30, 1, 0
05/22/95 10:40:30, 50819, 0
```

RFC Compliance

Table 11-11 summarizes compliance of hrSWRunPerf group support with RFC 1514.

Table 11-11. Compliance With hrSWRunPerf Group Definitions in RFC 1514

Object Descriptor	Compliance	Explanation
hrSWRunPerfCPU	Yes	See Table 11-10.
hrSWRunPerfMem	Partial	For non-operating system entries, the number of kilobytes associated with the memory pages that have been swapped in by the process and that are still resident is returned. For operating system entries, the amount of RAM installed in the corresponding CPU is returned.

hrSWRunPerf Group Maintenance

Whenever a value in the hrSWRun group changes, the subagent reflects the change in the hrSWRunPerf group to retain a one-to-one correspondence of entries in tables in both groups. All old values are deleted and replaced with all new values.

You control whether the subagent creates or maintains hrSWRun group objects by using the -w startup parameter or, after startup, by setting the value of zhrmEnableSoftwareGroup in the zhrmSaProcess group:

- Use the -w startup parameter to suppress creation of hrSWRun group objects. Omit the -w startup parameter to enable object creation.
- Set zhrmEnableSoftwareGroup to 0 to disable maintenance of hrSWRunGroup objects. Set it to 1 to enable object maintenance.

zhrmTableInfo Group

This group, defined by HP, provides information about the status of operations performed on tables in the hrStorage and hrDevice groups. The zhrmTableInfo group consists of eight groups, each of which contains objects that describe one of the tables in the hrStorage or hrDevice groups:

zhrmTableInfo Group	**Corresponding hrStorage Group Table**	**Corresponding hrDevice Group Table**
zhrmInfStorTable	hrStorageTable	
zhrmInfDevTable		hrDeviceTable
zhrmInfCpuTable		hrProcessorTable
zhrmInfNetTable		hrNetworkTable
zhrmInfPrnTable		hrPrinterTable
zhrmInfDiskTable		hrDiskStorageTable
zhrmInfPartTable		hrPartitionTable
zhrmInfFSTable		hrFSTable

Objects in the zhrmTableInfo group are identified by a check mark in the following list:

```
iso (1)
        identified-organization (3)
            dod (6)
                internet (1)
                    private (4)
                        enterprises (1)
                            tandem (169)
                                nonstopsystems (3)
                                    zhrm (180)
                                        zhrmTableInfo (1) √
                                            zhrmInfStorTable (1) √
                                                zhrmInfStorEntries (1) √
                                                zhrmInfStorFirstIndex (2) √
                                                zhrmInfStorLastIndex (3) √
                                                zhrmInfStorLastDynamRefresh (4) √
                                                zhrmInfStorLastStableRefresh (5) √
                                                zhrmInfStorGetCounter (6) √
                                                zhrmInfStorGetNextCounter (7) √
                                                zhrmInfStorSetCounter (8) √
                                            zhrmInfDevTable (2) √
                                                zhrmInfDevEntries (1) √
                                                zhrmInfDevFirstIndex (2) √
                                                zhrmInfDevLastIndex (3) √
                                                zhrmInfDevLastDynamRefresh (4) √
                                                zhrmInfDevLastStableRefresh (5) √
                                                zhrmInfDevGetCounter (6) √
```

```
    zhrmInfDevGetNextCounter (7) √
    zhrmInfDevSetCounter (8) √
zhrmInfCpuTable (3) √
    zhrmInfCpuEntries (1) √
    zhrmInfCpuFirstIndex (2) √
    zhrmInfCpuLastIndex (3) √
    zhrmInfCpuLastDynamRefresh (4) √
    zhrmInfCpuLastStableRefresh (5) √
    zhrmInfCpuGetCounter (6) √
    zhrmInfCpuGetNextCounter (7) √
    zhrmInfCpuSetCounter (8) √
zhrmInfNetTable (4) √
    zhrmInfNetEntries (1) √
    zhrmInfNetFirstIndex (2) √
    zhrmInfNetLastIndex (3) √
    zhrmInfNetLastDynamRefresh (4) √
    zhrmInfNetLastStableRefresh (5) √
    zhrmInfNetGetCounter (6) √
    zhrmInfNetGetNextCounter (7) √
    zhrmInfNetSetCounter (8) √

zhrmInfPrnTable (5) √
    zhrmInfPrnEntries (1) √
    zhrmInfPrnFirstIndex (2) √
    zhrmInfPrnLastIndex (3) √
    zhrmInfPrnLastDynamRefresh (4) √
    zhrmInfPrnLastStableRefresh (5) √
    zhrmInfPrnGetCounter (6) √
    zhrmInfPrnGetNextCounter (7) √
    zhrmInfPrnSetCounter (8) √
zhrmInfDiskTable (6) √
    zhrmInfDiskEntries (1) √
    zhrmInfDiskFirstIndex (2) √
    zhrmInfDiskLastIndex (3) √
    zhrmInfDiskLastDynamRefresh (4) √
    zhrmInfDiskLastStableRefresh (5) √
    zhrmInfDiskGetCounter (6) √
    zhrmInfDiskGetNextCounter (7) √
    zhrmInfDiskSetCounter (8) √
zhrmInfPartTable (7) √
    zhrmInfPartEntries (1) √
    zhrmInfPartFirstIndex (2) √
    zhrmInfPartLastIndex (3) √
    zhrmInfPartLastDynamRefresh (4) √
    zhrmInfPartLastStableRefresh (5) √
    zhrmInfPartGetCounter (6) √
    zhrmInfPartGetNextCounter (7) √
    zhrmInfPartSetCounter (8) √
```

zhrmInfFSTable (8) √
  zhrmInfFSEntries (1) √
  zhrmInfFSFirstIndex (2) √
  zhrmInfFSLastIndex (3) √
  zhrmInfFSLastDynamRefresh (4) √
  zhrmInfFSLastStableRefresh (5) √
  zhrmInfFSGetCounter (6) √
  zhrmInfFSGetNextCounter (7) √
  zhrmInfFSSetCounter (8) √

MIB Objects

Table 11-12 describes how the Host Resources Subagent supports objects in the zhrmTableInfo group.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 1 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfStorTable Group Objects:	Objects describing the status and usage of the hrStorageTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfStorEntries 1.3.6.1.4.1.169.3.180.1.1.1 read-only Gauge	The current number of entries in the hrStorageTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfStorFirstIndex 1.3.6.1.4.1.169.3.180.1.1.2 read-only INTEGER (1..65535)	The lowest value of hrStorageIndex.	Integer without leading 0s.	hrStorageTable.
zhrmInfStorLastIndex 1.3.6.1.4.1.169.3.180.1.1.3 read-only INTEGER (1..65535)	The highest value of hrStorageIndex.	Integer without leading 0s.	hrStorageTable.
zhrmInfStorLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.1.4 read-only DisplayString	The system time at which hrStorageTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 2 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfStorLastStableRefresh 1.3.6.1.4.1.169.3.180.1.1.5 read-only DisplayString	The system time at which hrStorageTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfStorGetCounter 1.3.6.1.4.1.169.3.180.1.1.6 read-only Counter	The number of Get operations performed on hrStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfStorGetNextCounter 1.3.6.1.4.1.169.3.180.1.1.7 read-only Counter	The number of GetNext operations performed on hrStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfStorSetCounter 1.3.6.1.4.1.169.3.180.1.1.8 read-only Counter	The number of Set operations performed on hrStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDevTable Group Objects:	Objects describing the status and usage of the hrDeviceTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfDevEntries 1.3.6.1.4.1.169.3.180.1.2.1 read-only Gauge	The current number of entries in the hrDeviceTable.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 3 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfDevFirstIndex 1.3.6.1.4.1.169.3.180.1.2.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex.	Integer without leading 0s.	hrDeviceTable.
zhrmInfDevLastIndex 1.3.6.1.4.1.169.3.180.1.2.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex.	Integer without leading 0s.	hrDeviceTable.
zhrmInfDevLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.2.4 read-only DisplayString	The system time at which hrDeviceTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfDevLastStableRefresh 1.3.6.1.4.1.169.3.180.1.2.5 read-only DisplayString	The system time at which hrDeviceTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfDevGetCounter 1.3.6.1.4.1.169.3.180.1.2.6 read-only Counter	The number of Get operations performed on hrDeviceTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDevGetNextCounter 1.3.6.1.4.1.169.3.180.1.2.7 read-only Counter	The number of GetNext operations performed on hrDeviceTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 4 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfDevSetCounter 1.3.6.1.4.1.169.3.180.1.2.8 read-only Counter	The number of Set operations performed on hrDeviceTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfCpuTable Group Objects:	Objects describing the status and usage of the hrProcessorTable.		Refer to "zhrmTableInfo Group Maintenance" later in this section for information on how object values are maintained.
zhrmInfCpuEntries 1.3.6.1.4.1.169.3.180.1.3.1 read-only Gauge	The current number of entries in the hrProcessorTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfCpuFirstIndex 1.3.6.1.4.1.169.3.180.1.3.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex for hrProcessorTable entries.	Integer without leading 0s.	hrProcessorTable.
zhrmInfCpuLastIndex 1.3.6.1.4.1.169.3.180.1.3.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex for hrProcessorTable entries.	Integer without leading 0s.	hrProcessorTable.
zhrmInfCpuLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.3.4 read-only DisplayString	The system time at which hrProcessorTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfCpuLastStableRefresh 1.3.6.1.4.1.169.3.180.1.3.5 read-only DisplayString	The system time at which hrProcessorTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 5 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfCpuGetCounter 1.3.6.1.4.1.169.3.180.1.3.6 read-only Counter	The number of Get operations performed on hrProcessorTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfCpuGetNextCounter 1.3.6.1.4.1.169.3.180.1.3.7 read-only Counter	The number of GetNext operations performed on hrProcessorTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfCpuSetCounter 1.3.6.1.4.1.169.3.180.1.3.8 read-only Counter	The number of Set operations performed on hrProcessorTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfNetTable Group Objects:	Objects describing the status and usage of the hrNetworkTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfNetEntries 1.3.6.1.4.1.169.3.180.1.4.1 read-only Gauge	The current number of entries in the hrNetworkTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfNetFirstIndex 1.3.6.1.4.1.169.3.180.1.4.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex for hrNetworkTable entries.	Integer without leading 0s.	hrNetworkTable.
zhrmInfNetLastIndex 1.3.6.1.4.1.169.3.180.1.4.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex for hrNetworkTable entries.	Integer without leading 0s.	hrNetworkTable.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 6 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfNetLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.4.4 read-only DisplayString	The system time at which hrNetworkTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfNetLastStableRefresh 1.3.6.1.4.1.169.3.180.1.4.5 read-only DisplayString	The system time at which hrNetworkTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfNetGetCounter 1.3.6.1.4.1.169.3.180.1.4.6 read-only Counter	The number of Get operations performed on hrNetworkTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfNetGetNextCounter 1.3.6.1.4.1.169.3.180.1.4.7 read-only Counter	The number of GetNext operations performed on hrNetworkTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfNetSetCounter 1.3.6.1.4.1.169.3.180.1.4.8 read-only Counter	The number of Set operations performed on hrNetworkTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 7 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfPrnTable Group Objects:	Objects describing the status and usage of the hrPrinterTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfPrnEntries 1.3.6.1.4.1.169.3.180.1.5.1 read-only Gauge	The current number of entries in the hrPrinterTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPrnFirstIndex 1.3.6.1.4.1.169.3.180.1.5.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex for hrPrinterTable entries.	Integer without leading 0s.	hrPrinterTable.
zhrmInfPrnLastIndex 1.3.6.1.4.1.169.3.180.1.5.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex for hrPrinterTable entries.	Integer without leading 0s.	hrPrinterTable.
zhrmInfPrnLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.5.4 read-only DisplayString	The system time at which hrPrinterTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfPrnLastStableRefresh 1.3.6.1.4.1.169.3.180.1.5.5 read-only DisplayString	The system time at which hrPrinterTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 8 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfPrnGetCounter 1.3.6.1.4.1.169.3.180.1.5.6 read-only Counter	The number of Get operations performed on hrPrinterTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPrnGetNextCounter 1.3.6.1.4.1.169.3.180.1.5.7 read-only Counter	The number of GetNext operations performed on hrPrinterTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPrnSetCounter 1.3.6.1.4.1.169.3.180.1.5.8 read-only Counter	The number of Set operations performed on hrPrinterTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDiskTable Group Objects:	Objects describing the status and usage of the hrDiskStorageTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfDiskEntries 1.3.6.1.4.1.169.3.180.1.6.1 read-only Gauge	The current number of entries in the hrDiskStorageTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDiskFirstIndex 1.3.6.1.4.1.169.3.180.1.6.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex for hrDiskStorageTable entries.	Integer without leading 0s.	hrDiskStorageTable.
zhrmInfDiskLastIndex 1.3.6.1.4.1.169.3.180.1.6.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex for hrDiskStorageTable entries.	Integer without leading 0s.	hrDiskStorageTable.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 9 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfDiskLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.6.4 read-only DisplayString	The system time at which hrDiskStorageTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfDiskLastStableRefresh 1.3.6.1.4.1.169.3.180.1.6.5 read-only DisplayString	The system time at which hrDiskStorageTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfDiskGetCounter 1.3.6.1.4.1.169.3.180.1.6.6 read-only Counter	The number of Get operations performed on hrDiskStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDiskGetNextCounter 1.3.6.1.4.1.169.3.180.1.6.7 read-only Counter	The number of GetNext operations performed on hrDiskStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfDiskSetCounter 1.3.6.1.4.1.169.3.180.1.6.8 read-only Counter	The number of Set operations performed on hrDiskStorageTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 10 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfPartTable Group Objects:	Objects describing the status and usage of the hrPartitionTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfPartEntries 1.3.6.1.4.1.169.3.180.1.7.1 read-only Gauge	The current number of entries in the hrPartitionTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPartFirstIndex 1.3.6.1.4.1.169.3.180.1.7.2 read-only INTEGER (1..65535)	The lowest value of hrDeviceIndex for hrPartitionTable entries.	Integer without leading 0s.	hrPartitionTable.
zhrmInfPartLastIndex 1.3.6.1.4.1.169.3.180.1.7.3 read-only INTEGER (1..65535)	The highest value of hrDeviceIndex for hrPartitionTable entries.	Integer without leading 0s.	hrPartitionTable.
zhrmInfPartLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.7.4 read-only DisplayString	The system time at which hrPartitionTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfPartLastStableRefresh 1.3.6.1.4.1.169.3.180.1.7.5 read-only DisplayString	The system time at which hrPartitionTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 11 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfPartGetCounter 1.3.6.1.4.1.169.3.180.1.7.6 read-only Counter	The number of Get operations performed on hrPartitionTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPartGetNextCounter 1.3.6.1.4.1.169.3.180.1.7.7 read-only Counter	The number of GetNext operations performed on hrPartitionTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfPartSetCounter 1.3.6.1.4.1.169.3.180.1.7.8 read-only Counter	The number of Set operations performed on hrPartitionTable objects since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfFSTable Group Objects:	Objects describing the status and usage of the hrFSTable.		Refer to “zhrmTableInfo Group Maintenance” later in this section for information on how object values are maintained.
zhrmInfFSEntries 1.3.6.1.4.1.169.3.180.1.8.1 read-only Gauge	The current number of entries in the hrFSTable.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfFSFirstIndex 1.3.6.1.4.1.169.3.180.1.8.2 read-only INTEGER (1..65535)	The lowest value of hrFSIndex.	Integer without leading 0s.	hrFSTable.
zhrmInfFSLastIndex 1.3.6.1.4.1.169.3.180.1.8.3 read-only INTEGER (1..65535)	The highest value of hrFSIndex.	Integer without leading 0s.	hrFSTable.

Table 11-12. zhrmTableInfo Group Objects Supported by Host Resources Subagent's MIB (page 12 of 12)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmInfFSLastDynamRefresh 1.3.6.1.4.1.169.3.180.1.8.4 read-only DisplayString	The system time at which hrFSTable values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were last refreshed.	`month` DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfFSLastStableRefresh 1.3.6.1.4.1.169.3.180.1.8.5 read-only DisplayString	The system time at which hrFSTable values controlled by the -s startup parameter or the zhrmStableRefresh Now object were last refreshed.	`month` DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmInfFSGetCounter 1.3.6.1.4.1.169.3.180.1.8.6 read-only Counter	The number of subagent accesses of hrFSTable objects to handle Get requests since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfFSGetNextCounter 1.3.6.1.4.1.169.3.180.1.8.7 read-only Counter	The number of subagent accesses of hrFSTable objects to handle GetNext requests since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmInfFSSetCounter 1.3.6.1.4.1.169.3.180.1.8.8 read-only Counter	The number of subagent accesses of hrFSTable objects to handle Set requests since the subagent was last started.	Integer without leading 0s.	Computed by Host Resources Subagent.

Sample Data

```
Time, zhrmInfStorEntries, zhrmInfStorFirstIndex, zhrmInfStorLastIndex,
zhrmInfStorLastDynamRefresh,
zhrmInfStorLastStableRefresh, zhrmInfStorGetCounter,
zhrmInfStorGetNextCounter, zhrmInfStorSetCounter

05/22/95 10:11:25, 14, 1, 10005, May 22, 1995 09:59:48,
May 21, 1995 17:20:47, 0, 106, 0

Time, zhrmInfDevEntries, zhrmInfDevFirstIndex, zhrmInfDevLastIndex,
zhrmInfDevLastDynamRefresh,
zhrmInfDevLastStableRefresh, zhrmInfDevGetCounter, zhrmInfDevGetNextCounter,
zhrmInfDevSetCounter

05/22/95 10:13:19, 15, 1, 10002, May 22, 1995 09:59:48,
May 21, 1995 17:20:47, 0, 111, 0

Time, zhrmInfCpuEntries, zhrmInfCpuFirstIndex, zhrmInfCpuLastIndex,
zhrmInfCpuLastDynamRefresh,
zhrmInfCpuLastStableRefresh, zhrmInfCpuGetCounter, zhrmInfCpuGetNextCounter,
zhrmInfCpuSetCounter

05/22/95 10:14:22, 4, 1, 4, May 22, 1995 17:20:40,
May 20, 1995 17:20:40, 0, 8, 0

Time, zhrmInfNetEntries, zhrmInfNetFirstIndex, zhrmInfNetLastIndex,
zhrmInfNetLastDynamRefresh,
zhrmInfNetLastStableRefresh, zhrmInfNetGetCounter, zhrmInfNetGetNextCounter,
zhrmInfNetSetCounter

05/22/95 10:15:23, 2, 101, 102, May 22, 1995 09:59:48,
May 21, 1995 17:20:46, 0, 1, 0

Time, zhrmInfPrnEntries, zhrmInfPrnFirstIndex, zhrmInfPrnLastIndex,
zhrmInfPrnLastDynamRefresh,
zhrmInfPrnLastStableRefresh, zhrmInfPrnGetCounter, zhrmInfPrnGetNextCounter,
zhrmInfPrnSetCounter

05/22/95 10:16:07, 2, 501, 502, May 22, 1995 09:59:48,
May 21, 1995 17:20:47, 0, 1, 0

Time, zhrmInfDiskEntries, zhrmInfDiskFirstIndex, zhrmInfDiskLastIndex,
zhrmInfDiskLastDynamRefresh,
zhrmInfDiskLastStableRefresh, zhrmInfDiskGetCounter,
zhrmInfDiskGetNextCounter, zhrmInfDiskSetCounter

05/22/95 10:16:47, 5, 1001, 1005, May 22, 1995 10:09:50,
May 21, 1995 17:20:47, 0, 1, 0

Time, zhrmInfPartEntries, zhrmInfPartFirstIndex, zhrmInfPartLastIndex,
zhrmInfPartLastDynamRefresh,
zhrmInfPartLastStableRefresh, zhrmInfPartGetCounter,
zhrmInfPartGetNextCounter, zhrmInfPartSetCounter

05/22/95 10:17:43, 5, 1, 5, May 22, 1995 10:09:51,
May 21, 1995 17:20:47, 0, 0, 0

Time, zhrmInfFSEntries, zhrmInfFSFirstIndex, zhrmInfFSLastIndex,
zhrmInfFSLastDynamRefresh,
zhrmInfFSLastStableRefresh, zhrmInfFSGetCounter, zhrmInfFSGetNextCounter,
zhrmInfFSSetCounter

05/22/95 10:18:22, 5, 1, 5, May 22, 1995 10:09:51,
May 21, 1995 17:20:47, 0, 0, 0
```

zhrmTableInfo Group Maintenance

Most zhrmTableInfo group values are derived from hrStorage group and hrDevice group values as Table 11-12 describes. Values in the zhrmTableInfo group are refreshed immediately after hrStorage group and hrDevice group values are refreshed. Refer to hrStorage Group Maintenance on page 11-34 and hrDevice Group Maintenance on page 11-50 for information about when hrStorage group and hrDevice group values are refreshed.

zhrmThreshold Group

This group, defined by HP, describes RAM storage areas and disk devices in the hrStorage group, highlighting devices whose usage has reached a critically high level. Several objects in this group let you control whether the subagent sends traps when device utilization is too high. The zhrmThreshold group consists of a two groups, one describing RAM and one describing disks. Each group contains several scalar objects as well as a table whose entries describe devices whose utilization is excessive.

Objects in the zhrmThreshold group are identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          private (4)
            enterprises (1)
              tandem (169)
                nonstopsystems (3)
                  zhrm (180)
                    zhrmThreshold (2) √
                      zhrmThrRam (1) √
                        zhrmThrRAMTotal (1) √
                        zhrmThrRAMUse (2) √
                        zhrmThrRAMPercentUse (3) √
                        zhrmThrRAMTrapEnable (4) √
                        zhrmThrRAMHighValue (5) √
                        zhrmThrRAMLowValue (6) √
                        zhrmThrRAMTable (7) √
                          zhrmThrRAMEntry (1) √
                            zhrmThrRAMIndex (1) √
                            zhrmThrRAMDescr (2) √
                            zhrmThrRAMAllocationUnits (3) √
                            zhrmThrRAMSize (4) √
                            zhrmThrRAMUsed (5) √
                            zhrmThrRAMAllocationFailures (6) √
                      zhrmThrDisk (2) √
                        zhrmThrDiskTotal (1) √
                        zhrmThrDiskUse (2) √
```

- zhrmThrDiskPercentUse (3) √
- zhrmThrDiskTrapEnable (4) √
- zhrmThrDiskHighValue (5) √
- zhrmThrDiskLowValue (6) √
- zhrmThrDiskTable (7) √
  - zhrmThrDiskEntry (1) √
    - zhrmThrDiskIndex (1) √
    - zhrmThrDiskDescr (2) √
    - zhrmThrDiskAllocationUnits (3) √
    - zhrmThrDiskSize (4) √
    - zhrmThrDiskUsed (5) √
    - zhrmThrDiskAllocationFailures (6) √

MIB Objects

Table 11-13 describes how the Host Resources Subagent supports objects in the zhrmThreshold group.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 1 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrRam Group Objects:	Objects describing the status and usage of RAM with entries in the hrStorageTable.		Refer to “zhrmThreshold Group Maintenance” later in this section for information on how object values are maintained.
zhrmThrRAMTotal 1.3.6.1.4.1.169.3.180.2.1.1 read-only Gauge	The total number of megabytes of RAM in the system.	Integer without leading 0s.	The value of hrMemorySize, expressed as megabytes instead of kilobytes.
zhrmThrRAMUse 1.3.6.1.4.1.169.3.180.2.1.2 read-only Gauge	The total number of megabytes of RAM currently being used.	Integer without leading 0s.	Sum of hrStorageUsed values for all RAM entries in the hrStorageTable.
zhrmThrRAMPercentUse 1.3.6.1.4.1.169.3.180.2.1.3 read-only Gauge	The percent of zhrmThrRAMTotal that is currently being used.	Integer without leading 0s.	Computed by subagent.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 2 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrRAMTrapEnable 1.3.6.1.4.1.169.3.180.2.1.4 read-write INTEGER { ramTrapsDisabled(0), ramTrapsEnabled(1) }	When set to 1 (ramTrapsEnabled) a trap (zhrmRAM Threshold) is sent whenever utilization of a RAM storage area reaches or exceeds zhrmThrRAMHigh Value or reaches or falls below zhrmThrRAMLow Value.	One of these values is displayed: ramTraps Disabled ramTrapsEnabled When setting these values, use 0 or 1. Refer to “Traps” later in this section for a description of the trap and its generation algorithm.	Initialized to 1 (ramTrapsEnabled) at startup, but can be changed from an SNMP manager.
zhrmThrRAMHighValue 1.3.6.1.4.1.169.3.180.2.1.5 read-write INTEGER (1..100)	The percent utilization of a RAM storage area at or above which zhrmThrRAMTable entries are created and a trap generated if zhrmThrRAMTrap Enable is set to 1.	A value of 1 through 100, 90 by default. A value of 100 suppresses trap generation (regardless of the value of zhrmThr RAMTrapEnable) and zhrmThrRAM Tableentry creation.	Set at startup by using the -h startup parameter. Can be changed by setting the value of this MIB object.
zhrmThrRAMLowValue 1.3.6.1.4.1.169.3.180.2.1.6 read-write INTEGER (1..100)	The percent utilization of a RAM storage area at or below which a trap is generated if zhrmThrRAMTrap Enable is set to 1.	A value of 1 through 100, 45 by default. A value of 0 suppresses trap generation, regardless of the value of zhrmThrRAMTrap Enable.	Set at startup by using the -i startup parameter. Can be changed by setting the value of this MIB object.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 3 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrRAMTable Objects	Entries describing RAM in the hrStorageTable whose usage has reached or exceeded the value of zhrmThrRAMHigh Value.		hrStorageTable.
zhrmThrRAMIndex 1.3.6.1.4.1.169.3.180.2.1.7.1.1 read-only INTEGER (1..2147483647)	An index value having a corresponding value (hrFSStorageIndex) in the hrFSTable.	Integer, numbered sequentially starting at 1.	The value of hrStorageIndex for the device.
zhrmThrRAMDescr 1.3.6.1.4.1.169.3.180.2.1.7.1.2 read-only DisplayString	Description of the type of storage represented by the entry.	The number and type of a processor: CPU*nn* (*processor-type*) running D*nn.nn*	The value of hrStorageDescr for the device.
zhrmThrRAMAllocationUnits 1.3.6.1.4.1.169.3.180.2.1.7.1.3 read-only INTEGER (1..2147483647)	The number of bytes allocated by the entry when storage is requested.	Integer without leading 0s, indicating the page size.	The value of hrStorageAllocation Units for the device.
zhrmThrRAMSize 1.3.6.1.4.1.169.3.180.2.1.7.1.4 read-only INTEGER (1..2147483647)	The capacity of the entry, in number of allocation units.	Integer without leading 0s. If the number exceeds 2,147,483,647, the value returned is -1D.	The value of hrStorageSize for the device.
zhrmThrRAMUsed 1.3.6.1.4.1.169.3.180.2.1.7.1.5 read-only INTEGER (1..2147483647)	The capacity of the entry, in number of allocation units.	Integer without leading 0s.	The value of hrStorageUsed for the device.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 4 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrRAMAllocationFailures 1.3.6.1.4.1.169.3.180.2.1.7.1.6 read-only Counter	The number of requests for storage by the entry that could not be honored because of insufficient space.	Integer without leading 0s, indicating the number of page-fault interrupts since the processor was loaded.	The value of hrStorageAllocation Failuresfor the device.
zhrmThrDisk Group Objects:	Objects describing the status and usage of disks with entries in the hrStorageTable. Entries describe all writable disks. For OSS, entries are added for the OSS filesets instead of disks.		Refer to “zhrmThreshold Group Maintenance” later in this section for information on how object values are maintained.
zhrmThrDiskTotal 1.3.6.1.4.1.169.3.180.2.2.1 read-only Gauge	The total number of megabytes of writable disk in the system.	Integer without leading 0s.	Sum of hrStorageSize values for all entries in the hrStorageTable, expressed as megabytes.
zhrmThrDiskUse 1.3.6.1.4.1.169.3.180.2.2.2 read-only Gauge	The total number of megabytes of writable disk currently being used.	Integer without leading 0s.	Sum of hrStorageUsed values for all entries in the hrStorageTable.
zhrmThrDiskPercentUse 1.3.6.1.4.1.169.3.180.2.2.3 read-only Gauge	The percent of zhrmThrDiskTotal that is currently being used.	Integer without leading 0s.	Computed by subagent.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 5 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrDiskTrapEnable 1.3.6.1.4.1.169.3.180.2.2.4 read-write INTEGER { diskTrapsDisabled(0), diskTrapsEnabled(1) }	When set to 1 (diskTrapsEnabled) a trap (zhrmDiskThreshold) is sent whenever utilization of a writable disk reaches or exceeds zhrmThrDiskHigh Value or reaches or falls below zhrmThrDiskLow Value.	One of these values is displayed: diskTraps-Disabled diskTrapsEnabled When setting these values, use 0 or 1. Refer to “Traps” later in this section for a description of the trap and its generation algorithm.	Initialized to 1 (diskTrapsEnabled) at startup, but can be changed from an SNMP manager.
zhrmThrDiskHighValue 1.3.6.1.4.1.169.3.180.2.2.5 read-write INTEGER (1..100)	The percent utilization of a writable disk at or above which zhrmThrDiskTable entries are created and a trap generated if zhrmThrDiskTrap Enable is set to 1.	A value of 1 through 100, 90 by default. A value of 100 suppresses trap generation (regardless of the value of zhrmThrDiskTrap Enable) and zhrmThrDiskTable entry creation.	Set at startup by using the -h startup parameter. Can be changed by setting the value of this MIB object.
zhrmThrDiskLowValue 1.3.6.1.4.1.169.3.180.2.2.6 read-write INTEGER (1..100)	The percent utilization of a writable disk at or below which a trap is generated if zhrmThrDiskTrap Enable is set to 1.	A value of 1 through 100, 45 by default. A value of 0 suppresses trap generation, regardless of the value of zhrmThrDiskTrap Enable.	Set at startup by using the -i startup parameter. Can be changed by setting the value of this MIB object.

Table 11-13. zhrmThreshold Group Objects Supported by Host Resources Subagent's MIB (page 6 of 6)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmThrDiskTable Objects	Entries describing writable disks in the hrStorageTable whose usage has reached or exceeded the value of zhrmThrDiskHigh Value.		hrStorageTable.
zhrmThrDiskIndex 1.3.6.1.4.1.169.3.180.2.2.7.1.1 read-only INTEGER (1..2147483647)	An index value having a corresponding value (hrFSStorageIndex) in the hrFSTable.	Integer, numbered sequentially starting at 1.	The value of hrStorageIndex for the device.
zhrmThrDiskDescr 1.3.6.1.4.1.169.3.180.2.2.7.1.2 read-only DisplayString	Description of the type of storage represented by the entry.	The name and type of the disk: $volume(*disk-type*)	The value of hrStorageDescr for the device.
zhrmThrDiskAllocationUnits 1.3.6.1.4.1.169.3.180.2.2.7.1.3 read-only INTEGER (1..2147483647)	The number of bytes allocated by the entry when storage is requested.	2048, indicating one page.	The value of hrStorageAllocation Units for the device.
zhrmThrDiskSize 1.3.6.1.4.1.169.3.180.2.2.7.1.4 read-only INTEGER (1..2147483647)	The capacity of the entry, in number of allocation units.	Integer without leading 0s. The number describes the pages formatted.	The value of hrStorageSize for the device.
zhrmThrDiskUsed 1.3.6.1.4.1.169.3.180.2.2.7.1.5 read-only INTEGER (1..2147483647)	The amount of allocated storage associated with the entry, in allocation units.	Integer without leading 0s.	The value of hrStorageUsed for the device.
zhrmThrDiskAllocationFailures 1.3.6.1.4.1.169.3.180.2.2.7.1.6 read-only Counter	The number of requests for storage by the entry that could not be honored because of insufficient space.	This value is not available, and 0 is returned.	The value of hrStorageAllocation Failures for the device.

Sample Data

```
Time, zhrmThrRAMTotal, zhrmThrRAMUse, zhrmThrRAMPercentUse,
zhrmThrRAMTrapEnable, zhrmThrRAMHighValue,
zhrmThrRAMLowValue

05/22/95 10:19:41, 64, 59, 92, ramTrapsEnabled, 90,
45

Time, zhrmThrRAMIndex, zhrmThrRAMDescr, zhrmThrRAMAllocationUnits,
zhrmThrRAMSize, zhrmThrRAMUsed,
zhrmThrRAMAllocationFailures

05/22/95 10:20:10, 1, CPU00 (NSR-L) running D20.06, 4096, 8192, 8169,
4047

Time, zhrmThrDiskTotal, zhrmThrDiskUse, zhrmThrDiskPercentUse,
zhrmThrDiskTrapEnable, zhrmThrDiskHighValue,
zhrmThrDiskLowValue

05/22/95 10:20:48, 4047, 2824, 69, diskTrapsEnabled, 90,
45

Time, zhrmThrDiskIndex, zhrmThrDiskDescr, zhrmThrDiskAllocationUnits,
zhrmThrDiskSize, zhrmThrDiskUsed,
zhrmThrDiskAllocationFailures

05/22/95 10:21:15, 1, $DSM-P(4230-1C), 2048, 316470, 311818,
0
05/22/95 10:21:15, 2, $DSM2-P(4220-1), 2048, 146726, 145449,
0
```

zhrmThreshold Group Maintenance

Most zhrmThreshold group values are derived from hrStorage group values as Table 11-13 describes. Values in the zhrmThreshold group are refreshed immediately after hrStorage group values are refreshed. Refer to hrStorage Group Maintenance on page 11-34 for information about when hrStorage group values are refreshed.

The values of zhrmThrRAMTrapEnable and zhrmnThrDiskTrapEnable are always set to 1 at subagent startup. The values of these objects do not change while the subagent is running unless you set them.

The values of the following objects are initialized at subagent startup in accord with the startup parameters indicated:

zhrmThrRAMHighValue (-h startup parameter)
zhrmThrRAMLowValue (-i startup parameter)
zhrmThrDiskHighValue (-h startup parameter)
zhrmThrDiskLowValue (-i startup parameter)

The values of these objects also do not change while the subagent is running unless you set them.

zhrmDevUnavail Group

This group, defined by HP, provides summary information about devices that are in a state that might require operator attention. MIB values reflect devices represented in the hrDevice group. One object in this group lets you control whether the subagent sends traps when the state of a device changes. The zhrmDevUnavail group consists of several scalar objects and one table (the zhrmDevUnTable). Objects in the zhrmDevUnavail group are identified by a check mark in the following list:

- iso (1)
  - identified-organization (3)
    - dod (6)
      - internet (1)
        - private (4)
          - enterprises (1)
            - tandem (169)
              - nonstopsystems (3)
                - zhrm (180)
                  - zhrmDevUnavail (3) √
                    - zhrmDevUnCurrTime (1) √
                    - zhrmDevUnUp (2) √
                    - zhrmDevUnDown (3) √
                    - zhrmDevUnOther (4) √
                    - zhrmDevUnTrapEnable (5) √
                    - zhrmDevUnTable (6) √
                      - zhrmDevUnEntry (1) √
                        - zhrmDevUnIndex (1) √
                        - zhrmDevUnType (2) √
                        - zhrmDevUnDescr (3) √
                        - zhrmDevUnID (4) √
                        - zhrmDevUnStatus (5) √
                        - zhrmDevUnErrors (6) √

MIB Objects

Table 11-14 describes how the Host Resources Subagent supports objects in the zhrmDevUnavail group.

Table 11-14. zhrmDevUnavail Group Objects Supported by Host Resources Subagent's MIB (page 1 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmDevUnCurrTime 1.3.6.1.4.1.169.3.180.3.1 read-only DisplayString	The current system time.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmDevUnUp 1.3.6.1.4.1.169.3.180.3.2 read-only Gauge	The total number of devices for which the value of hrDeviceStatus is 2 (running).	Integer without leading 0s.	Sum of entries in hrDeviceTable having specified status.
zhrmDevUnDown 1.3.6.1.4.1.169.3.180.3.3 read-only Gauge	The total number of devices for which the value of hrDeviceStatus is 5 (down).	Integer without leading 0s.	Sum of entries in hrDeviceTable having specified status.
zhrmDevUnOther 1.3.6.1.4.1.169.3.180.3.4 read-only Gauge	The total number of devices for which the value of hrDeviceStatus is 1 (unknown), 3 (warning), or 4 (testing).	Integer without leading 0s.	Sum of entries in hrDeviceTable having specified status.
zhrmDevUnTrapEnable 1.3.6.1.4.1.169.3.180.3.5 read-write INTEGER { devUnTrapsDisabled(0), devUnTrapsEnabled(1) }	When set to 1 (devUnTraps Enabled), a trap (zhrmTrapDevice StateChange) is sent whenever the value of hrDeviceStatus changes.	One of these values is displayed: devUnTrapsDisabled devUnTrapsEnabled When setting these values, use 0 or 1. Refer to “Traps” later in this section for a description of the trap.	Initialized to 1 (devUnTraps Enabled) at startup, but can be changed from an SNMP manager.
zhrmDevUnTable Objects:	Entries duplicating entries in the hrDeviceTable for devices that have an hrDeviceStatus value other than 2 (running).		Refer to “hrSWRunPerf Group Maintenance” later in this section for information on how entries are maintained.

Table 11-14. zhrmDevUnavail Group Objects Supported by Host Resources Subagent's MIB (page 2 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmDevUnIndex 1.3.6.1.4.1.169.3.180.3.6.1.1 read-only INTEGER (1..2147483647)	Index values that point to entries in the hrProcessorTable, the hrPrinterTable, the hrDiskStorageTable, the hrNetworkTable, and the hrPartitionTable.	Integers running sequentially from 1.	The value of hrDeviceIndex for qualifying devices.
zhrmDevUnType 1.3.6.1.4.1.169.3.180.3.6.1.2 read-only OBJECT IDENTIFIER	One of the following device types defined by RFC 1514 that characterizes the entry: hrDeviceProcessor hrDeviceNetwork hrDevicePrinter hrDeviceDiskStorage hrDeviceTape hrDeviceDiskStorage	For CPUs: 1.3.6.1.2.1.25.3.1.3. For printers: 1.3.6.1.2.1.25.3.1.5. For disks: 1.3.6.1.2.1.25.3.1.6. For tape drives: 1.3.6.1.2.1.25.3.1.18. For network devices: 1.3.6.1.2.1.25.3.1.4. For OSS: 1.3.6.1.4.1.169.3.180. 24.0	The value of hrDeviceType for qualifying devices.

Table 11-14. zhrmDevUnavail Group Objects Supported by Host Resources Subagent's MIB (page 3 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmDevUnDescr 1.3.6.1.4.1.169.3.180.3.6.1.3 read-only DisplayString (SIZE (0..64))	Description of the type of device represented by the entry.	For CPUs: the number and type: CPU*nn* (*processor-type*). For printers: name of a printer/spooler collector. Printers have name form $*printer-id*. Spooler collectors, used as pseudo printers, have name form [\\*node-name*]$*collector-process-name*[.#*group-name*]. For disks: name & type of disk: $*volume*(*disk-type*). For tapes: $*tape-drive-name.* For network devices: *$network*--*device.* For a loop back device: $NOIOP. For OSS: *fileset*	The value of hrDeviceDescr for qualifying devices.

Table 11-14. zhrmDevUnavail Group Objects Supported by Host Resources Subagent's MIB (page 4 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmDevUnID 1.3.6.1.4.1.169.3.180.3.6.1.4 read-only OBJECT IDENTIFIER	An identifier characterizing the entry as a particular device type and subtype.	The root of the identifier is the same for all entries: 1.3.6.1.4.1.169.3.180. Appended to this value are the type and subtype values. For example, the hrDeviceID for a 4500 (1038-MB formatted capacity per spindle) disk would look like this:1.3.6.1.4.1.169.3.180.3.22. Refer to the *Guardian Procedure Calls Reference Manual* for a complete	The value of hrDeviceID for qualifying devices.
zhrmDevUnStatus 1.3.6.1.4.1.169.3.180.3.6.1.5 read-only INTEGER { unknown(1), running(2), warning(3), testing(4), down(5) }	The current state of the device represented by the entry.	One of these values: unknown warning testing down	The value of hrDeviceStatus for qualifying devices.
zhrmDevUnErrors 1.3.6.1.4.1.169.3.180.3.6.1.6) read-only Counter	The number of errors detected on the device described by the entry.	Integer without leading 0s.	The value of hrDeviceErrors for qualifying devices.

Sample Data

```
Time, zhrmDevUnCurrTime, zhrmDevUnUp, zhrmDevUnDown, zhrmDevUnOther,
zhrmDevUnTrapEnable

05/22/95 10:25:57, May 22, 1995 10:18:51, 13, 2, 0, devUnTrapsEnabled

Time, zhrmDevUnIndex, zhrmDevUnType, zhrmDevUnDescr, zhrmDevUnID,
zhrmDevUnStatus, zhrmDevUnErrors

05/22/95 10:26:19, 1, 1.3.6.1.2.1.25.3.1.3, CPU02(unknown),
1.3.6.1.4.1.169.3.180.0.6, down, 1
05/22/95 10:26:19, 2, 1.3.6.1.2.1.25.3.1.3, CPU03(unknown),
1.3.6.1.4.1.169.3.180.0.6, down, 1
```

zhrmDevUnavail Group Maintenance

Most zhrmDevUnavail group values are derived from hrDevice group values as Table 11-14 describes. Values in the zhrmDevUnavail group are refreshed immediately after hrDevice group values are refreshed. Refer to hrDevice Group Maintenance on page 11-50 for information about when hrDevice group values are refreshed.

The value of zhrmDevUnTrapEnable is always set to 1 at subagent startup. The value does not change while the subagent is running unless you set it. If you change the value of this object while the subagent is running, the value remains where you last set it when values for the other objects in the group are refreshed.

zhrmSaProcess Group

This group, defined by HP, provides information about the subagent process and lets you change the priority at which it runs. It also lets you cause the subagent's backup process to take over and a new backup process to be created. This procedure offers a way to refresh all MIB values in one step. One of the objects in this group lets you enable or disable creation and maintenance of hrSWRun and hrSWRunPerf groups. The zhrmSaProcess group consists of a series of scalar objects. Objects in the zhrmSaProcess group are identified by a check mark in the following list:

```
iso (1)
        identified-organization (3)
            dod (6)
                internet (1)
                    private (4)
                        enterprises (1)
                            tandem (169)
                                nonstopsystems (3)
                                    zhrm (180)
                                        zhrmSaProcess (4) √
                                            zhrmSaProcCurrTime (1) √
                                            zhrmSaProcVersion (2) √
                                            zhrmSaProcName (3) √
                                            zhrmSaPaid (4) √
                                            zhrmSaPID (5) √
                                            zhrmSaCreatTime (6) √
                                            zhrmSaCpuTime (7) √
                                            zhrmSaPri (8) √
                                            zhrmSaHomeTerm (9) √
                                            zhrmSaIniFile (10) √
                                            zhrmSaHeapInitial (11) √
                                            zhrmSaHeapCurrent (12) √
                                            zhrmGetPDUsCounter (13) √
                                            zhrmGetNextPDUsCounter (14) √
                                            zhrmSetPDUsCounter (15) √
                                            zhrmTrapPDUsCounter (16) √
                                            zhrmEnableSoftwareGroup (17) √
                                            zhrmSwitchToBackupNow (18) √
                                            zhrmExamineGuardian (19) √
                                            zhrmExamineOSS (20) √
```

MIB Objects

Table 11-15 describes how the Host Resources Subagent supports objects in the zhrmSaProcess group.

Table 11-15. zhrmSaProcess Group Objects Supported by Host Resources Subagent's MIB (page 1 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmSaProcCurrTime 1.3.6.1.4.1.169.3.180.4.1 read-only DisplayString	The current system time.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmSaProcVersion 1.3.6.1.4.1.169.3.180.4.2 read-only DisplayString	The version of the Host Resources Subagent.	T8496D*nn_date* -- HP Host Resources Subagent Where D*nn* is the product version of the Host Resources Subagent; for example, D21.	Host Resources Subagent.
zhrmSaProcName 1.3.6.1.4.1.169.3.180.4.3 read-only DisplayString	The name of the subagent process.	$*process-name*, where *process-name* is from 1 through 5 alphanumeric characters.	The name specified in the TACL NAME option when the subagent was started.
zhrmSaPaid 1.3.6.1.4.1.169.3.180.4.4 read-only DisplayString	The process access ID (PAID) of the subagent process.	*group-number*,*user-number*	The group and user numbers associated with the user who started the subagent.
zhrmSaPID 1.3.6.1.4.1.169.3.180.4.5 read-only DisplayString	The process ID (PID) of the subagent process.	(*CPU-number*,*PIN-number*)	Host Resources Subagent.
zhrmSaCreatTime 1.3.6.1.4.1.169.3.180.4.6 read-only DisplayString	The system time at which the subagent process started.	started *month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmSaCpuTime 1.3.6.1.4.1.169.3.180.4.7 read-only DisplayString	The amount of CPU time used by the subagent since starting.	HH:MM:SS	Computed by Host Resources Subagent.
zhrmSaPri 1.3.6.1.4.1.169.3.180.4.8 read-write INTEGER	The priority of the subagent primary and backup processes.	Integer without leading 0s.	Initially, the value of the -n startup parameter (145 or a user-specified value). After startup, the value can be set from an SNMP manager.

Table 11-15. zhrmSaProcess Group Objects Supported by Host Resources Subagent's MIB (page 2 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmSaHomeTerm 1.3.6.1.4.1.169.3.180.4.9 read-only DisplayString	The name of the terminal from which the subagent was started.	\node.$*device-name*. #*subdevice-name*	Guardian procedure PROCESS_GETINFO_.
zhrmSaIniFile 1.3.6.1.4.1.169.3.180.4.10 read-only DisplayString	The name of the subagent's initialization file.	\node.$*volume*. *subvolume*.*file*	The name specified in the -e startup parameter. If not specified, the name HMSAINI.
zhrmSaHeapInitial 1.3.6.1.4.1.169.3.180.4.11 read-only INTEGER	The number of bytes of memory buffer space allocated at subagent startup.	Integer without leading 0s.	Internal library routine.
zhrmSaHeapCurrent 1.3.6.1.4.1.169.3.180.4.12 read-only INTEGER	The number of bytes of memory buffer space currently in use by the subagent.	Integer without leading 0s.	Internal library routine.
zhrmGetPDUsCounter 1.3.6.1.4.1.169.3.180.4.13 read-only Counter	The number of Get requests received from the SNMP agent since starting.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmGetNextPDUsCounter 1.3.6.1.4.1.169.3.180.4.14 read-only Counter	The number of GetNext requests received from the SNMP agent since starting.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmSetPDUsCounter 1.3.6.1.4.1.169.3.180.4.15 read-only Counter	The number of Set requests received from the SNMP agent since starting.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmTrapPDUsCounter 1.3.6.1.4.1.169.3.180.4.16 read-only Counter	The number of traps generated by the subagent since starting.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-15. zhrmSaProcess Group Objects Supported by Host Resources Subagent's MIB (page 3 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmEnableSoftwareGroup 1.3.6.1.4.1.169.3.180.4.17 read-write INTEGER { swValuesDisabled(0), swValuesEnabled(1) }	A switch that suppresses maintenance of hrSWRun and hrSWRunPerf group objects when set to 0.	One of these values is displayed: swValuesDisabled swValuesEnabled When setting these values, use 0 or 1.	At startup, the subagent sets this value to 1 (swValuesEnabled) unless the -w startup parameter is used. The -w startup parameter sets this value to 0.
zhrmSwitchToBackupNow 1.3.6.1.4.1.169.3.180.4.18 read-write INTEGER { autoProcessPairControl(0), forceBackupTakeover(1) }	A switch that causes the subagent's backup process to take over and a new backup process to be created when set to 1.	One of these values is displayed: autoProcessPairControl forceBackupTakeover When setting these values, use 0 or 1.	At startup, the subagent sets this value to 0 (autoProcessPairControl) but it can be changed from an SNMP manager if the subagent was started using the -b startup parameter.
zhrmExamineGuardian 1.3.6.1.4.1.169.3.180.4.19 read-write INTEGER { disabled(0), enabled(1) }	A switch that enables or disables monitoring of the Guardian personality.	One of these values is displayed: disabled enabled When setting these values, use 0 or 1.	If the -u option is specified but the -t option is not specified, this is set to 0. Otherwise it is set to 1 at startup.
zhrmExamineOSS 1.3.6.1.4.1.169.3.180.4.20 read-write INTEGER { disabled(0), enabled(1) }	A switch that enables or disables monitoring of the OSS personality.	One of these values is displayed: disabled enabled When setting these values, use 0 or 1.	If the -u option is specified, this is set to 1; otherwise, it is set to 0 at startup.

Sample Data

```
Time, zhrmSaProcCurrTime, zhrmSaProcVersion, zhrmSaProcName, zhrmSaPaid,
zhrmSaPID, zhrmSaCreatTime, zhrmSaCpuTime,
zhrmSaPri, zhrmSaHomeTerm, zhrmSaIniFile, zhrmSaHeapInitial,
zhrmSaHeapCurrent, zhrmGetPDUsCounter,
zhrmGetNextPDUsCounter, zhrmSetPDUsCounter, zhrmTrapPDUsCounter,
zhrmEnableSoftwareGroup, zhrmSwitchToBackupNow
```

```
05/22/95 10:28:36, May 22, 1995 10:21:30, T8496D20_06JUL95_28APR95 -- Tandem
Host Resources Subagent, $HMSA1, 255,255,
(0,47), started May 20, 1995 17:20:20.7, 0:28:50.582,
145, \SHERIFF.$ZTNT.#PTY4, \sheriff.$dsm.vinhms.hmsaini, 1536000, 279408, 0,
822, 2, 2, swValuesEnabled, autoProcessPairControl
```

zhrmSaProcess Group Maintenance

The zhrmSaProcess group values are updated:

- At subagent startup time. The values of MIB objects associated with startup parameters and startup process attributes are initialized.
- The counters that tally the requests handled by the subagent as well as the values of zhrmSaCpuTime and zhrmSaHeapCurrent are updated as requests are received.
- When a request is received from an SNMP manager to change the subagent's process priority. The value of zhrmSaPri does not persist after the subagent stops. The next time the subagent starts, this object is assigned the subagent priority specified at invocation.
- When a request is received from an SNMP manager to enable or disable software group value maintenance. When one of the refresh timers (-s or -d) runs down or when the value of zhrmStableRefreshNow or zhrmDynamicRefreshNow is set to 1:
    - A change from 1 to 0 deletes all the software group values.
    - A change from 0 to 1 creates software group values.
- When the value of zhrmSwitchToBackupNow is set to 1. If a backup process exists, the subagent's backup process takes over, refreshing all values in the subagent's MIB before handling SNMP manager requests. After takeover is complete, the subagent sets the value of zhrmSwitchToBackupNow to 0.
- When a request is received from an SNMP manager to enable or disable the monitoring of the NonStop Kernel Guardian or OSS personality.

zhrmRefresh Group

This group, defined by HP, provides information about MIB value refreshes and supports on-demand refreshes of MIB values. The zhrmRefresh group consists of a series of scalar objects. Objects in the zhrmRefresh group are identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                zhrm (180)
                                    zhrmRefresh (5) √
                                        zhrmRefreshCurrtime (1) √
                                        zhrmDynamRefreshInterval (2) √
                                        zhrmStableRefreshInterval (3) √
                                        zhrmLastDynamRefreshTime (4) √
                                        zhrmLastStableRefreshTime (5) √
                                        zhrmDynamRefreshCnt (6) √
                                        zhrmStableRefreshCnt (7) √
                                        zhrmDynamRefreshNow (8) √
                                        zhrmStableRefreshNow (9) √
```

MIB Objects

Table 11-16 describes how the Host Resources Subagent supports objects in the zhrmRefresh group.

Table 11-16. zhrmRefresh Group Objects Supported by Host Resources Subagent's MIB (page 1 of 2)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmRefreshCurrTime 1.3.6.1.4.1.169.3.180.5.1 read-only DisplayString	The current system time.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmDynamRefreshInterval 1.3.6.1.4.1.169.3.180.5.2 read-only INTEGER	The number of minutes assigned to the -d startup parameter when the subagent was started.	Integer without leading 0s.	Subagent -d startup parameter.
zhrmStableRefreshInterval 1.3.6.1.4.1.169.3.180.5.3 read-only INTEGER	The number of minutes assigned to the -s startup parameter when the subagent was started.	Integer without leading 0s.	Subagent -s startup parameter.
zhrmLastDynamRefreshTime 1.3.6.1.4.1.169.3.180.5.4 read-only DisplayString	The system time at which MIB values controlled by the -d startup parameter or the zhrmDynam RefreshNow object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmLastStableRefreshTime 1.3.6.1.4.1.169.3.180.5.5 read-only DisplayString	The system time at which MIB values controlled by the -s startup parameter or the zhrmStable RefreshNow object were last refreshed.	*month* DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
zhrmDynamRefreshCnt 1.3.6.1.4.1.169.3.180.5.6 read-only Counter	The number of times since the subagent was started that MIB values controlled by the -d startup parameter or the zhrmDynamRefresh Now object were refreshed.	Integer without leading 0s.	Computed by Host Resources Subagent.

Table 11-16. zhrmRefresh Group Objects Supported by Host Resources Subagent's MIB (page 2 of 2)

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmStableRefreshCnt 1.3.6.1.4.1.169.3.180.5.7 read-only Counter	The number of times since the subagent was started that MIB values controlled by the -s startup parameter or the zhrmStableRefresh Now object were refreshed.	Integer without leading 0s.	Computed by Host Resources Subagent.
zhrmDynamRefreshNow 1.3.6.1.4.1.169.3.180.5.8 read-write INTEGER { autoDynamicRefresh(0), forceDynamicRefresh(1) }	When set to 1 (force DynamicRefresh)MIB objects whose values are controlled by the -d startup parameter are refreshed.	One of these values is displayed: autoDynamic Refresh forceDynamicRefresh When setting these values, use 0 or 1.	Initialized to 0 (autoDynamicRefres h) at startup, but can be changed from an SNMP manager.
zhrmStableRefreshNow 1.3.6.1.4.1.169.3.180.5.9 read-write INTEGER { autoStableRefresh(0), forceStableRefresh(1) }	When set to 1 (forceStableRefresh), MIB objects whose values are controlled by the -s startup parameter are refreshed.	One of these values is displayed: autoStableRefresh forceStableRefresh When setting these values, use 0 or 1.	Initialized to 0 (autoStableRefresh) at startup, but can be changed from an SNMP manager.

Sample Data

```
Time, zhrmRefreshCurrTime, zhrmDynamRefreshInterval,
zhrmStableRefreshInterval, zhrmLastDynamRefreshTime,
zhrmLastStableRefreshTime, zhrmDynamRefreshCnt, zhrmStableRefreshCnt,
zhrmDynamRefreshNow, zhrmStableRefreshNow

05/22/95 10:29:55, May 22, 1995 10:23:01, 10, 1440, May 22, 1995 10:09:52,
May 21, 1995 17:20:48, 241, 2, autoDynamicRefresh, autoStableRefresh
```

zhrmRefresh Group Maintenance

The zhrmRefresh group values are updated:

- At subagent startup time.
- Whenever MIB values controlled by the -d and -s startup parameters are refreshed. Refer to Starting and Stopping the Subagent on page 11-12 for a summary of which MIB values are refreshed when the timers associated with these startup parameters run down.
- When a request is received from an SNMP manager to set the value of zhrmDynamRefreshNow or zhrmStableRefreshNow to 1. After refreshes occur, the subagent sets the value of these objects back to 0.

Traps

You can configure the Host Resources Subagent to generate traps when noteworthy hardware conditions occur. Three traps are defined in the subagent’s MIB:

- **zhrmTrapDeviceStateChange**. This trap identifies a cpu, printer, disk, or tape drive that might require operation attention.
- **zhrmRAMThreshold**. This trap identifies a CPU whose RAM storage area is experiencing critically high usage.
- **zhrmDiskThreshold**. This trap identifies a disk device or, for OSS, any file system, that is experiencing critically high usage.

These traps are defined in the zhrmTraps group of the subagent’s MIB. Variable bindings accompanied by a check mark are defined in this group. Variable bindings not accompanied by a check mark are defined in the zhrmThreshold group.

```
iso (1)
        identified-organization (3)
            dod (6)
                internet (1)
                    private (4)
                        enterprises (1)
                            tandem (169)
                                nonstopsystems (3)
                                    zhrm (180)
                                        zhrmTraps (7) √
                                            zhrmTrapDeviceStateIndex (1) √
                                            zhrmTrapDeviceDescr (2) √
                                            zhrmTrapDeviceStatus (3) √
                                            zhrmTrapDeviceStateChange (trap 1) √
                                                zhrmTrapDeviceStateIndex √
                                                zhrmTrapDeviceDescr √
                                                zhrmTrapDeviceStatus √
                                            zhrmTrapStorageThrIndex (4) √
                                            zhrmTrapStorageDescr (5) √

                                            zhrmRAMThreshold (trap 2) √
                                                zhrmTrapStorageThrIndex √
                                                zhrmTrapStorageDescr √
                                                zhrmThrRAMPercentUse
                                            zhrmDiskThreshold (trap 3) √
                                                zhrmTrapStorageThrIndex √
                                                zhrmTrapStorageDescr √
                                                zhrmThrDiskPercentUse
```

This subsection describes configuration options for these three traps and defines the variable bindings in each trap.

Routing Traps

Traps from the Host Resources Subagent are routed to SNMP managers identified in the SNMP agent's TRAPDEST objects. Refer to Configuring Trap Destinations on page 2-38 for more information on this topic.

Enabling Traps

By default, all subagent traps are enabled whenever the subagent is started.

You can disable zhrmRAMThreshold and zhrmDiskThreshold traps at startup by assigning these values to the -h and -i startup parameters:

```
-h 100
-i 0
```

When the subagent is running, you can set the value of several MIB objects to enable or disable traps:

zhrmDevUnTrapEnable	This zhrmDevUnavail group object enables or disables the zhrmTrapDeviceStateChange trap.
zhrmThrRAMTrapEnable	This zhrmThreshold group object enables or disables the zhrmRAMThreshold trap for RAM storage areas being utilized too much.
zhrmThrDiskTrapEnable	This zhrmThreshold group object enables or disables the zhrmDiskThreshold trap for disk devices being utilized too much.

Thresholds and Traps

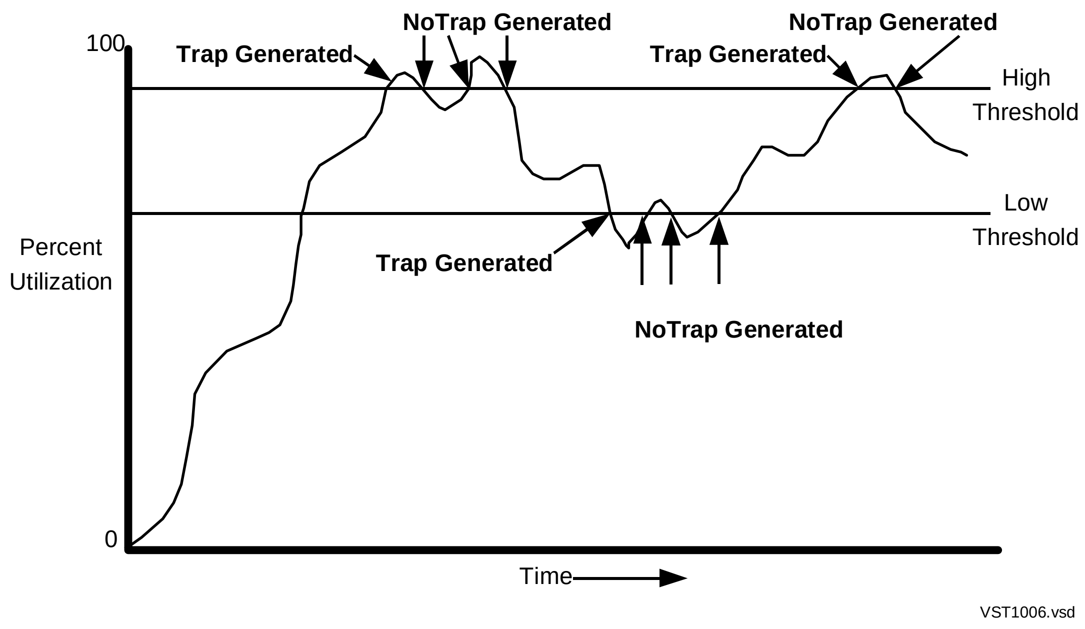
Two of the traps are generated when RAM or disk utilization reaches a high or low threshold value: zhrmRAMThreshold and zhrmDiskThreshold.

Figure 11-6 illustrates how the high and low thresholds regulate trap generation. When the subagent starts, the high and low thresholds are defined by the -h and -i startup parameters. As RAM or disk utilization fluctuates, these thresholds are deactivated, and reactivated and traps generated:

- When RAM or disk utilization reaches the high threshold value, a trap is generated. In addition, the high threshold is deactivated so that oscillations around it do not cause repeated trap generation.
- When RAM or disk utilization drops to the low threshold value, another trap is generated and the low threshold deactivated to prevent repeated trap generation as utilization fluctuates around the low threshold. The high threshold is reactivated.
- Because the high threshold is activated, another trap is generated when RAM or disk utilization reaches it. Then the high threshold is deactivated and the low threshold reactivated.

You can change the high and low threshold values while the subagent is running by setting the values of zhrmThrRAMHighValue and zhrmThrRAMLowValue (for RAM storage areas) and zhrmThrDiskHighValue and zhrmDiskLowValue (for disks) in the zhrmThreshold group. If a threshold is changed such that a trap would be generated at the current activity level, a trap is not generated until the activity level changes.

Figure 11-6. Relationship Between Threshold Values and Trap Generation



Trap PDU

The trap PDU contains these fields:

enterprise	The object identifier for the SNMP agent, indicating the origin of the trap: 1.3.6.1.4.1.169.3.155.1.
agent-address	The Internet address of the system on which the SNMP agent forwarding the trap is installed.
generic-trap	A 16-bit number set to 6 to signify that the trap is enterpriseSpecific.
specific-trap	A 16-bit number set to 0.
time-stamp	The value of sysUpTime, defined in Section 3, MIBs Supported by the SNMP Agent.
variable-bindings	The objects that the Host Resources Subagent includes in each trap, described in the following subsections.

zhrmTrapDeviceStateChange Trap

When zhrmDevUnTrapEnable is set to 1, this trap is generated whenever the status of any device represented in the hrDevice group changes. Table 11-17 describes the MIB objects constituting the variable bindings of this trap.

Table 11-17. Variable Bindings in the zhrmTrapDeviceStateChange Trap

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmTrapDeviceState Index 1.3.6.1.4.1.169.3.180.7.1 not-accessible INTEGER (1..2147483647)	Index value that points to the entry (in the hrProcessorTable, the hrPrinterTable, the hrDiskStorageTable, the hrNetworkTable, or the hrPartitionTable) for the device that the trap describes.	Integer, numbered sequentially starting at 1.	The value of zhrmDevUnIndex for the device.
zhrmTrapDeviceDescr 1.3.6.1.4.1.169.3.180.7.2 not-accessible DisplayString (SIZE (0..64))	A description of the type of device that the trap describes.	For printers: the name of a printer or Spooler collector. Printers have names of the form $printer-id. Spooler collectors, used as pseudo printers, have names of the form: `[\node- name] $collector-process-name[.#group-name].` For disks: the name and type of the disk: `$volume(disk-type).` For tapes: `$tape-drive-name.`	The value of zhrmDevUnDescr for the device.
zhrmTrapDeviceStatus 1.3.6.1.4.1.169.3.180.7.3 not-accessible INTEGER { unknown(1), running(2), warning(3), testing(4), down(5) }	The current state the device that the trap describes is in.	One of the following values: unknown warning testing down	The value of zhrmDevUnStatus for the device.

zhrmRAMThreshold Trap

When zhrmThrRAMTrapEnable is set to 1, this trap is generated whenever the utilization of any RAM storage area represented in the hrStorage group:

- Reaches or exceeds a high threshold, represented by the value of zhrmThrRAMHighValue in the zhrmThreshold group
- Reaches or falls below a low threshold, represented by the value of zhrmThrRAMLowValue in the zhrmThreshold group

Refer to Thresholds and Traps on page 11-100 for an explanation of the relationship between thresholds and trap generation.

Table 11-18 describes the MIB objects constituting the variable bindings of this trap.

Table 11-18. Variable Bindings in the zhrmRAMThreshold Trap

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmTrapStorageThrIndex 1.3.6.1.4.1.169.3.180.7.4 not-accessible INTEGER (1..2147483647)	Index value that points to the entry in the hrStorageTable for the device that the trap describes.	Integer, numbered sequentially starting at 1.	The value of zhrmThrRAMIndex for the RAM storage area.
zhrmTrapStorageDescr 1.3.6.1.4.1.169.3.180.7.5 not-accessible DisplayString (SIZE (0..64))	Description of the type of storage represented by the entry.	The number and type of a processor: CPU*nn*(processor-type) running D*nn*.*nn*.	The value of zhrmThrRAMDescr for the RAM storage area.
zhrmThrRAMPercentUse 1.3.6.1.4.1.169.3.180.2.1.3 read-only Gauge	The percent of zhrmThrRAMTotal that is currently being used.	Integer without leading 0s.	The value of zhrmThrRAMPercentUse for the RAM storage area.

zhrmDiskThreshold Trap

When zhrmThrDiskTrapEnable is set to 1, this trap is generated whenever the utilization of any disk, or, for OSS, file system, represented in the hrStorage group:

- Reaches or exceeds a high threshold, represented by the value of zhrmThrDiskHighValue in the zhrmThreshold group
- Reaches or falls below a low threshold, represented by the value of zhrmThrDiskLowValue in the zhrmThreshold group

Refer to Thresholds and Traps on page 11-100 for an explanation of the relationship between thresholds and trap generation.

Table 11-19 describes the MIB objects constituting the variable bindings of this trap.

Table 11-19. Variable Bindings in the zhrmDiskThreshold Trap

Object and Attributes	Definition	Format of Value	Derivation of Value
zhrmTrapStorageThrIndex 1.3.6.1.4.1.169.3.180.7.4 not-accessible INTEGER (1..2147483647)	Index value that points to the entry in the hrStorageTable for the device that the trap describes.	Integer, numbered sequentially starting at 1.	The value of zhrmThrDiskIndex for the device.
zhrmTrapStorageDescr 1.3.6.1.4.1.169.3.180.7.5 not-accessible DisplayString (SIZE (0..64))	Description of the type of storage represented by the entry.	The name and type of the disk: $`volume`(`disk-type`)	The value of zhrmThrDiskDescr for the device.
zhrmThrDiskPercentUse 1.3.6.1.4.1.169.3.180.2.2.3 read-only Gauge	The percent of zhrmThrDiskTotal that is currently being used.	Integer without leading 0s.	The value of zhrmThrDiskPercentUse for the device.

EMS Support

This subsection describes the Event Management Service (EMS) events generated by the Host Resources Subagent (subsystem abbreviation ZHRM). The events are listed in Table 11-20.

Table 11-20. Host Resources Subagent Event Messages

Number	Event	Standard Event
1	ZHRM-EVT-HRM-SA-TERMINATED	Object Unavailable
2	ZHRM-EVT-HRM-SA-STARTED	Object Available
3	ZHRM-EVT-HRM-SA-IO-ERR	Transient Fault
4	ZHRM-EVT-HRM-SA-NO-MEM-SPACE	Transient Fault
5	ZHRM-EVT-HRM-SA-PROG-ERR	Transient Fault

The Host Resources Subagent events are EMS standard events. Table 11-20 lists the standard event that corresponds to each ZHRM event. See the *EMS Manual* for a complete description of standard events.

The Host Resources Subagent sends these messages to EMS collector $0 on the local node.

Data Definitions

Because they are standard events, most of the tokens, structures, and values appearing in ZHRM events are defined in the ZSPI and ZEMS data definition files ZSPIDDL and ZEMSDDL and their associated language-specific files. Data elements defined by the Host Resources Subagent are in the file ZHRMDDL and associated language-specific files:

ZHRMDDL	Data Definition Language (DDL) definitions from which the language-specific definitions are derived
ZHRMC	Definitions for C programs
ZHRMCOB	Definitions for COBOL programs
ZHRMPAS	Definitions for Pascal programs
ZHRMTACL	Definitions for TACL programs
ZHRMTAL	Definitions for TAL programs

The complete set of SPI definition files is usually located in the ZSPIDEF subvolume of the Guardian operating system installation volume. See the *SPI Programming Manual* for information on the ZSPI data definitions and data definition files in general. See the *EMS Manual* for information on ZEMS data definitions.

Subsystem ID

The subsystem ID that the Host Resources Subagent uses to identify itself as the source of event messages is:

```
CONSTANT ZHRM-VAL-VERSION           VALUE  VERSION "D21".
CONSTANT ZSPI-SSN-ZHRM              VALUE  180.

DEFINITION ZHRM-VAL-SSID            TACL   SSID.
  02 z-filler  TYPE character 8     VALUE IS ZSPI-VAL-TANDEM.
  02 z-owner   TYPE ZSPI-DDL-CHAR8  REDEFINES z-filler.
  02 z-number  TYPE ZSPI-DDL-INT    VALUE IS ZSPI-SSN-ZHRM.
  02 z-version TYPE ZSPI-DDL-UINT   VALUE IS ZHRM-VAL-
VERSION.
END
```

Tokens in ZHRM Event Messages

This subsection lists the tokens that the subagent includes in the event messages it generates.

Host Resources Subagent (ZHRM) Tokens

Table 11-21 lists all the tokens defined by the subagent. These tokens are defined in the ZHRMDDL definition file and are described where they appear in the event message descriptions later in this section.

Table 11-21. ZHRM Tokens in ZHRM Event Messages

Token	Contents
`ZHRM-TKN-ERR-TEXT`	Additional error text
`ZHRM-TKN-FILE-NAME`	Name of file related to event
`ZHRM-TKN-FS-ERR`	File system error code
`ZHRM-TKN-SUBJ-HRM-SA`	Host Resources Subagent subject token

Standard SPI Tokens

Table 11-22 lists the SPI-defined tokens that are used by the subagent. The standard SPI tokens are defined in the ZSPIDDL definition file. See the *SPI Programming Manual* for more information on these tokens and related data definitions.

Table 11-22. ZSPI Tokens in ZHRM Event Messages

Token	Contents
ZSPI-TKN-MAX-FIELD-VERSION	Latest version of any extensible structured token in the buffer
ZSPI-TKN-SSID	ZHRM subsystem ID
ZSPI-TKN-USEDLEN	Bytes used in EMS buffer

Standard EMS Tokens

Table 11-23 lists all the EMS-defined tokens that are used by the subagent. These tokens are defined in the ZEMSDDL definition file. See the *EMS Manual* for more information on these tokens and related data definitions.

Table 11-23. ZEMS Tokens in ZHRM Event Messages (page 1 of 2)

Token	Contents
ZEMS-TKN-BATCHJOB-ID	Batch job ID of process
ZEMS-TKN-CONTENT-STANDARD	Type of standard event
ZEMS-TKN-CONTENT-USER	Type of user-defined event
ZEMS-TKN-CPU	CPU number of event sender
ZEMS-TKN-EMPHASIS	Critical/Noncritical event flag
ZEMS-TKN-EVENTNUMBER	Event number
ZEMS-TKN-GENTIME	Event generation time

Table 11-23. ZEMS Tokens in ZHRM Event Messages (page 2 of 2)

Token	Contents
ZEMS-TKN-LOGTIME	Event log time
ZEMS-TKN-NAME-MANAGER	Manager process name
ZEMS-TKN-NODENUM	System number of event sender
ZEMS-TKN-PIN	PIN of event sender
ZEMS-TKN-SUBJECT-MARK	Event subject marker
ZEMS-TKN-SUPPRESS-DISPLAY	Display/don’t display event flag
ZEMS-TKN-USERID	User ID of event sender
ZEMS-TKN-CHANGE-REASON	Object state change reason; private enumerations: ZHRM-VAL-CR-PROCESS-NOMEM ZHRM-VAL-CR-PROCESS-SIGABRT ZHRM-VAL-CR-PROCESS-SIGFPE ZHRM-VAL-CR-PROCESS-SIGILL ZHRM-VAL-CR-PROCESS-SIGSEGV ZHRM-VAL-CR-PROCESS-SIGSTK ZHRM-VAL-CR-PROCESS-SIGTERM ZHRM-VAL-CR-PROCESS-SIGTIMEOUT ZHRM-VAL-CR-PROCESS-STOPPED ZHRM-VAL-CR-PROCESS-UP
ZEMS-TKN-STATE-CURRENT	Current object state; private enumerations: ZHRM-VAL-HRM-SA-DEFINED ZHRM-VAL-HRM-SA-STARTED
ZEMS-TKN-STATE-PREVIOUS	Previous object state; private enumerations: ZHRM-VAL-HRM-SA-DEFINED ZHRM-VAL-HRM-SA-STARTED
ZEMS-TKN-TXFAULT-TYPE	Type of transient fault; private enumerations: ZHRM-VAL-TF-IO (I/O error) ZHRM-VAL-TF-MEM (no memory space) ZHRM-VAL-TF-PROG (internal error)

Event Message Descriptions

ZHRM event messages are described in order by event number. Each description includes:

- A list of tokens that are included in the event message. Tokens listed as “unconditional” always appear in the message. Tokens listed as “conditional” are included only under described conditions.

  Standard ZSPI and ZEMS tokens are listed only if they contain information that appears in the printed message text or if they contain ZHRM-defined values. See the *EMS Manual* for a complete list and descriptions of standard events and the common tokens they contain.

- The Event-Message text that is generated when the contents of the event message are displayed according to the message template defined in the file ZHRMTMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the message template source file SHRMTMPL.

- Descriptions of listed tokens or values.
- The cause of the event, the conditions that prompted the subagent to generate the event message.
- The effects associated with or resulting from the cause.
- Recovery procedures you can follow to solve the problem reported by the event message.
- An example of the formatted message.

001: ZHRM-EVT-HRM-SA-TERMINATED

The Host Resources Subagent terminated normally.

Unconditional Tokens	**Value**
`ZHRM-TKN-SUBJ-HRM-SA`	`<1>` *`Subagent process name`*
`ZEMS-TKN-EVENTNUMBER`	`<2> ZHRM-EVT-HRM-SA-TERMINATED`
`ZEMS-TKN-CHANGE-REASON`	`<3> ZHRM-VAL-CR-`*`reason`*

Conditional Tokens

`None`

Message Text

```
Object Unavailable - Host Resources Subagent process <1>,
event number: <2>, cause: <3>
```

`ZHRM-TKN-SUBJ-HRM-SA`

identifies the subject of the event (always the Host Resources Subagent process). The DDL heading of this token (“Host Resources Subagent process”) and the token’s value (the name of the process) are inserted in the message text.

`ZHRM-EVT-HRM-SA-TERMINATED`

is the event number (1). The DDL AS clause of this value (“Process Terminated”) appears in the message text following “event number:”

`ZHRM-VAL-CR-`*`reason`*

is one of the following causes. The DDL AS clause associated with this value is inserted in the message text following "cause:"

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS clause)**
ZHRM-VAL-CR-PROCESS-SIGILL	"Instruction failure"
ZHRM-VAL-CR-PROCESS-SIGABRT	"Process aborted"
ZHRM-VAL-CR-PROCESS-SIGFPE	"Arithmetic overflow"
ZHRM-VAL-CR-PROCESS-SIGSEGV	"Illegal address reference"
ZHRM-VAL-CR-PROCESS-SIGSTK	"Stack overflow"
ZHRM-VAL-CR-PROCESS-SIGTIMEOUT	"Loop-timer timeout"
ZHRM-VAL-CR-PROCESS-SIGTERM	"Termination"
ZHRM-VAL-CR-PROCESS-STOPPED	"Process stopped"
ZHRM-VAL-CR-PROCESS-NOMEM	"Memory Space Exhausted"

Cause. The subagent process terminated normally. (Abnormal termination is reported by the operating system.)

Effect. The backup process of the NonStop process pair should take over. If recovery is successful, an event announcing the takeover is generated.

Recovery. Examine the event logs for related events. Often event message 3 (ZHRM-EVT-HRM-SA-IO-ERR) is generated prior to this event. Refer to the message contents for a description of the associated problem. If you can determine the cause of the termination, correct it and, if necessary, restart the subagent process to allow continued monitoring of the host resources. If the termination resulted from incorrect specification of the agent process name, rerun the subagent using the -a startup parameter to specify the proper name.

Sample Message

```
95-03-10 16:34:20 \COMM.$HMSA     TANDEM.HRM.D21       000001 Object Unavailable -
                                                     Host Resources Subagent process
\COMM.$HMSA,
                                                     event number: Process Terminated,
                                                     cause: Process stopped
```

002: ZHRM-EVT-HRM-SA-STARTED

The Host Resources Subagent process has started.

Unconditional Tokens		**Value**
`ZHRM-TKN-SUBJ-HRM-SA`	`<1>`	*`Subagent process name`*
`ZEMS-TKN-EVENTNUMBER`	`<2>`	`ZHRM-EVT-HRM-SA-STARTED`
`ZEMS-TKN-CHANGE-REASON`	`<3>`	`ZHRM-VAL-CR-PROCESS-UP`
`ZEMS-TKN-STATE-PREVIOUS`	`<4>`	`ZHRM-VAL-HRM-SA-DEFINED`
`ZEMS-TKN-STATE-CURRENT`	`<5>`	`ZHRM-VAL-HRM-SA-STARTED`

Conditional Tokens

None

Message Text

```
Object Available - Host Resources Subagent process <1>, event
number: <2>, reason: <3>, previous state: <4>, current state:
<5>
```

`ZHRM-TKN-SUBJ-HRM-SA`

identifies the subject of the event (always the Host Resources Subagent process). The DDL heading of this token (“Host Resources Subagent process”) and the token’s value (the name of the process) are inserted in the message text.

`ZHRM-EVT-HRM-SA-STARTED`

is the event number (2). The DDL AS clause of this value (“Process Started”) is inserted in the message text following “event number:”

`ZHRM-VAL-CR-PROCESS-UP`

is the change reason. The DDL AS clause of this value (“Process up”) is inserted in the message text following “reason:”

`ZHRM-VAL-HRM-SA-DEFINED`

is the previous state. The DDL AS clause of this value (“not running“) is inserted in the message text following “previous state:”

`ZHRM-VAL-HRM-SA-STARTED`

is the current state. The DDL AS clauses of this value (“started”) is inserted in the message text following “current state:”

Cause. The Host Resources Subagent process has started.

Effect. The process is ready for service.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
95-03-10 16:33:58 \COMM.$HMSA     TANDEM.HRM.D21   000002 Object Available  -
                                               Host Resources Subagent process
\COMM.$HMSA,
                                               event number: Process Started,
                                               reason: Process up, previous state: not
                                               running, current state: started
```

003: ZHRM-EVT-HRM-SA-IO-ERR

The Host Resources Subagent process encountered an I/O error.

Unconditional Tokens	Value
`ZEMS-TKN-EVENTNUMBER`	<1> *Subagent process name*
`ZEMS-TKN-TXFAULT-TYPE`	<2> `ZHRM-EVT-HRM-SA-IO-ERR`
`ZHRM-TKN-SUBJ-HRM-SA`	<3> `ZHRM-VAL-TF-IO`

Conditional Tokens	Value
`ZHRM-TKN-FS-ERR`	<4> *error code and subcode*
`ZHRM-TKN-FILE-NAME`	<5> *file name*
`ZHRM-TKN-ERR-TEXT`	<6> *descriptive text*

Message Text

```
Transient Fault - Host Resources Subagent process <1>, event
number: <2>, fault type: <3>, FS error: <4>, subcode: <4>,
File Name: <5>, Additional Info: <6>
```

`ZHRM-TKN-SUBJ-HRM-SA`

identifies the subject of the event (always the Host Resources Subagent process). The DDL heading of this token (“Host Resources Subagent process”) and the token’s value (the name of the process) are inserted in the message text.

`ZHRM-EVT-HRM-SA-IO-ERR`

is the event number (3). The DDL AS clause of this value (“Process I/O error”) is inserted in the message text following “event number:”

`ZHRM-VAL-TF-IO`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“IO error”) is inserted in the message text following “fault type:”

ZHRM-TKN-FS-ERR

contains the file system error code and subcode, documented in the *Guardian Procedure Errors and Messages Manual*. The token is based on this structure:

```
DEFINITION ZHRM-DDL-FS-ERR.
  02 ERR-CODE      TYPE ZSPI-DDL-INT
                   HEADING 'error code'
  02 ERR-SUBCODE   TYPE ZSPI-DDL-INT
                   HEADING 'error subcode'
END.
```

ZHRM-TKN-FILE-NAME

contains the name of the file with which the subagent experienced an I/O error. This token is of type ZSPI-TYP-STRING.

ZHRM-TKN-ERR-TEXT

contains text describing the error. This token is of type ZSPI-TYP-STRING.

Cause. The subagent process encountered an I/O error. The cause is most likely the assignment of an incorrect agent process name at subagent startup or a stopped SNMP agent process.

Effect. The subagent cannot function.

Recovery. Investigate the reported error code and subcode and then correct the condition they describe. If the error code is 14 (file does not exist), the subagent could not open the SNMP agent process, and you should ensure that the SNMP agent is running or that you specified the correct agent process name when starting the subagent. For more detailed information, including recovery actions, refer to the *Guardian Procedure Errors and Messages Manual*.

If the subagent was running as a process pair when the I/O error occurred, the backup process takes over. This new primary process tries to establish communication with the SNMP agent. If the process cannot establish communication, it retries every 10 minutes for a period of 1 hour and then stops.

Sample Message

```
95-03-10 16:34:20 \COMM.$HMSA     TANDEM.HRM.D21     000003 Transient Fault -
                                                    Host Resources Subagent process
                                                    \COMM.$HMSA,
                                                    event number: Process I/O error,
                                                    fault type: IO error, FS error:
14 --
                                                    subcode: 0, File Name:
$zsnmp.#PEER,
                                                    Additional Info: Error in opening
agent
                                                    process
```

004: ZHRM-EVT-HRM-SA-NO-MEM-SPACE

The Host Resources Subagent process ran out of memory.

Unconditional Tokens	Value
`ZHRM-TKN-SUBJ-HRM-SA`	`<1>` *Subagent process name*
`ZEMS-TKN-EVENTNUMBER`	`<2> ZHRM-EVT-HRM-SA-NO-MEM-SPACE`
`ZEMS-TKN-TXFAULT-TYPE`	`<3> ZHRM-VAL-TF-MEM`

Conditional Tokens

None

Message Text

```
Transient Fault - Host Resources Subagent process <1>, event
number: <2>, fault type: <3>
```

`ZHRM-TKN-SUBJ-HRM-SA`

identifies the subject of the event (always the Host Resources Subagent process). The DDL heading of this token (“Host Resources Subagent process”) and the token’s value (the name of the process) are inserted in the message text.

`ZHRM-EVT-HRM-SA-NO-MEM-SPACE`

is the event number (4). The DDL AS clause of this value (“Process No Memory Space”) is inserted in the message text following “event number:”

`ZHRM-VAL-TF-MEM`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Memory full”) is inserted in the message text.

Cause. The subagent process ran out of memory. The most likely causes are a problem of internal resource management or insufficient swap space.

Effect. This error can prevent a part of the subagent from functioning. The subagent might be able to correct the situation or might subsequently fail. In the latter case, additional events would be generated.

Recovery. Stop the subagent if necessary. Try using another disk as a swap device. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
95-03-10 16:34:20 \COMM.$HMSA     TANDEM.HRM.D21      000004 Transient Fault -
                                                      Host Resources Subagent process
                                                      \COMM.$HMSA,
                                                      event number: Process No Memory
Space,
                                                      fault type: Memory full
```

005: ZHRM-EVT-HRM-SA-PROG-ERR

The Host Resources Subagent process experienced an internal or logic error.

Unconditional Tokens		Value
ZHRM-TKN-SUBJ-HRM-SA	<1>	*Subagent process name*
ZEMS-TKN-EVENTNUMBER	<2>	ZHRM-EVT-HRM-SA-PROG-ERR
ZEMS-TKN-TXFAULT-TYPE	<3>	ZHRM-VAL-TF-PROG

Conditional Tokens		Value
ZHRM-TKN-ERR-TEXT	<4>	*descriptive text*

Message Text

```
Transient Fault - Host Resources Subagent process <1>, event
number: <2>, fault type: <3>, error detail: <4>
```

`ZHRM-TKN-SUBJ-HRM-SA`

identifies the subject of the event (always the Host Resources Subagent process). The DDL heading of this token (“Host Resources Subagent process”) and the token’s value (the name of the process) are inserted in the message text.

`ZHRM-EVT-HRM-SA-PROG-ERR`

is the event number (5). The DDL AS clause of this value (“Process Internal Error”) is inserted in the message text following “event number:”

`ZHRM-VAL-TF-PROG`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Internal error”) is inserted in the message text.

`ZHRM-TKN-ERR-TEXT`

contains text describing the error. This token is of type ZSPI-TYP-STRING.

Cause. The subagent process experienced an internal or logic error.

Effect. The process might or might not be able to recover from the error. If the primary subagent process cannot recover, its backup process should take over. If recovery is successful, an event announcing the takeover is generated. If the error is not recoverable and the process terminates, additional events are generated.

Recovery. If the subagent primary and backup processes cannot recover from the error, you can try restarting the subagent. Regardless of whether the subagent is able to recover, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
95-03-10 16:34:20 \COMM.$HMSA     TANDEM.HRM.D21        000005 Transient Fault -
                                               Host Resources Subagent process
\COMM.$HMSA,
                                               event number: Process Internal Error,
                                               fault type: Internal error,
                                               error detail: Invalid State: 99
```

Related Operating System Event Messages

The Host Resources Subagent operates in the Guardian environment and relies on Guardian services to generate process-related events.

The following four Host Resource Subagent-related events are generated by the operating system:

Process Trap Event

A “Process Trap” event is reported for a process whenever the process failed because of a Guardian trap.

Process Abended Event

A “Process Abended” event is reported for a process whenever the process is abended (stopped abnormally).

Process Suspended Event

A “Process Suspended” event is reported for a process whenever the process is suspended.

Process Activated Event

A “Process Activated” event is reported for a process whenever the process is activated.

Converting Events to Traps

Any process that generates events, including the Host Resources Subagent, can have its events translated into traps by the EMS Trap Subagent, which uses an event filter known as a routing distributor filter to determine where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert into traps. Refer to Section 9, EMS Trap Subagent, for more information.

12 Trap Multiplexer Subagent/Manager

The Trap Multiplexer is both a manager and a subagent. As a manager, the Trap Multiplexer receives traps from network devices. It converts the traps into Event Management Service (EMS) events, which can be accessed by NonStop management applications or viewed on devices of different types. As a subagent, the Trap Multiplexer interacts with the SNMP agent to provide SNMP manager access to its MIB, which contains objects describing its trap conversion activities and its configuration attributes.

This section describes the Trap Multiplexer, its MIB, and the events it generates. At the end of this section a description of a sample application demonstrates how to programmatically access trap-related information from events the Trap Multiplexer generates when it receives traps.

Architectural Overview

As Figure 12-1 illustrates, the Trap Multiplexer interacts with TCP/IP processes to receive traps and with the SNMP agent to receive and respond to SNMP requests.

Trap-to-Event Conversion

In its manager role, the Trap Multiplexer interacts with TCP/IP processes to collect traps and with EMS to convert them into event messages.

The Trap Multiplexer collects traps from TCP/IP processes specified when you invoke it. The UDP (User Datagram Protocol) port used for trap receipt from each TCP/IP process is typically port 162, but this port number is configurable. In addition to collecting traps from network devices, the Trap Multiplexer can collect traps sent to the SNMP agent from SNMP subagents when a TCP/IP process from which the Trap Multiplexer collects traps has been configured as a TRAPDEST object for the SNMP agent. See Section 2, Installing and Configuring the SNMP Agent.

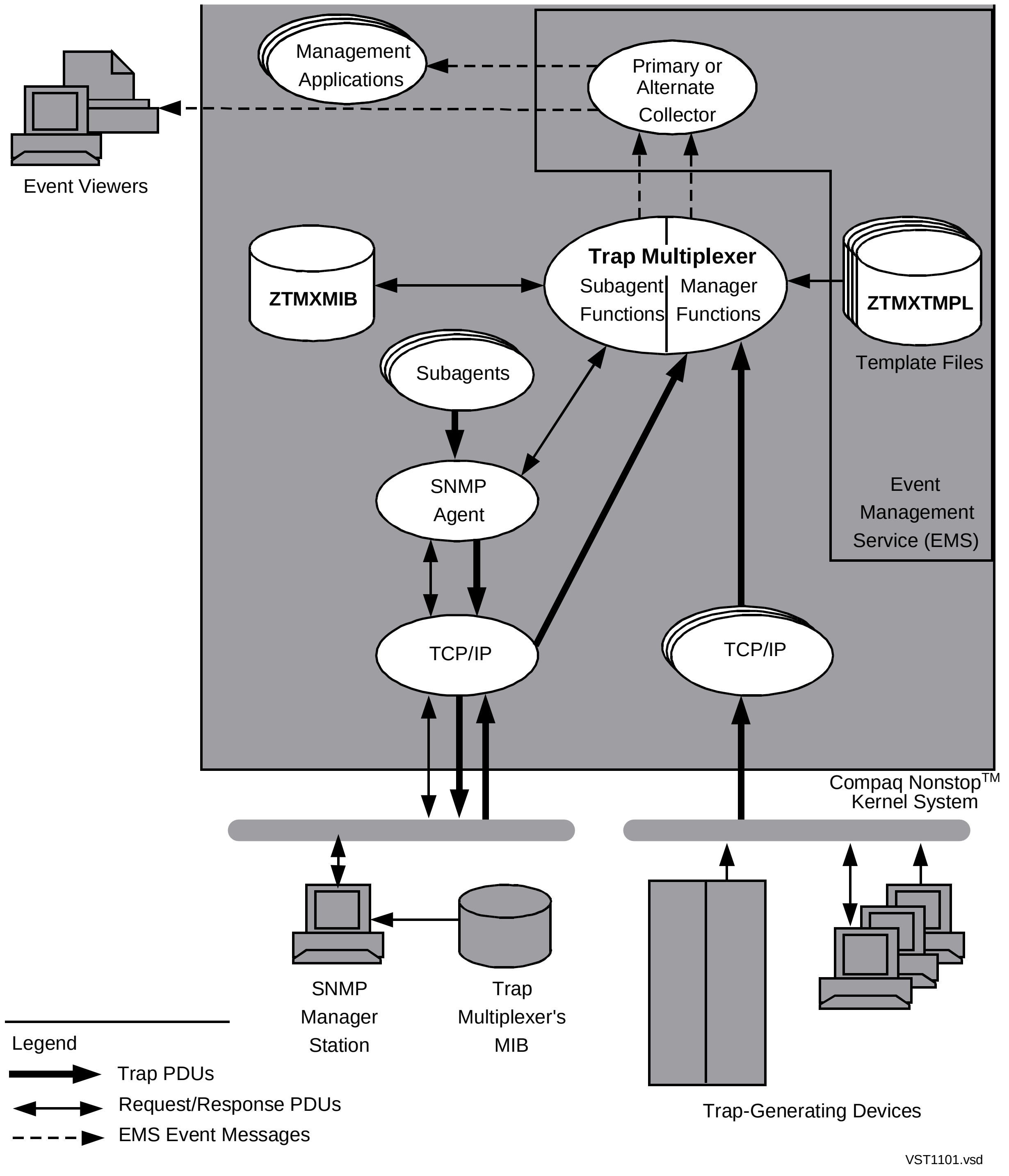
Although Figure 12-1 shows all TCP/IP processes on the same host as the Trap Multiplexer, they can actually be on any NonStop host with which the system on which the Trap Multiplexer is installed can communicate. Additionally, the Trap Multiplexer can receive traps from a TCP/IP process being used for request/response messages hosted by a SNMP agent process.

Using the EMS template in ZTMXTMPL, the Trap Multiplexer maps each trap into an EMS event, which is then forwarded to the EMS collector specified when the Trap Multiplexer is started. You can configure EMS to distribute events for access by management applications or for viewing at a terminal, printer, or console device. Refer to the *EMS Manual* for a complete description of how to configure and use EMS.

The subagent component of the Trap Multiplexer also generates events, which describe noteworthy conditions related to the Trap Multiplexer process.

How the Trap Multiplexer maps traps into EMS events and what each process-oriented and trap-oriented event and its tokens look like are topics in EMS Messages on page 12-21.

Figure 12-1. Trap Multiplexer Functions and Components



SNMP Request Processing

In its subagent role, the Trap Multiplexer provides a MIB that SNMP managers can access by routing requests through a SNMP agent that resides either on the same host as the Trap Multiplexer or on a remote node. The Trap Multiplexer's MIB, which resides in the subtree registered to Tandem, is organized into two groups, identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                ztmx (185)
                                    ztmxPDUStatistics (1) √
                                    ztmxProcess (2) √
```

The ztmxPDUStatistics group contains counts of the UDP and SNMP PDUs received by the Trap Multiplexer. The ztmxProcess group objects let you review and change configuration attributes of the Trap Multiplexer. Details about each of the objects in these groups can be found in ztmxPDUStatistics Group on page 12-12 and ztmxProcess Group on page 12-16.

Related Documents

The following documents provide information related to the Trap Multiplexer:

- The *EMS Manual* describes the Event Management Service (EMS). EMS is a collection of processes, tools, and interfaces that provide event-message collection and distribution in the Distributed Systems Management environment. The Trap Multiplexer converts SNMP traps into EMS event messages, by mapping elements of the trap PDU into EMS tokens.
- The *SPI Programming Manual* describes the standard Subsystem Programmatic Interface (SPI) tokens that appear in EMS events generated by the Trap Multiplexer. It also describes how to use SPI procedures to programmatically retrieve information from generated events. The sample application described at the end of this section illustrates how to embed SPI procedure calls in a C language program in order to retrieve information from events created from traps received by the Trap Multiplexer.

Installation

This subsection describes how to install the Trap Multiplexer.

Dependencies

The following products must be configured and running for the Trap Multiplexer to operate properly:

- SNMP agent (D22 or later)
- EMS (D21 or later)
- DSM Template Services (D21 or later)
- TCP/IP (D21 or later)

To access the Trap Multiplexer's MIB, you need at least one SNMP manager that can send requests to the SNMP agent for processing by the Trap Multiplexer. To generate trap events, you need at least one device or process that can send traps to the TCP/IP processes and port being monitored by the Trap Multiplexer.

Installation Steps

Run the Distributed Systems Management/Software Configuration Manager (DSM/SCM) program to install EMS, DSM Template Services, and the Trap Multiplexer. Refer to the *DSM/SCM User's Guide* for complete software installation information.

Ensure that the EMS file named ZTMXTMPL is installed in the system template file. When you use the SYSGEN utility to generate your system, this file is automatically installed in the system template file. Otherwise, you must manually install it. Refer to the *EMS Manual* for more information.

Load the ASN.1 source code for the Trap Multiplexer's MIB onto any SNMP manager you want to communicate with the subagent. The MIB definitions are contained in a file named ZTMXMIB. The Trap Multiplexer can communicate with a SNMP agent on either the same system or a remote system, but only one Trap Multiplexer process can communicate with any particular SNMP agent process.

Configure devices from which you want the Trap Multiplexer to collect traps so that traps are sent to the TCP/IP processes you want the Trap Multiplexer to monitor. The Trap Multiplexer can monitor TCP/IP processes on both the system where it resides and remote systems.

Configuration

You can configure the behavior of the Trap Multiplexer at startup time or from an SNMP manager after startup. Table 12-1 summarizes the available options.

The Trap Multiplexer attributes that can be controlled by setting MIB object values are reinitialized when the Trap Multiplexer is restarted to default values or values specified in startup parameters.

Refer to Starting and Stopping the Trap Multiplexer on page 12-6 for information on configuration using RUN startup parameters. Refer to subsequent subsections for each MIB group for information on settable MIB objects.

Table 12-1. Trap Multiplexer Configuration Options

Configuration Attribute	RUN Startup Parameter	Settable MIB Object Name (Group Name)	Default Behavior
SNMP agent process name	`-a`		$ZSNMP on the same system as the Trap Multiplexer specified at startup.
Trap Multiplexer process			
Priority	`-n`	ztmxProcPri (ztmxProcess group)	Priority is 145.
Backup process creation	`-b`		No backup process.
Backup process takeover		ztmxSwitchToBackupNow (ztmxProcess group)	Backup process takes over if primary process fails.
Run independent of SNMP agent	`-x`		TrapMux starts being associated with the agent.
Interval between connection attempts	`-k`	ztmxProcKeepAliveTimer	Every 15 minutes
Trap handling			
TCP/IP processes to monitor for traps	`-t`	zsmpTcpIpName zsmpTcpIpAddr	$ZTC0.
EMS collector process to receive events generated	`-c`		$0 on the same system as the Trap Multiplexer specified at startup.
UDP port to bind to	`-p`		Port 162.
Tracing			
To a terminal or Spooler destination	`-o`		No tracing.
To a disk file	`-l`		No tracing.

Starting and Stopping the Trap Multiplexer

To use the subagent functions of the Trap Multiplexer, the SNMP agent must be running before the Trap Multiplexer is started. If the agent is not running, start it with the RUN command:

```
RUN SNMPAGT /NAME $agent-process, NOWAIT/
```

For complete information on starting the SNMP agent, refer to Section 2, Installing and Configuring the SNMP Agent.

Start the Trap Multiplexer by entering the RUN command at a terminal that is running the TACL program. To monitor port 162 (the standard SNMP port for receiving traps), the Trap Multiplexer must be started by a member of the super user group (user ID 255, *n*). In addition, ensure that you specify the name of a running SNMP agent and that the TCP/IP processes and port you specify are not in use by another Trap Multiplexer process. Assuming the name of the SNMP agent is $ZSNMP, the following is an example of starting the Trap Multiplexer so that it receives traps on port 162 from the TCP/IP processes $ZTC0 and $ZTC1.

```
RUN SNMPTMUX /NAME $ZTMX/ -t ($ZTC0, $ZTC1)
```

The following is an example of starting the Trap Multiplexer so that it receives traps on port 162 from one or more TCP/IP processes with IP addresses.

```
RUN SNMPTMUX /NAME $ZTMX/ -t ($ZTC0, ($ZTC1:192.168.10.31))
```

The Trap Multiplexer can also be started with optional PARAMs. (See Using the -t and PARAM Startup Parameters on page 12-10.)

```
PARAM $ZTC2
PARAM 192.168.10.44
RUN SNMPTMUX /NAME $ZTMX/ -t ($ZTC0, ($ZTC1:192.168.10.31))
```

Syntax of the startup parameters follows established conventions used in the UNIX environment.

```
[RUN] [[$volume.]subvolume.]SNMPTMUX
   / NAME $trap-multiplexer-process [,other-run-option]... /
   [ ? | startup-parameter [startup-parameter]...]
```

volume

identifies the volume on which SNMPTMUX resides. You can omit it if SNMPTMUX resides on your current subvolume. By default, the installation program puts SNMPTMUX into $SYSTEM.

subvolume

identifies the subvolume on which SNMPTMUX resides. You can omit it if it is named in your TACL #PMSEARCHLIST. By default, the installation program puts SNMPTMUX into SYS*nn*.

trap-multiplexer-process

identifies the Trap Multiplexer process. You can specify from one through five alphanumeric characters, but the first character must be alphabetic.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the NOWAIT option so that you can resume TACL operations once the subagent is started.

?

displays help information.

startup-parameter

is one of the following parameters you can specify to control attributes of the Trap Multiplexer process.

```
-a $agent-process
-b [backup-cpu-number]
-c $alternate-collector-process
-n process-priority
-p trap-port-number
-t ([tcpip-proc-name ]... )
-k [keep-alive-timer]
-o
-l log-file
-x
```

-a $*agent-process*

is the name of the SNMP agent process from which you want the Trap Multiplexer to receive SNMP manager requests. If not specified, $ZSNMP on the same system as the Trap Multiplexer you specify is assumed.

-b [*backup-cpu-number*]

specifies that the Trap Multiplexer is to run as a process pair and optionally specifies a backup CPU number (0 through 15) identifying the CPU in which you want the backup process to run. If the primary process fails, the backup process automatically takes over. You can force the backup process to take over and a new backup process to be created by setting the value of ztmxSwitchToBackupNow in the ztmxProcess group from an SNMP manager after the subagent is started.

-c $*alternate-collector-process*

identifies the EMS collector to which the Trap Multiplexer sends event messages. The default is $0, the primary EMS collector, on the local node.

`-n` *`process-priority`*

sets the initial process priority of the Trap Multiplexer. If this parameter is not specified, the default initial priority is 145. If the Trap Multiplexer is run as a process pair, this priority applies to both primary and backup processes.

You can change the Trap Multiplexer process priority after it is started by setting the value of ztmxProcPri in the ztmxProcess group.

`-p` *`trap-port-number`*

specifies the UDP port number to monitor for traps. The port number you specify applies to all TCP/IP processes specified in the -t startup parameter.

The default value is 162, the SNMP standard port for receiving traps. If a nonprivileged port number (greater than 1023) is used, the Trap Multiplexer process need not be started by a super user group (255, *n*) member.

There are two ways to associate the Trap Multiplexer process with the super user group. From a TACL prompt:

- Log on using a super group user ID before starting the Trap Multiplexer.
- Use the File Utility Program (FUP) to give ownership of the Trap Multiplexer program file (SNMPTMUX) to a super user ID. Then secure SNMPTMUX so that a user who does not have a super group user ID can execute it, and set the PROGID so that the owner ID of SNMPTMUX is used as the creator accessor ID when the program is run. Refer to the *File Utility Program (FUP) Reference Manual* for more information.

`-t ([`*`tcpip-proc-name`*` ] [,`*`tcpip-proc-name`*` ] ... )`

identifies one or more TCP/IP processes (with or without IP addresses) from which you want the Trap Multiplexer to receive traps. If you omit this startup parameter, TCP/IP process $ZTC0 on the local node is assumed. You can specify *`tcpip-proc-name`* in the following formats:

```
$tcpip-proc-name
($tcpip-proc-name:ip-address1 [,ip-address2 ] )
ip-address
```

You can use this startup parameter with or without the startup PARAMs. (See Using the -t and PARAM Startup Parameters on page 12-10.) You can specify the -t parameter, the PARAMs, or both. If you use the PARAMs, you must enter both both the ZSMP^TCPIP^NAME and the ZSMP^TCPIP^ADDR PARAMs. Otherwise they are ignored.

`-k` *`keep-alive-timer`*

specifies how often you want the subagent to try to reestablish connections with a TCP/IP process when connections have failed. The *`keep-alive-timer`* interval is the number of seconds between attempts.

If you omit the -k startup parameter, the subagent tries to establish lost connections every 15 minutes.

To change the time interval after the Trap Multiplexer is started, set the value of ztmxProcKeepAliveTimer in the ztmxProcess group.

`-o`

causes the Trap Multiplexer to write a formatted trace of its activities to the terminal or to a Spooler destination. By default, trace information is sent to the terminal from which the Trap Multiplexer was started. To send the information to a Spooler destination, specify the destination as a TACL RUN option:

```
RUN SNMPTMUX /NAME $ZTMX, OUT $S.#OUT, NOWAIT/ -o
```

Spooler information is formatted as a C language-compatible file (file code 180), which you can convert to an edit file (file code 101) or use directly from any utility program.

If you want a second copy of the log file, you can also use the -l startup parameter.

Use of this startup parameter slows Trap Multiplexer processing time because a high volume of trace information is produced. In addition, the Trap Multiplexer might miss some traps.

`-l` *`log-file`*

causes the Trap Multiplexer to write a formatted trace of its activities to a C language-compatible disk file (file code 180), which you can convert to an edit file (file code 101) or use directly from any utility program.

If you want a second copy of the log file, you can also use the -o startup parameter.

Use of this startup parameter slows Trap Multiplexer processing time because a high volume of trace information is produced. In addition, the Trap Multiplexer might miss some traps.

`-x`

enables Trap Multiplexer to run independently of the SNMP agent. This option has priority over the -a start-up option. If you specify both options, the -a option is ignored.

Using this option suppresses the access to the MIBs maintained by Trap Multiplexer and allows it to receive only the traps.

To stop the Trap Multiplexer process, provide its name in the TACL STOP command:

```
STOP $ZTMX
```

Using the -t and PARAM Startup Parameters

For Parallel Library TCP/IP compatibility, you can specify IP addresses along with the TCP/IP process names. The format is:

`-t ([`*`tcpip-proc-name`*` ] [,`*`tcpip-proc-name`*` ] ... )`

where *`tcpip-proc-name`* is:

> `$`*`tcpip-process-name`*
>
> `($`*`tcpip-process-name:ip-address1`*` [, `*`ip-address2`*` ] )`
>
> *`ip-address`*

If the -t parameter specifies:

- A TCP/IP process and one or more IP addresses, the IP addresses are associated with the specified TCP/IP process.
- Only a TCP/IP process name, the TCP/IP process name is associated with all subnets for the TCP/IP subsystem.
- Only an IP address, the IP address is associated with the default TCP/IP process, $ZTC0.

You can also use PARAMs with the -t parameter to specify additional TCP/IP processes. You must specify both PARAMs. Otherwise, the PARAMs are ignored.

Syntax of the startup PARAMs is:

```
PARAM ZSMP^TCPIP^NAME tcpip-proc-name
PARAM ZSMP^TCPIP^ADDR ip-address
```

`tcpip-proc-name`

specifies a TCP/IP process from which you want the Trap Multiplexer to receive traps.

`ip-address`

specifies the IP address of the TCP/IP process from which you want the Trap Multiplexer to receive traps.

Examples

Using Only -t Startup Parameter

```
RUN SNMPTMUX /NAME $ZTMUX/ -t ($ZTC0, ($ZTC1:192.168.10.31))
RUN SNMPTMUX /NAME $ZTMUX/ -t ($ZTC0, ($ZTC1:192.168.10.42,192.168.10.43))
RUN SNMPTMUX /NAME $ZTMUX/ -t 192.168.12.4
```

Using Only PARAMs

```
PARAM ZSMP^TCPIP^NAME $ztc2
PARAM ZSMP^TCPIP^ADDR 192.168.10.44
RUN SNMPTMUX /NAME $ZTMUX/
```

Using Both -t Startup Parameter and PARAMs

```
PARAM ZSMP^TCPIP^NAME $ZTC1
PARAM ZSMP^TCPIP^ADDR 192.168.10.44
RUN SNMPTMUX /NAME $ZTMUX/ - T ($ZTC0, ($ZTC1:192.168.10.31))
```

```
PARAM ZSMP^TCPIP^NAME $ZTC3
PARAM ZSMP^TCPIP^ADDR 192.168.10.44
RUN SNMPTMUX /NAME $ZTMUX/ -T ($ZTC0, ($ZTC1:192.168.10.42,192.168.10.43))
```

ztmxPDUStatistics Group

The ztmxPDUStatistics group contains a collection of scalar objects describing the PDUs processed by the Trap Multiplexer. It also contains a group (ztmxTrapStatistics) of objects providing counts of the various kinds of traps received. The ztmxPDUStatistics group contains the objects identified by a check mark in the following list:

```
iso (1)
    identified-organization (3)
        dod (6)
            internet (1)
                private (4)
                    enterprises (1)
                        tandem (169)
                            nonstopsystems (3)
                                ztmx (185)
                                    ztmxPDUStatistics (1)
                                        ztmxUdpStatsCurrTime (1) √
                                        ztmxUdpInDatagrams (2) √
                                        ztmxInDecodeErrors (3) √
                                        ztmxInGetPdus (4) √
                                        ztmxInGetNextPdus (5) √
                                        ztmxInSetPdus (6) √
                                        ztmxInGetResponsePdus (7) √
                                        ztmxInTrapPdus (8) √
                                        ztmxTrapStatistics (9) √
                                            ztmxTrapStatsCurrTime (1) √
                                            ztmxInColdStartTraps (2) √
                                            ztmxInWarmStartTraps (3) √
                                            ztmxInLinkDownTraps (4) √
                                            ztmxInLinkUpTraps (5) √
                                            ztmxInAuthFailTraps (6) √
                                            ztmxInEgpNeighborLossTraps (7) √
                                            ztmxInEnterpriseSpecificTraps (8) √
```

MIB Objects

Table 12-2 describes each object in the ztmxPDUStatistics group.

Table 12-2. ztmxPDUStatistics Group Objects Supported by Trap Multiplexer's MIB (page 1 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxUdpStatsCurrTime 1.3.6.1.4.1.169.3.185.1.1 read-only DisplayString	The system time when a Get operation is performed for this object.	`month` DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
ztmxUdpInDatagrams 1.3.6.1.4.1.169.3.185.1.2 read-only Counter	The total number of UDP datagrams delivered to the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxInDecodeErrors 1.3.6.1.4.1.169.3.185.1.3 read-only Counter	The total number of UDP datagrams delivered to the Trap Multiplexer that could not be interpreted as SNMP Version 1 PDUs.	Integer without leading 0s.	Trap Multiplexer.
ztmxInGetPdus 1.3.6.1.4.1.169.3.185.1.4 read-only Counter	The total number of Get PDUs received by the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxInGetNextPdus 1.3.6.1.4.1.169.3.185.1.5 read-only Counter	The total number of GetNext PDUs received by the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxInSetPdus 1.3.6.1.4.1.169.3.185.1.6 read-only Counter	The total number of Set PDUs received by the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxInGetResponsePdus 1.3.6.1.4.1.169.3.185.1.7 read-only Counter	The total number of GetResponse PDUs received by the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxInTrapPdus 1.3.6.1.4.1.169.3.185.1.8 read-only Counter	The total number of Trap PDUs received by the Trap Multiplexer.	Integer without leading 0s.	Trap Multiplexer.
ztmxTrapStatistics Group Objects:	Objects providing a count of the various kinds of traps received.		

Table 12-2. ztmxPDUStatistics Group Objects Supported by Trap Multiplexer's MIB (page 2 of 3)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxTrapStatsCurrTime 1.3.6.1.4.1.169.3.185.1.9.1 read-only DisplayString	The system time when a Get operation is performed for this object.	`month` DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
ztmxInColdStartTraps 1.3.6.1.4.1.169.3.185.1.19.2 read-only Counter	The total number of coldStart traps received by the Trap Multiplexer. These standard SNMP traps signify that the sending entity has reinitialized, most likely because of a restart due to an error condition.	Integer without leading 0s.	Trap Multiplexer.
ztmxInWarmStartTraps 1.3.6.1.4.1.169.3.185.1.9.3 read-only Counter	The total number of warmStart traps received by the Trap Multiplexer. These standard SNMP traps signify that the sending entity has reinitialized, most likely because of a routine restart.	Integer without leading 0s.	Trap Multiplexer.
ztmxInLinkDownTraps 1.3.6.1.4.1.169.3.185.1.9.4 read-only Counter	The total number of linkDown traps received by the Trap Multiplexer. These standard SNMP traps signify that the sending entity has experienced a failure in one of its communication links.	Integer without leading 0s.	Trap Multiplexer.
ztmxInLinkUpTraps 1.3.6.1.4.1.169.3.185.1.9.5 read-only Counter	The total number of linkUp traps received by the Trap Multiplexer. These standard SNMP traps signify that one of the sending entity's communication links has become activated.	Integer without leading 0s.	Trap Multiplexer.

Table 12-2. ztmxPDUStatistics Group Objects Supported by Trap Multiplexer's MIB (page 3 of 3)

Object and Attributes	**Definition**	**Format of Value**	**Derivation of Value**
ztmxInAuthFailTraps 1.3.6.1.4.1.169.3.185.1.9.6 read-only Counter	The total number of authenticationFailure traps received by the Trap Multiplexer. These standard SNMP traps signify that the sending entity has received a request that failed authentication.	Integer without leading 0s.	Trap Multiplexer.
ztmxInEgpNeighborossTraps 1.3.6.1.4.1.169.3.185.1.9.7 read-only Counter	The total number of egpNeighborLoss traps received by the Trap Multiplexer. These standard SNMP traps signify that the sending entity no longer has a peer relationship with an EGP (External Gateway Protocol) neighbor.	Integer without leading 0s.	Trap Multiplexer.
ztmxInEnterpriseSpecificTraps 1.3.6.1.4.1.169.3.185.1.9.8 read-only Counter	The total number of enterpriseSpecific traps received by the Trap Multiplexer. These traps signify that the sending entity has experienced some enterprise-defined event.	Integer without leading 0s.	Trap Multiplexer.

ztmxPDUStatistics Group Maintenance

The values of the counters in this group updated as PDUs are received by the Trap Multiplexer.

ztmxProcess Group

This group provides information about the Trap Multiplexer process. It also lets you change the priority at which the process runs and force a backup process to take over on demand. The ztmxProcess group consists of a series of scalar objects, identified by a check mark in the following list:

- iso (1)
  - identified-organization (3)
    - dod (6)
      - internet (1)
        - private (4)
          - enterprises (1)
            - tandem (169)
              - nonstopsystems (3)
                - ztmx (185)
                  - ztmxProcess (2)
                    - ztmxProcCurrTime (1) √
                    - ztmxProcVersion (2) √
                    - ztmxProcName (3) √
                    - ztmxProcPaid (4) √
                    - ztmxProcPrimPID (5) √
                    - ztmxProcBkupPID (6) √
                    - ztmxProcCreatTime (7) √
                    - ztmxProcCpuTime (8) √
                    - ztmxProcPri (9) √
                    - ztmxProcHomeTerm (10) √
                    - ztmxProcHeapInitial (11) √
                    - ztmxProcHeapCurrent (12) √
                    - ztmxProcEmsCollector (13) √
                    - ztmxProcAgent (14) √
                    - ztmxProcAgentSessionStatus (15) √
                    - ztmxProcUdpPort (16) √
                    - ztmxProcTcpIpProcess (17) √
                    - ztmxSwitchToBackupNow (18) √
                    - ztmxProcProgramFileName (19) √
                    - ztmxProcEmsWriteTotal (20) √
                    - ztmxProcEmsWriteErrors (21) √
                    - ztmxProcKeepAliveTimer (22) √

MIB Objects

Table 12-3 describes each object in the ztmxProcess group.

Table 12-3. ztmxProcess Group Objects Supported by Trap Multiplexer's MIB (page 1 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxProcCurrTime 1.3.6.1.4.1.169.3.185.2.1 read-only DisplayString	The current system time.	month DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
ztmxProcVersion 1.3.6.1.4.1.169.3.185.2.2 read-only DisplayString	The version of the Trap Multiplexer.	T1041D*nn*_date -- HP SNMP Trap Multiplexer Where D*nn* is the product version of the Trap Multiplexer; for example, D20.	Trap Multiplexer.
ztmxProcName 1.3.6.1.4.1.169.3.185.2.3 read-only DisplayString	The name of the Trap Multiplexer process.	*$process-name*, where *process-name* is from 1 through 5 alphanumeric characters.	The name specified in the TACL NAME option when the Trap Multiplexer was started.
ztmxProcPaid 1.3.6.1.4.1.169.3.185.2.4 read-only DisplayString	The process access ID (PAID) of the Trap Multiplexer process.	*group-number*,*user-number*	The group and user numbers associated with the user who started the Trap Multiplexer.
ztmxProcPrimPID 1.3.6.1.4.1.169.3.185.2.5 read-only DisplayString	The process ID (PID) of the Trap Multiplexer primary process.	(*CPU-number,PIN-number*)	Trap Multiplexer.
ztmxProcBkupPID 1.3.6.1.4.1.169.3.185.2.6 read-only DisplayString	The process ID (PID) of the Trap Multiplexer backup process.	(*CPU-number,PIN-number*)	Trap Multiplexer.
ztmxProcCreatTime 1.3.6.1.4.1.169.3.185.2.7 read-only DisplayString	The system time at which the Trap Multiplexer process started.	started month DD, YYYY HH:MM:SS	Guardian procedure JULIANTIMESTAMP.
ztmxProcCpuTime 1.3.6.1.4.1.169.3.185.2.8 read-only DisplayString	The amount of CPU time used by the Trap Multiplexer since starting.	HH:MM:SS	Trap Multiplexer.

Table 12-3. ztmxProcess Group Objects Supported by Trap Multiplexer's MIB (page 2 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxProcPri 1.3.6.1.4.1.169.3.185.2.9 read-write INTEGER	The priority of the Trap Multiplexer primary and backup processes.	Integer without leading 0s.	Initially, the value of the -n startup parameter (145 or a user-specified value). After startup, the value can be set from an SNMP manager.

ztmxProcHomeTerm
1.3.6.1.4.1.169.3.185.2.10
read-only
DisplayString
	The name of the terminal from which the Trap Multiplexer was started.	`\`*`node.$device-name.`*
		`#subdevice-name`
			Guardian procedure PROCESS_GETINFO_.

ztmxProcHeapInitial 1.3.6.1.4.1.169.3.185.2.11 read-only Gauge	The number of bytes of memory buffer space allocated at Trap Multiplexer startup.	Integer without leading 0s.	Internal library routine.
ztmxProcHeapCurrent 1.3.6.1.4.1.169.3.185.2.12 read-only Gauge	The number of bytes of memory buffer space currently in use by the Trap Multiplexer.	Integer without leading 0s.	Internal library routine.

ztmxProcEmsCollector
1.3.6.1.4.1.169.3.185.2.13
read-only
DisplayString
	The name of the EMS collector process to which event messages are being sent.	`\`*`node.$volume.`*
		`subvolume.process`
			The name specified in the -c startup parameter. If not specified, the name $0.

ztmxProcAgent
1.3.6.1.4.1.169.3.185.2.14
read-only
DisplayString
	The name of the Nonstop agent process the Trap Multiplexer is communicating with.	`\`*`node.$volume.`*
		`subvolume.process`
			The name specified in the -a startup parameter. If not specified, the name $ZSNMP.

Table 12-3. ztmxProcess Group Objects Supported by Trap Multiplexer's MIB (page 3 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxProcAgentSessionStatus 1.3.6.1.4.1.169.3.185.2.15 read-only INTEGER { open(1), close(2) }	The status of the session between the Trap Multiplexer and the SNMP agent. A value of close indicates that the Trap Multiplexer is not communicating with the SNMP agent.	One of these values is displayed: open close	Trap Multiplexer.
ztmxProcUdpPort 1.3.6.1.4.1.169.3.185.2.16 read-only INTEGER	The UDP port to which the Trap Multiplexer is bound for trap receipt.	Integer without leading 0s.	The port number specified in the -p startup parameter. If not specified, 162.
ztmxProcTcpIpProcess 1.3.6.1.4.1.169.3.185.2.17 read-only DisplayString	The names of the TCP/IP processes from which the Trap Multiplexer is receiving traps.	Each TCP/IP process name has the following form: `tcpip-proc-name`	The TCP/IP process names specified in the -t startup parameter. If not specified, the name is $ZTC0.
ztmxSwitchToBackupNow 1.3.6.1.4.1.169.3.185.2.18 read-write INTEGER { autoProcessPairControl(0), forceBackupTakeover(1) }	A switch that causes the Trap Multiplexer backup process to take over and a new backup process to be created when set to 1.	One of these values is displayed: autoProcessPairControl forceBackupTakeover When setting these values, use 0 or 1.	At startup, the subagent sets this value to 0, but it can be changed from an SNMP manager if the Trap Multiplexer was started using the -b startup parameter.
ztmxProcProgramFileName 1.3.6.1.4.1.169.3.185.2.19 read-only DisplayString	The program file name of the Trap Multiplexer.	`\node.$volume.` `subvolume.file-name`	Trap Multiplexer.

Table 12-3. ztmxProcess Group Objects Supported by Trap Multiplexer's MIB (page 4 of 4)

Object and Attributes	Definition	Format of Value	Derivation of Value
ztmxProcEmsWriteTotal 1.3.6.1.4.1.169.3.185.2.20 read-only Counter	The total number of attempts to write an event message converted from a trap to the EMS collector.	Integer without leading 0s.	Trap Multiplexer.
ztmxProcEmsWriteErrors 1.3.6.1.4.1.169.3.185.2.21 read-only Counter	The total number of write operations to the EMS collector that could not be completed.	Integer without leading 0s.	Trap Multiplexer.
ztmxProcKeepAliveTimer 1.3.6.1.4.1.169.3.185.2.22 read-write INTEGER	The number of seconds between attempts the subagent makes to connect a TCP/IP process.	Integer without leading 0s.	The value specified in the -k startup parameter. If not specified, the default is 15 minutes. After startup, the value can be set from an SNMP manager.

ztmxProcess Group Maintenance

The ztmxProcess group values are updated:

- At Trap Multiplexer startup time. The values of MIB objects associated with startup parameters and startup process attributes are initialized.
- When traps and requests are received. The counters in the ztmxPDUStatisticsGroup, the counters that tally the write operations to the EMS collector, and the values of ztmxProcCpuTime and ztmxProcHeapCurrent are updated as traps and requests are received.
- When a request is received from an SNMP manager to change the Trap Multiplexer's process priority. The value of ztmxProcPri does not persist after the Trap Multiplexer stops. The next time the Trap Multiplexer starts, this object is assigned the priority specified at invocation.

- When the value of ztmxSwitchToBackupNow is set to a 1. If a backup process exists, the Trap Multiplexer's backup process takes over, refreshing all values in the Trap Multiplexer's MIB before handling SNMP manager requests or processing incoming traps. After takeover is complete, the Trap Multiplexer sets the value of ztmxSwitchToBackupNow to 0. Use the ztmxProcKeepAliveTimer startup parameter if you want to keep monitoring a TCP/IP process that has gone down until it comes back up.

ztmxProcTcpIpProcess maintains the list of TCP/IP processes being monitored by the Trap Multiplexer in the following format:

```
$tcpname1:ipaddr1, ipaddr2 $tcpname2
```

For more information about using the -t parameter and PARAMs, see Using the -t and PARAM Startup Parameters on page 12-10.

EMS Messages

This subsection describes the EMS events generated by the Trap Multiplexer (subsystem abbreviation ZTMX).

Event Summary

The Trap Multiplexer generates events that fall into two categories:

- **Trap events**. These events describe each trap received.
- **Process events**. These events highlight Trap Multiplexer process changes.

Trap Events

The trap events are listed in Table 12-4. Each SNMP trap type has a corresponding event.

Table 12-4. Trap Multiplexer Trap Event Messages

Number	Event	Corresponding Trap Type
0	ZTMX-EVT-TRAP-COLDSTART	coldStart
1	ZTMX-EVT-TRAP-WARMSTART	warmStart
2	ZTMX-EVT-TRAP-LINKDOWN	linkDown
3	ZTMX-EVT-TRAP-LINKUP	linkUp
4	ZTMX-EVT-TRAP-AUTH-FAIL	authenticationFailure
5	ZTMX-EVT-TRAP-EGP-LOSS	egpNeighborLoss
6	ZTMX-EVT-TRAP-ENT-SPFC	enterpriseSpecific

Each header variable of a trap PDU is mapped to an EMS token, as Figure 12-2 illustrates.

Variable bindings are mapped to a collection of EMS tokens grouped into a series of lists. The first token in the list describes the number of variable bindings (varbinds) in the trap PDU. If this value is greater than 0, a nested list of tokens is used to describe each varbind; for example:

```
ZSPI-TKN-DATALIST                          <--start of outer list
   ZTMX-TKN-TRAP-VARBIND-COUNT             <--number of varbinds
      ZSPI-TKN-DATALIST                    <--start of first varbind list
         ZTMX-TKN-TRAP-VARBIND-OID         <--OID of varbind
         ZTMX-TKN-TRAP-VARBIND-DATATYPE    <--data type of varbind
         value-of-variable-binding         <--value of varbind
      ZSPI-TKN-ENDLIST                     <--end of first varbind list
      .
      .
      .
      ZSPI-TKN-DATALIST                    <--start of last varbind list
         ZTMX-TKN-TRAP-VARBIND-OID         <--OID of varbind
         ZTMX-TKN-TRAP-VARBIND-DATATYPE    <--data type of varbind
         value-of-variable-binding         <--value of varbind
      ZSPI-TKN-ENDLIST                     <--end of last varbind list
ZSPI-TKN-ENDLIST                           <--end of outer list
```

Figure 12-2. Mapping of Trap Header Variables to EMS Event Tokens

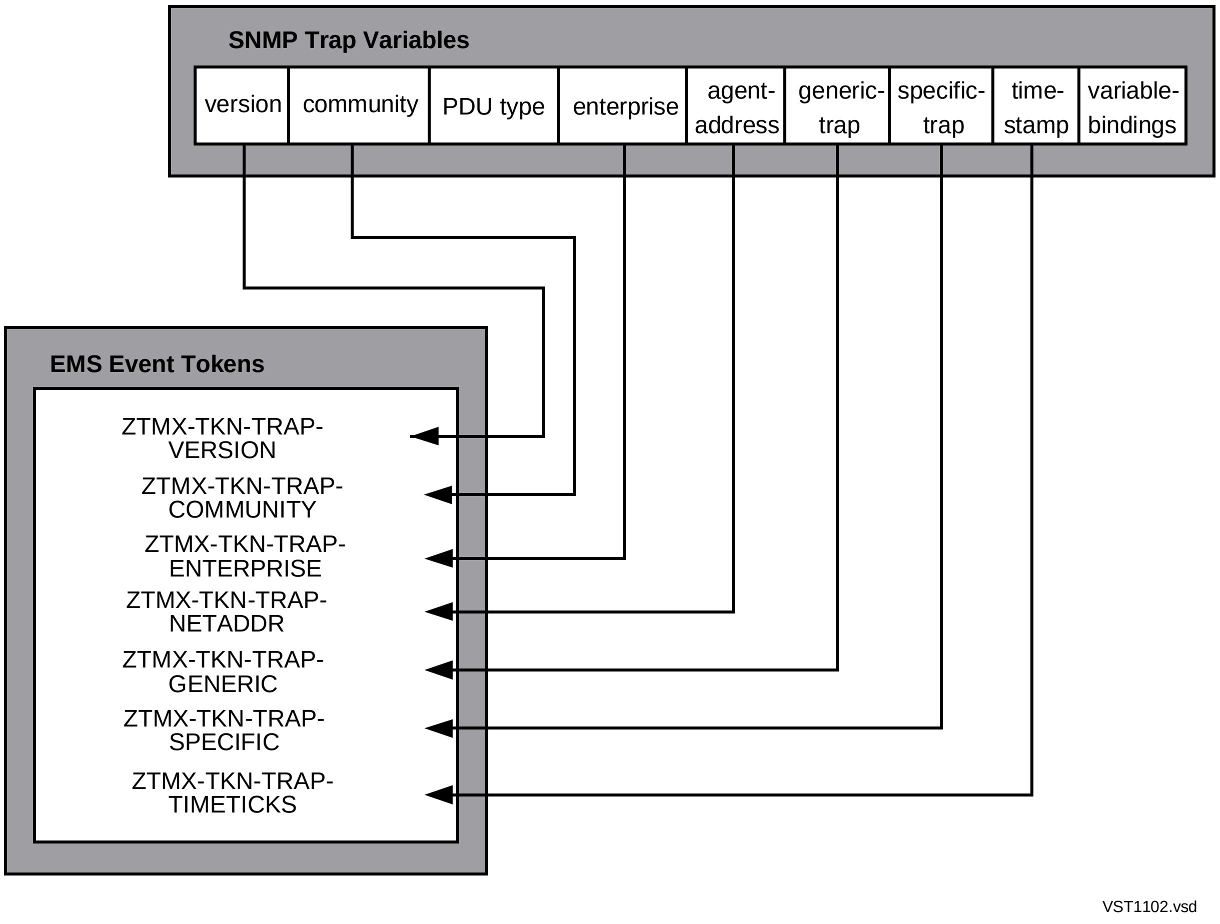


Table 12-5 summarizes the layout of tokens in an EMS event generated from a trap.

Table 12-5. Trap Event Token Summary (page 1 of 2)

EMS Token	**Contents**
ZEMS-TKN-EVENTNUMBER	The event number: 0 ZTMX-EVT-TRAP-COLDSTART 1 ZTMX-EVT-TRAP-WARMSTART 2 ZTMX-EVT-TRAP-LINKDOWN 3 ZTMX-EVT-TRAP-LINKUP 4 ZTMX-EVT-TRAP-AUTH-FAIL 5 ZTMX-EVT-TRAP-EGP-LOSS 6 ZTMX-EVT-TRAP-ENT-SPFC
ZEMS-TKN-SUBJ-TMX	The Trap Multiplexer process name
ZTMX-TKN-TRAP-PDU	A copy of the trap received, in its original BER-encoded format
ZTMX-TKN-TRAP-VERSION	The number in the version field of the trap PDU
ZTMX-TKN-TRAP-COMMUNITY	The community name associated with the trap originator, from the community field of the trap PDU
ZTMX-TKN-TRAP-ENTERPRISE	The enterprise identifier of the trap originator, from the enterprise field of the trap PDU
ZTMX-TKN-TRAP-NETADDR	The IP address of the trap originator, from the agent-addr field of the trap PDU
ZTMX-TKN-TRAP-GENERIC	The predefined trap type in the generic-trap field of the trap PDU: 0 coldStart 1 warmStart 2 linkDown 3 linkUp 4 authenticationFailure 5 egpNeighborLoss 6 enterpriseSpecific
ZTMX-TKN-TRAP-SPECIFIC	The value in the specific-trap field of the trap PDU
ZTMX-TKN-TRAP-TIMETICKS	The value in the timestamp field of the trap PDU, the value of sysDescr.sysUpTime.0 at trap generation
ZSPI-TKN-DATALIST	A marker indicating the start of varbind information
ZTMX-TKN-TRAP-VARBIND-COUNT	The number of varbinds in the trap PDU

Table 12-5. Trap Event Token Summary (continued) (page 2 of 2)

EMS Token	Contents
One of the following lists for each varbind in the trap PDU:	
ZSPI-TKN-DATALIST	A marker indicating the start of a token list describing one varbind
ZTMX-TKN-TRAP-VARBIND-OID	The object identifier of the varbind, from the trap PDU
ZTMX-TKN-TRAP-VARBIND-DATATYPE	A token identifying the SNMP data type of the varbind value: ZTMX-TNM-SMI-NULL ZTMX-TNM-SMI-INTEGER ZTMX-TNM-SMI-OCTETSTRING ZTMX-TNM-SMI-OID ZTMX-TNM-SMI-IPADDR ZTMX-TNM-SMI-COUNTER ZTMX-TNM-SMI-GAUGE ZTMX-TNM-SMI-TICKS ZTMX-TNM-SMI-OPAQUE
One of the following tokens: ZTMX-TKN-TRAP-NULL ZTMX-TKN-TRAP-INTEGER ZTMX-TKN-TRAP-OCTETSTRING ZTMX-TKN-TRAP-OID ZTMX-TKN-TRAP-IPADDR ZTMX-TKN-TRAP-COUNTER ZTMX-TKN-TRAP-GAUGE ZTMX-TKN-TRAP-TIMETICKS ZTMX-TKN-TRAP-OPAQUE	The value of the varbind. The exact token that appears depends on the value of ZTMX-TKN-TRAP-VARBIND-DATATYPE. For example, if the varbind has a data type value of ZTMX-TNM-SMI-INTEGER, its value is associated with ZTMX-TKN-TRAP-INTEGER.
ZSPI-TKN-ENDLIST	A marker indicating the end of the token list describing one varbind
ZSPI-TKN-ENDLIST	A marker indicating the end of varbind information

Each trap event is described individually in Trap Event Message Descriptions on page 12-28.

Process Events

The process events are listed in Table 12-6. All the process events are standard events, as the table indicates. See the *EMS Manual* for a complete description of standard events.

Table 12-6. Trap Multiplexer Process Event Messages

Number	Event	Standard Event
101	ZTMX-EVT-TMX-TERMINATED	Object Unavailable
102	ZTMX-EVT-TMX-STARTED	Object Available
103	ZTMX-EVT-TMX-IO-ERR	Transient Fault
104	ZTMX-EVT-TMX-NO-MEM-SPACE	Transient Fault
105	ZTMX-EVT-TMX-PROG-ERR	Transient Fault

Each process event is described individually in Process Event Message Descriptions on page 12-39.

Data Definitions

Many of the tokens, structures, and values in Trap Multiplexer events are defined in the ZSPI and ZEMS data definition files ZSPIDDL and ZEMSDDL and their associated language-specific files. Data elements defined by the Trap Multiplexer are in the file ZTMXDDL and associated language-specific files:

`ZTMXDDL`	Data Definition Language (DDL) definitions from which the language-specific definitions are derived
`ZTMXC`	Definitions for C programs
`ZTMXCOB`	Definitions for COBOL programs
`ZTMXPAS`	Definitions for Pascal programs
`ZHTMXTACL`	Definitions for TACL programs
`ZTMXTAL`	Definitions for TAL programs

The complete set of SPI definition files is usually located in the ZSPIDEF subvolume of the NonStop Kernel installation volume. See the *SPI Programming Manual* for information on the ZSPI data definitions and data definition files in general. See the *EMS Manual* for information on ZEMS data definitions

Subsystem ID

```
CONSTANT ZTMX-VAL-VERSION           VALUE  VERSION "D20".
CONSTANT ZSPI-SSN-ZTMX              VALUE  185.

DEFINITION ZTMX-VAL-SSID            TACL   SSID.
  02 z-filler  TYPE character 8     VALUE IS ZSPI-VAL-TANDEM.
  02 z-owner   TYPE ZSPI-DDL-CHAR8  REDEFINES z-filler.
  02 z-number  TYPE ZSPI-DDL-INT    VALUE IS ZSPI-SSN-ZTMX.
  02 z-version TYPE ZSPI-DDL-UINT   VALUE IS ZTMX-VAL-
VERSION.
END
```

Tokens in ZTMX Event Messages

Trap Multiplexer (ZTMX) Tokens

Table 12-7 lists all the tokens defined by the Trap Multiplexer. These tokens are defined in the ZTMXDDL definition file and are described where they appear in the event message descriptions at the end of this section.

Table 12-7. ZTMX Tokens in ZTMX Event Messages (page 1 of 2)

Token	Contents
ZTMX-TKN-ERR-TEXT	Additional error text
ZTMX-TKN-EVENTNUMBER	The event number of a trap event.
ZTMX-TKN-FILE-NAME	Name of file related to event
ZTMX-TKN-FS-ERR	File system error code
ZTMX-TKN-SUBJ-TMX	Trap Multiplexer subject token
ZTMX-TKN-TRAP-COMMUNITY	Community value in a trap PDU
ZTMX-TKN-TRAP-ENTERPRISE	Enterprise value in a trap PDU
ZTMX-TKN-TRAP-GENERIC	Generic trap type in generic-trap field of a trap PDU
ZTMX-TKN-TRAP-NETADDR	IP address in a trap PDU
ZTMX-TKN-TRAP-PDU	Copy of trap received, in BER-encoded format
ZTMX-TKN-TRAP-SMI-COUNTER	Value of a varbind with a data type of Counter
ZTMX-TKN-TRAP-SMI-GAUGE	Value of a varbind with a data type of Gauge
ZTMX-TKN-TRAP-SMI-INTEGER	Value of a varbind with a data type of INTEGER
ZTMX-TKN-TRAP-SMI-IPADDR	Value of a varbind with a data type of IpAddress
ZTMX-TKN-TRAP-SMI-NULL	Value of a varbind with a data type of NULL

Table 12-7. ZTMX Tokens in ZTMX Event Messages (page 2 of 2)

Token	Contents
ZTMX-TKN-TRAP-SMI-OCTETSTRING	Value of a varbind with a data type of OctetString
ZTMX-TKN-TRAP-SMI-OID	Value of a varbind with a data type of OBJECT IDENTIFIER
ZTMX-TKN-TRAP-SMI-OPAQUE	Value of a varbind with a data type of Opaque
ZTMX-TKN-TRAP-SMI-TIMETICKS	Value of a varbind with a data type of TimeTicks
ZTMX-TKN-TRAP-SPECIFIC	Specific trap type in specific-trap field of a trap PDU
ZTMX-TKN-TRAP-TIMETICKS	Timestamp value in a trap PDU
ZTMX-TKN-TRAP-VARBIND-COUNT	Number of varbinds in a trap
ZTMX-TKN-TRAP-VARBIND-DATATYPE	SNMP data type of a varbind in a trap
ZTMX-TKN-TRAP-VARBIND-OID	Object identifier of a varbind in a trap
ZTMX-TKN-TRAP-VERSION	SNMP version number in a trap PDU

Standard SPI Tokens

Table 12-8 lists the SPI-defined tokens that are used by the Trap Multiplexer. The standard SPI tokens are defined in the ZSPIDDL definition file. See the *SPI Programming Manual* for more information on these tokens and related data definitions.

Table 12-8. ZSPI Tokens in ZTMX Event Messages

Token	Contents
ZSPI-TKN-MAX-FIELD-VERSION	Latest version of any extensible structured token in the buffer
ZSPI-TKN-SSID	ZTMX subsystem ID
ZSPI-TKN-USEDLEN	Bytes used in EMS buffer

Standard EMS Tokens

Table 12-9 lists all the EMS-defined tokens used by the subagent. These tokens are defined in the ZEMSDDL definition file. See the *EMS Manual* for more information on these tokens and related data definitions.

Table 12-9. ZEMS Tokens in ZTMX Event Messages

Token	Contents
ZEMS-TKN-BATCHJOB-ID	Batch job ID of process
ZEMS-TKN-CONTENT-STANDARD	Type of standard event
ZEMS-TKN-CONTENT-USER	Type of user-defined event
ZEMS-TKN-CPU	CPU number of event sender
ZEMS-TKN-EMPHASIS	Critical/Noncritical event flag
ZEMS-TKN-EVENTNUMBER	Event number
ZEMS-TKN-GENTIME	Event generation time
ZEMS-TKN-LOGTIME	Event log time
ZEMS-TKN-NAME-MANAGER	Manager process name
ZEMS-TKN-NODENUM	System number of event sender
ZEMS-TKN-PIN	PIN of event sender
ZEMS-TKN-SUBJECT-MARK	Event subject marker
ZEMS-TKN-SUPPRESS-DISPLAY	Display/do not display event flag
ZEMS-TKN-USERID	User ID of event sender
ZEMS-TKN-CHANGE-REASON	Object state change reason; private enumerations: ZTMX-VAL-CR-PROCESS-NOMEM ZTMX-VAL-CR-PROCESS-SIGABRT ZTMX-VAL-CR-PROCESS-SIGFPE ZTMX-VAL-CR-PROCESS-SIGILL ZTMX-VAL-CR-PROCESS-SIGSEGV ZTMX-VAL-CR-PROCESS-SIGSTK ZTMX-VAL-CR-PROCESS-SIGTERM ZTMX-VAL-CR-PROCESS-SIGTIMEOUT ZTMX-VAL-CR-PROCESS-STARTED ZTMX-VAL-CR-PROCESS-STOPPED
ZEMS-TKN-STATE-CURRENT	Current object state; private enumerations: ZTMX-VAL-TMX-DEFINED ZTMX-VAL-ZTMX-STARTED
ZEMS-TKN-STATE-PREVIOUS	Previous object state; private enumerations: ZTMX-VAL-HRM-SA-DEFINED ZTMX-VAL-HRM-SA-STARTED
ZEMS-TKN-TXFAULT-TYPE	Type of transient fault; private enumerations: ZTMX-VAL-TF-IO (I/O error) ZTMX-VAL-TF-MEM (no memory space) ZTMX-VAL-TF-PROG (internal error)

Trap Event Message Descriptions

- Tokens listed as “unconditional” always appear in the event message. Tokens listed as “conditional” are included only under described conditions.

Tokens not defined by ZTMX are listed only if they contain information that appears in the printed message text or if they contain ZTMX-defined values.

- The event message text illustrates what is generated when the contents of the event message are displayed according to the message template defined in the file ZTMXTMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the message template source file STMXTMPL.

  Nonprintable characters in the following values are ignored when formatting the message for display: the community string, Opaque values, and OctetString values.

Information specific to each trap event is described immediately following this generic description, in order by event number.

Unconditional Tokens	**Value**
ZTMX-TKN-SUBJ-TMX	<2> *Trap Multiplexer process name*
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-*trap-type*
ZTMX-TKN-TRAP-PDU	*Copy of BER-encoded trap*
ZTMX-TKN-TRAP-VERSION	<3> *Version number in trap*
ZTMX-TKN-TRAP-COMMUNITY	<4> *Community string in trap*
ZTMX-TKN-TRAP-ENTERPRISE	<5> *Enterprise OID in trap*
ZTMX-TKN-TRAP-NETADDR	<6> *IP address of trap origin*
ZTMX-TKN-TRAP-GENERIC	<7> *Generic trap type in trap*
ZTMX-TKN-TRAP-SPECIFIC	<8> *Specific trap type in trap*
ZTMX-TKN-TRAP-TIMETICKS	<9> *Time-stamp in trap*
ZTMX-TKN-TRAP-VARBIND-COUNT	
Conditional Tokens	
ZTMX-TKN-TRAP-VARBIND-OID	<10> *Varbind OID*
ZTMX-TKN-TRAP-VARBIND-DATATYPE	<11> *Varbind SNMP data type*
ZTMX-TKN-TRAP-SMI-NULL	<12> *NULL varbind value*
ZTMX-TKN-TRAP-SMI-INTEGER	<12> *INTEGER varbind value*
ZTMX-TKN-TRAP-SMI-OCTETSTRING	<12> *OCTET STRING varbind value*
ZTMX-TKN-TRAP-SMI-OID	<12> *OBJECT IDENTIFIER varbind value*
ZTMX-TKN-TRAP-SMI-IPADDR	<12> *IpAddress varbind value*
ZTMX-TKN-TRAP-SMI-COUNTER	<12> *Counter varbind value*
ZTMX-TKN-TRAP-SMI-GAUGE	<12> *Gauge varbind value*
ZTMX-TKN-TRAP-SMI-TIMETICKS	<12> *TimeTicks varbind value*
ZTMX-TKN-TRAP-SMI-OPAQUE	<12> *Opaque varbind value*

Message Text Without Varbinds

```
<1> trap type - SNMP-Trap-Multiplexer - <2>, Version: <3>,
Community: <4>, Enterprise: <5>, Network addr: <6>, Generic
trap: <7>, Specific trap: <8>, Time ticks: <9> (no varbinds)
```

Message Text With Varbinds

```
<1> trap type - SNMP-Trap-Multiplexer - <2>, Version: <3>,
Community: <4>, Enterprise: <5>, Network addr: <6>, Generic
trap: <7>, Specific trap: <8>, Time ticks: <9>, (varbind 1
OID=<10>, type=<11>, value=<12>), ...
(varbind n OID=<10>, type=<11>, value=<12>)
```

`ZTMX-TKN-SUBJ-TMX`

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

`ZTMX-EVT-TRAP-`*`trap-type`*

is the event number. This number and the *`trap-type`* text following it are unique to each trap event.

`ZTMX-TKN-TRAP-PDU`

is a copy of the trap PDU received, in its original BER-encoded form.

`ZTMX-TKN-TRAP-VERSION`

is the number in the version field of the trap PDU. In the message text, it follows the string “Version:”

`ZTMX-TKN-TRAP-COMMUNITY`

iIs the community name associated with the trap originator, from the community field of the trap PDU. In the message text, it follows the string “Community:”

`ZTMX-TKN-TRAP-ENTERPRISE`

is the enterprise identifier of the trap originator, from the enterprise field of the trap PDU. In the message text, it follows the string “Enterprise:”

`ZTMX-TKN-TRAP-NETADDR`

is the IP address of the trap originator, from the agent-addr field of the trap PDU. In the message text, it follows the string “Network addr:”

`ZTMX-TKN-TRAP-GENERIC`

is the predefined trap type in the generic-trap field of the trap PDU. In the message text, it follows the string “Generic trap:”

`ZTMX-TKN-TRAP-SPECIFIC`

is the value in the specific-trap field of the trap PDU. In the message text, it follows the string “Specific trap:”

`ZTMX-TKN-TRAP-TIMETICKS`

is the value in the time-stamp field of the trap PDU, the value of sysDescr.sysUpTime.0 at trap generation. In the message text, it follows the string “Time ticks:”

`ZTMX-TKN-TRAP-VARBIND-COUNT`

is the number of varbinds in the trap.

`ZTMX-TKN-TRAP-VARBIND-OID`

is the object identifier of a varbind in the trap. In the message text, it follows the string “varbind *n* OID=”, where *n* is a number identifying the varbind.

`ZTMX-TKN-TRAP-VARBIND-DATATYPE`

is the SNMP data type of a varbind in the trap. In the message text, it follows the string “type=”.

`ZTMX-TKN-TRAP-SMI-NULL`

is the value of a varbind with a data type of NULL. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-INTEGER`

is the value of a varbind with a data type of INTEGER. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-OCTETSTRING`

is the value of a varbind with a data type of OCTET STRING. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-OID`

is the value of a varbind with a data type of OBJECT IDENTIFIER. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-IPADDR`

is the value of a varbind with a data type of IpAddress. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-COUNTER`

is the value of a varbind with a data type of Counter. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-GAUGE`

is the value of a varbind with a data type of Gauge. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-TIMETICKS`

is the value of a varbind with a data type of TimeTicks. In the message text, it follows the string “value=”.

`ZTMX-TKN-TRAP-SMI-OPAQUE`

is the value of a varbind with a data type of Opaque. In the message text, it follows the string “value=”.

000: ZTMX-EVT-TRAP-COLDSTART

The Trap Multiplexer received a coldStart trap.

Unconditional Tokens	Value
`ZEMS-TKN-EVENTNUMBER`	`<1> ZTMX-EVT-TRAP-COLDSTART`

```
For other unconditional tokens, refer to the generic token
list preceding the individual trap event descriptions.
```

Conditional Tokens

```
For conditional tokens, refer to the generic token list
preceding the individual trap event descriptions.
```

Message Text

```
Refer to the generic message text preceding the individual
trap event descriptions and to the sample message that
follows.
```

`ZTMX-EVT-TRAP-COLDSTART`

is the event number (0). In the message text, it precedes a string identifying the trap type:

```
000000 Coldstart Trap
```

Cause. The Trap Multiplexer received a coldStart trap. A coldStart trap is a standard SNMP trap signifying that the sending entity has reinitialized—most likely because of a restart due to an error condition—and its configuration might have changed.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with determining how to interpret and handle the coldStart trap.

Sample Message

```
96-07-26 10:18:36 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000000 Coldstart
                                  Trap - SNMP-Trap-Multiplexer $ZTMX,
                                  Version: 0, Community: public, Enterprise:
                                  1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                  130.252.12.3, Generic trap: 0, Specific
                                  trap: 0, Time ticks: 45366736 (no
                                  varbinds)
```

001: ZTMX-EVT-TRAP-WARMSTART

The Trap Multiplexer received a warmStart trap.

Unconditional Tokens	**Value**
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-WARMSTART

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

ZTMX-EVT-TRAP-WARMSTART

is the event number (1). In the message text, it precedes a string identifying the trap type:

```
000001 Warmstart Trap
```

Cause. The Trap Multiplexer received a warmStart trap. A warmStart trap is a standard SNMP trap signifying that the sending entity has reinitialized, most likely because of a routine restart, but its configuration will not have changed.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with determining how to interpret and handle the warmStart trap.

Sample Message

```
96-07-26 10:20:12 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000001 Warmstart
                                  Trap - SNMP-Trap-Multiplexer $ZTMX,
                                  Version: 0, Community: public, Enterprise:
                                  1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                  130.252.12.3, Generic trap: 1, Specific
                                  trap: 0, Time ticks: 45366736 (no
                                  varbinds)
```

002: ZTMX-EVT-TRAP-LINKDOWN

The Trap Multiplexer received a linkDown trap.

Unconditional Tokens	Value
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-LINKDOWN

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

`ZTMX-EVT-TRAP-LINKDOWN`

is the event number (2). In the message text, it precedes a string identifying the trap type:

```
000002 Link Down Trap
```

Cause. The Trap Multiplexer received a linkDown trap. A linkDown trap is a standard SNMP trap signifying that the sending entity has experienced a failure in one of its communication links.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with the tools that might be used to identify the source and nature of the problem that caused the linkDown trap.

Sample Message

```
96-07-26 10:22:45 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000002 Link Down
                                  Trap - SNMP-Trap-Multiplexer $ZTMX,
                                  Version: 0, Community: public, Enterprise:
                                  1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                  130.252.12.3, Generic trap: 2, Specific
                                  trap: 0, Time ticks: 45366736 (no
                                  varbinds)
```

003: ZTMX-EVT-TRAP-LINKUP

The Trap Multiplexer received a linkUp trap.

Unconditional Tokens	Value
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-LINKUP

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

ZTMX-EVT-TRAP-LINKUP

is the event number (3). In the message text, it precedes a string identifying the trap type:

```
000003 Link Up Trap
```

Cause. The Trap Multiplexer received a linkUp trap. A linkUp trap is a standard SNMP trap signifying that one of the sending entity's communication links has become activated.

Effect. The effect is specific to the product that generated the trap.

Recovery. This message is intended for monitoring programs. If necessary, contact your SNMP site expert for assistance with determining how to interpret and handle the trap.

Sample Message

```
96-07-26 10:26:36 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000003 Link Up Trap
                                  - SNMP-Trap-Multiplexer $ZTMX, Version: 0,
                                  Community: public, Enterprise:
                                  1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                  130.252.12.3, Generic trap: 3, Specific
                                  trap: 0, Time ticks: 45366736 (no
                                  varbinds)
```

004: ZTMX-EVT-TRAP-AUTH-FAIL

The Trap Multiplexer received an authenticationFailure trap.

Unconditional Tokens	Value
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-AUTH-FAIL

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

`ZTMX-EVT-TRAP-AUTH-FAIL`

is the event number (4). In the message text, it precedes a string identifying the trap type:

```
000004 Authentication Failure Trap
```

Cause. The Trap Multiplexer received an authenticationFailure trap. An authenticationFailure trap is a standard SNMP trap signifying that the sending entity has received a request that failed authentication.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with the tools that might be used to identify the source and nature of the problem that caused the authenticationFailure trap.

Sample Message

```
96-07-26 10:30:09 \COMM.$ZTMX       TANDEM.TRAPMUX.D31    000004
                                    Authentication Failure Trap -
                                    SNMP-Trap-Multiplexer $ZTMX, Version: 0,
                                    Community: public, Enterprise:
                                    1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                    130.252.12.3, Generic trap: 4, Specific
                                    trap: 0, Time ticks: 45366736 (no
                                    varbinds)
```

005: ZTMX-EVT-TRAP-EGP-LOSS

The Trap Multiplexer received an egpNeighborLoss trap.

Unconditional Tokens	**Value**
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-EGP-LOSS

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

ZTMX-EVT-TRAP-EGP-LOSS

is the event number (5). In the message text, it precedes a string identifying the trap type:

```
000005 Exterior Gateway Protocol Neighbor Loss Trap
```

Cause. The Trap Multiplexer received an egpNeighborLoss trap. An egpNeighborLoss trap is a standard SNMP trap signifying that the sending entity no longer has a peer relationship with an EGP (Exterior Gateway Protocol) neighbor.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with the tools that might be used to identify the source and nature of the problem that caused the egpNeighborLoss trap.

Sample Message

```
96-07-26 10:31:44 \COMM.$ZTMX       TANDEM.TRAPMUX.D31    000005 Exterior
                                    Gateway Protocol Neighbor Loss Trap -
                                    SNMP-Trap-Multiplexer $ZTMX, Version: 0,
                                    Community: public, Enterprise:
                                    1.3.6.1.4.1.169.2.2.66.0, Network addr:
                                    130.252.12.3, Generic trap: 5, Specific
                                    trap: 0, Time ticks: 45366736 (no
                                    varbinds)
```

006: ZTMX-EVT-TRAP-ENT-SPFC

The Trap Multiplexer received an enterpriseSpecific trap.

Unconditional Tokens	**Value**
ZEMS-TKN-EVENTNUMBER	<1> ZTMX-EVT-TRAP-ENT-SPFC

For other unconditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Conditional Tokens

For conditional tokens, refer to the generic token list preceding the individual trap event descriptions.

Message Text

Refer to the generic message text preceding the individual trap event descriptions and to the sample message that follows.

ZTMX-EVT-TRAP-ENT-SPFC

is the event number (6). In the message text, it precedes a string identifying the trap type:

```
000006 Enterprise Specific Trap
```

Cause. The Trap Multiplexer received an enterpriseSpecific trap. This trap indicates that the sending entity has experienced some enterprise-defined event.

Effect. The effect is specific to the product that generated the trap.

Recovery. The recovery actions required are specific to the product that generated the trap. If necessary, contact your SNMP site expert for assistance with the tools that might be used to identify the source and nature of the problem that caused the enterpriseSpecific trap.

Sample Message

The following messages were generated from two traps received from the Host Resources Subagent, described in Section 11, Host Resources Subagent.

```
96-07-26 11:11:02 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000006 Enterprise
                                  Specific Trap - SNMP-Trap-Multiplexer
                                  $ZTMX, Version: 0, Community: public,
                                  Enterprise: 1.3.6.1.4.1.169.3.180.7,
                                  Network addr: 130.252.109.237, Generic
                                  trap: 6, Specific trap: 1, Time ticks:
                                  8961029, (varbind 1
                                  OID=1.3.6.1.4.1.169.3.180.7.1,
                                  type=INTEGER, value=10001), (varbind 2
                                  OID=1.3.6.1.4.1.169.3.180.7.2,
                                  type=OCTET-STRING, value=$TAPE), (varbind
                                  3 OID=1.3.6.1.4.1.169.3.180.7.3,
                                  type=INTEGER, value=2)


96-07-26 11:15:14 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000006 Enterprise
                                  Specific Trap - SNMP-Trap-Multiplexer
                                  $ZTMX, Version: 0, Community: public,
                                  Enterprise: 1.3.6.1.4.1.169.3.180.7,
                                  Network addr: 130.252.12.3, Generic trap:
                                  6, Specific trap: 2, Time ticks: 741158,
                                  (varbind 1 OID=1.3.6.1.4.1.169.3.180.7.4,
                                  type=INTEGER, value=1071), (varbind 2
                                  OID=1.3.6.1.4.1.169.3.180.7.5,
                                  type=OCTET-STRING, value=$OPER-P(4500-2)),
                                  (varbind 3
                                  OID=1.3.6.1.4.1.169.3.180.2.2.3,
                                  type=INTEGER, value=81)
```

Process Event Message Descriptions

- Token lists. Tokens listed as “unconditional” always appear in the event message. Tokens listed as “conditional” are included only under described conditions.

  Tokens not defined by ZTMX are listed only if they contain information that appears in the printed message text or if they contain ZTMX-defined values.

- Event message text. The event message text illustrates what is generated when the contents of the event message are displayed according to the message template defined in the file ZTMXTMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the message template source file STMXTMPL.

- Descriptions of listed tokens or values.
- The cause of the event, the conditions that prompted the Trap Multiplexer to generate the event message.
- The effects associated with or resulting from the cause.

- Recovery procedures you can follow to solve the problem reported by the event message.
- An example of the formatted message.

101: ZTMX-EVT-TMX-TERMINATED

The Trap Multiplexer process terminated normally.

Unconditional Tokens	Value
`ZTMX-TKN-SUBJ-TMX`	`<1>` *`Trap Multiplexer process name`*
`ZEMS-TKN-EVENTNUMBER`	`<2> ZTMX-EVT-TMX-TERMINATED`
`ZEMS-TKN-CHANGE-REASON`	`<3> ZTMX-VAL-CR-reason`
`ZEMS-TKN-STATE-PREVIOUS`	`<4> ZTMX-VAL-TMX-STARTED`
`ZEMS-TKN-STATE-CURRENT`	`<5) ZTMX-VAL-TMX-DEFINED`
`ZEMS-TKN-CONTENT-USER`	`<6> undefined`

Conditional Tokens

```
None
```

Message Text

```
Object unavailable - SNMP-Trap-Multiplexer - <1>, event
number: <2>, cause: <3>, previous state: <4>, current state:
(5), user content: <6>
```

`ZTMX-TKN-SUBJ-TMX`

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

`ZTMX-EVT-TMX-TERMINATED`

is the event number (101). The DDL AS clause of this value (“Process-Terminated”) appears in the message text following “event number:”

`ZTMX-VAL-CR-`*`reason`*

is one of the causes following below. The DDL AS clause associated with this value is inserted in the message text following “cause:”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS clause)**
ZTMX-VAL-CR-PROCESS-SIGILL	"Instruction-failure"
ZTMX-VAL-CR-PROCESS-SIGABRT	"Process-aborted"
ZTMX-VAL-CR-PROCESS-SIGFPE	"Arithmetic-overflow"
ZTMX-VAL-CR-PROCESS-SIGSEGV	"Illegal-address-reference"
ZTMX-VAL-CR-PROCESS-SIGSTK	"Stack-overflow"
ZTMX-VAL-CR-PROCESS-SIGTIMEOUT	"Loop-timer-timeout"
ZTMX-VAL-CR-PROCESS-SIGTERM	"Termination-request"
ZTMX-VAL-CR-PROCESS-STOPPED	"Process-stopped"
ZTMX-VAL-CR-PROCESS-NOMEM	"Memory-Space-Exhausted"

`ZTMX-VAL-TMX-STARTED`

is the previous state. The DDL AS clause of this value (“started”) appears in the message text following “previous state:”

`ZTMX-VAL-TMX-DEFINED`

is the current state. The DDL AS clause of this value (“not-running”) appears in the message text following “current state:”

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”

Cause. The Trap Multiplexer process terminated normally. (Abnormal termination is reported by the operating system.)

Effect. The backup process of the NonStop process pair should take over. If recovery is successful, an event announcing the takeover is generated.

Recovery. Examine the event logs for related events. Often event message 103 (ZTMX-EVT-TMX-IO-ERR) is generated prior to this event; refer to the message contents for a description of the associated problem. If you can determine the cause of the termination, correct it and, if necessary, restart the Trap Multiplexer process. If the termination resulted from incorrect specification of the agent process name, rerun the Trap Multiplexer using the -a startup parameter to specify the proper name.

Sample Message

```
96-07-26 11:43:07 \COMM.$ZTMX      TANDEM.TRAPMUX.D31    000101 Object
                                   unavailable  SNMP-Trap-Multiplexer -
                                   \COMM.$ZTMX, event number:
                                   Process-Terminated, cause:
                                   Process-stopped, previous state: started,
                                   current state: not-running, user content:
                                   undefined
```

102: ZTMX-EVT-TMX-STARTED

The Trap Multiplexer process has started.

Unconditional Tokens	Value
ZTMX-TKN-SUBJ-TMX	<1> *Trap Multiplexer process name*
EMS-TKN-EVENTNUMBER	<2> ZTMX-EVT-TMX-STARTED
ZEMS-TKN-CHANGE-REASON	<3> ZTMX-VAL-CR-PROCESS-STARTED
ZEMS-TKN-STATE-PREVIOUS	<4> ZTMX-VAL-TMX-DEFINED
ZEMS-TKN-STATE-CURRENT	<5> ZTMX-VAL-TMX-STARTED
ZEMS-TKN-CONTENT-USER	<6> undefined

Conditional Tokens

None

Message Text

```
Object available - SNMP-Trap-Multiplexer <1>, event number:
<2>, reason: <3>, previous state: <4>, current state: <5>,
user content <6>
```

`ZTMX-TKN-SUBJ-TMX`

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

`ZTMX-EVT-TMX-STARTED`

is the event number (1022). The DDL AS clause of this value (“Process-Started”) is inserted in the message text following “event number:”

`ZTMX-VAL-CR-PROCESS-STARTED`

is the change reason. The DDL AS clause of this value (“Process-started”) is inserted in the message text following “reason:”

`ZTMX-VAL-TMX-DEFINED`

is the previous state. The DDL AS clause of this value (“not-running”) appears in the message text following “previous state:”

`ZTMX-VAL-TMX-STARTED`

is the current state. The DDL AS clause of this value (“started”) appears in the message text following “current state:”

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”

Cause. The Trap Multiplexer process has started.

Effect. The process is ready for service.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-07-26 11:34:51 \COMM.$ZTMX      TANDEM.TRAPMUX.D31    000102 Object
                                   available  SNMP-Trap-Multiplexer -
                                   \COMM.$ZTMX, event number:
                                   Process-Started, reason: Process-started,
                                   previous state: not-running, current
                                   state: started, user content: undefined
```

103: ZTMX-EVT-TMX-IO-ERR

The Trap Multiplexer process encountered an I/O error.

Unconditional Tokens	**Value**
ZTMX-TKN-SUBJ-TMX	<1> *Trap Multiplexer process name*
ZEMS-TKN-EVENTNUMBER	<2> ZTMX-EVT-TMX-IO-ERR
ZEMS-TKN-TXFAULT-TYPE	<3> ZTMX-VAL-TF-IO
Conditional Tokens	**Value**
ZTMX-TKN-FS-ERR	<4> *error code and subcode*
ZTMX-TKN-FILE-NAME	<5> *file name*
ZTMX-TKN-ERR-TEXT	<6> *descriptive text*

Message Text

```
Transient Fault - SNMP-Trap-Multiplexer <1>, event number:
<2>, fault type: <3>, FS error: <4>, subcode: <4>, File Name:
<5>, Additional Info: <6>
```

`ZTMX-TKN-SUBJ-TMX`

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

`ZTMX-EVT-TMX-IO-ERR`

is the event number (103). The DDL AS clause of this value (“Process-I/O-error”) is inserted in the message text following “event number:”

`ZTMX-VAL-TF-IO`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“IO-error”) is inserted in the message text following “fault type:”

`ZTMX-TKN-FS-ERR`

contains the file-system error code and subcode, documented in the *Guardian Procedure Errors and Messages Manual*. The token is based on this structure:

```
DEFINITION ZTMX-DDL-FS-ERR.
  02 ERR-CODE       TYPE ZSPI-DDL-INT
                    HEADING 'error code'
  02 ERR-SUBCODE    TYPE ZSPI-DDL-INT
                    HEADING 'error subcode'
END.
```

`ZTMX-TKN-FILE-NAME`

contains the name of the file with which the subagent experienced an I/O error. This token is of type ZSPI-TYP-STRING.

`ZTMX-TKN-ERR-TEXT`

contains text describing the error. This token is of type ZSPI-TYP-STRING.

Cause. The Trap Multiplexer process encountered an I/O error. The cause is most likely the assignment of an incorrect agent process name at Trap Multiplexer startup or a stopped SNMP agent process.

Effect. The Trap Multiplexer cannot function.

Recovery. Investigate the reported error code and subcode and then correct the condition they describe. If the error code is 14 (file does not exist), the Trap Multiplexer could not open the SNMP agent process, and you should ensure that the SNMP agent is running or that you specified the correct agent process name when starting the Trap Multiplexer. If the error code is in the 4000 range, ensure that the Trap Multiplexer process is started by a member of the super user group and that no other process is controlling the port you want to monitor for traps.

If the Trap Multiplexer was running as a process pair when the I/O error occurred, the backup process takes over. This new primary process tries to establish communication with the SNMP agent. If the process cannot establish communication, it retries every 10 minutes for a period of 1 hour and then stops.

For more detailed information, including recovery actions, refer to the information about file-system errors in the *Guardian Procedure Errors and Messages Manual.*

Sample Message

```
96-07-26 11:48:59 \COMM.$ZTMX     TANDEM.TRAPMUX.D31    000103 Transient
                                  Fault - SNMP-Trap-Multiplexer \COMM.$ZTMX,
                                  event number: Process-I/O-error, fault
                                  type: IO-error, FS error: 4013 -- subcode:
                                  0, File Name: $ZTC0, Additional Info:
                                  Cannot bind to port
```

104: ZTMX-EVT-TMX-NO-MEM-SPACE

The Trap Multiplexer process ran out of memory.

Unconditional Tokens	Value
ZTMX-TKN-SUBJ-HRM-SA	<1> *Trap Multiplexer process name*
ZEMS-TKN-EVENTNUMBER	<2> ZTMX-EVT-TMX-NO-MEM-SPACE
ZEMS-TKN-TXFAULT-TYPE	<3> ZTMX-VAL-TF-MEM

Conditional Tokens

None

Message Text

```
Transient Fault - SNMP-Trap-Multiplexer <1>, event number:
<2>, fault type: <3>
```

ZTMX-TKN-SUBJ-TMX

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

ZTMX-EVT-TMX-NO-MEM-SPACE

is the event number (1044). The DDL AS clause of this value (“Process-No-Memory-Space”) is inserted in the message text following “event number:”

ZTMX-VAL-TF-MEM

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Memory-full”) is inserted in the message text.

Cause. The Trap Multiplexer process ran out of memory. The most likely cause is a problem of internal resource management.

Effect. This error might prevent a part of the Trap Multiplexer from functioning. The Trap Multiplexer might be able to correct the situation or might subsequently fail. In the latter case, additional events would be generated.

Recovery. Try using another disk as a swap device. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
96-07-26 11:48:59 \COMM.$ZTMX      TANDEM.TRAPMUX.D31    000104 Transient
                                   Fault - SNMP-Trap-Multiplexer \COMM.$ZTMX,
                                   event number: Process-No-Memory-Space,
                                   fault type: Memory-full
```

105: ZTMX-EVT-TMX--PROG-ERR

The Trap Multiplexer process experienced an internal or logic error.

Unconditional Tokens	Value
ZTMX-TKN-SUBJ-TMX	<1> *Trap Multiplexer process name*
ZEMS-TKN-EVENTNUMBER	<2> ZTMX-EVT-TMX-PROG-ERR
ZEMS-TKN-TXFAULT-TYPE	<3> ZTMX-VAL-TF-PROG

Conditional Tokens	Value
ZTMX-TKN-ERR-TEXT	<4> *descriptive text*

Message Text

```
Transient Fault - SNMP-Trap-Multiplexer <1>, event number:
<2>, fault type: <3>, error detail: <4>
```

`ZTMX-TKN-SUBJ-TMX`

identifies the subject of the event (always the Trap Multiplexer process). The DDL heading of this token (“SNMP-Trap-Multiplexer”) and the token’s value (the name of the process) are inserted in the message text.

`ZTMX-EVT-TMX-PROG-ERR`

is the event number (105). The DDL AS clause of this value (“Process-Internal-Error”) is inserted in the message text following “event number:”

`ZTMX-VAL-TF-PROG`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Internal-error”) is inserted in the message text.

`ZTMX-TKN-ERR-TEXT`

contains text describing the error. This token is of type ZSPI-TYP-STRING.

Cause. The Trap Multiplexer process experienced an internal or logic error.

Effect. The process might or might not be able to recover from the error. If the primary process cannot recover, its backup process should take over. If recovery is successful, an event announcing the takeover is generated. If the error is not recoverable and the process terminates, additional events are generated.

Recovery. If the Trap Multiplexer primary and backup processes cannot recover from the error, you can try restarting the subagent. Regardless of whether the Trap Multiplexer is able to recover, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
96-07-26 11:43:07 \COMM.$ZTMX      TANDEM.TRAPMUX.D31    000105 Transient
                                   Fault - SNMP-Trap-Multiplexer \COMM.$ZTMX,
                                   event number: Process-Internal-Error,
                                   fault type: Internal-error, error detail:
                                   Cannot create socket
```

Related Operating System Event Messages

The following four Trap Multiplexer-related events are generated by the operating system:

Process Trap Event

A “Process Trap” event is reported for a process whenever the process failed because of a Guardian trap.

Process Abended Event

A “Process Abended” event is reported for a process whenever the process is abended (stopped abnormally).

Process Suspended Event

A “Process Suspended” event is reported for a process whenever the process is suspended.

Process Activated Event

A “Process Activated” event is reported for a process whenever the process is activated.

Converting Events to Traps

Any process that generates events, including the Trap Multiplexer, can have its events translated into traps by the EMS Trap Subagent. which uses an event filter know as a routing distributor filter to determine where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert into traps. Refer to Section 9, EMS Trap Subagent, for more information.

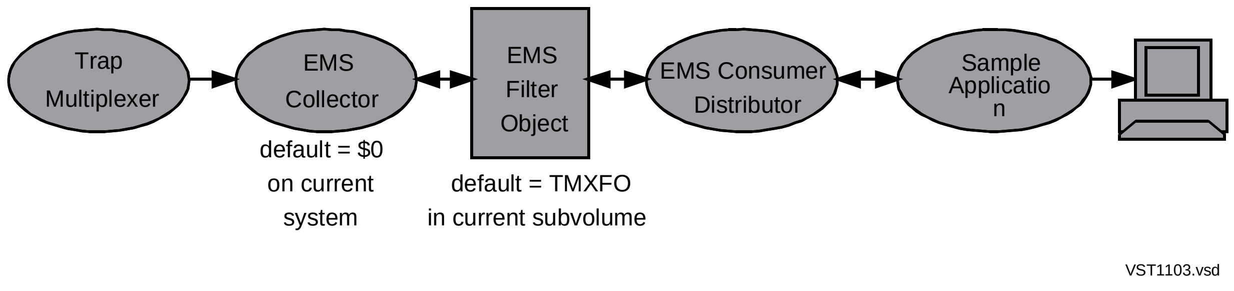
Sample EMS Application

This subsection describes a sample application that demonstrates how a management application can extract information from events created from traps received by the Trap Multiplexer.

Application Components

The sample application is in a C language program that uses Subsystem Programmatic Interface (SPI) procedures to retrieve the values of tokens from event traps. As Figure 12-3 indicates, the application interacts with an EMS consumer distributor process to retrieve trap events from an EMS collector process. Only events that meet the specifications of an EMS filter, whose source code is shown in Example 12-1, are passed to the application. The application decodes the event information to find the trap-oriented tokens, then displays their values.

Figure 12-3. Sample Application Interactions



Example 12-1. Sample Application Filter Source Code

```
filter tmxf;

begin
==[#set ztmx^val^ssid [zspi^val^tandem].
==                    [zspi^ssn^ztmx].
==                    [ztmx^val^version] ]
[#set ztmx^val^ssid TANDEM.185.D20 ]
if zspi^tkn^ssid = ssid(ztmx^val^ssid) then
   pass
else
   fail;
end;
```

Preparing the Application

Before you can run the application, the C source code and the EMS filter source code need to be compiled and the Trap Multiplexer needs to be sending trap events to an EMS collector process.

To compile the application source code, invoke the C compiler. In the following example, the source code is in a file named TMUXAPPC, and the program file name is TMUXAX; the search subvolumes identify where the compiler can locate include files used by the application:

```
c/in tmuxappc, out $s.#tmuxapp/tmuxax&
ssv0 $system.zspidef, ssv1 $system.system
```

When the source code is examined later in this section, you will notice that it specifies several compiler pragmas and includes several library files and DDL files for use during compilation.

To compile the filter source, invoke EMF after loading the DDL definitions used by the filter. In the following example, the filter source code is in a file named TMXF, and EMF compiles it into a file named TMXFO:

```
#PUSH dummy
#LOAD/KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZSPITACL
#LOAD/KEEP 1, LOADED dummy/ $SYSTEM.ZSPIDEF.ZEMSTACL
#LOAD/KEEP 2, LOADED dummy/ $SYSTEM.ZSPIDEF.ZTMXTACL
#POP dummy
EMF /IN TMXF/ TMXFO
```

Refer to Starting and Stopping the Trap Multiplexer on page 12-6 for instructions on starting the Trap Multiplexer.

Running the Application

You can optionally specify one or two startup parameters when you run the application:

```
RUN application-name [-c collector-name] [-f filter-name]
```

application-name

is the name of the executable program.

collector-name

is the name of the EMS collector process to which the Trap Multiplexer is sending trap events. If you do not specify a collector name, the primary EMS collector ($0) on your current system is assumed.

filter-name

is the name of the file containing the EMS-compiled version of the filter shown in Figure 12-4. If you do not specify a filter name, TMXFO in your current subvolume is assumed.

The following command starts the application, which has been compiled into a file named TMUXAX. The Trap Multiplexer is sending trap events to an alternate EMS collector ($ACOL), and the EMS-compiled filter is in a file named MYFILTO:

```
run tmuxax -c $acol -f myfilto
```

Example 12-2 illustrates what the sample application displays when it receives an event converted from a trap containing four varbinds.

Example 12-2. Sample Application Output

```
Waiting for the next Event.....

VersionNumber    : 0
Community        : public
Enterprise       : 1.3.6.1.4.1.169.3
Network Address  : 130.252.12.3
Generic trap No  : 6
Specific Trap No : 5
Timeticks        : 54321
Varbind Count    : 4

Varbind #1
OID   : 1.2.3
Type  : OPAQUE
Value : 1@234&5*67

Varbind #2
OID   : 1.3.2.85.6
Type  : OCTETSTRING
Value : Tandem Computers

Varbind #3
OID   : 1.3.2.208.9
Type  : NULL
Value : 0

Varbind #4
OID   : 1.2.3.56.4
Type  : IPADDR
Value : 130.252.109.237
```

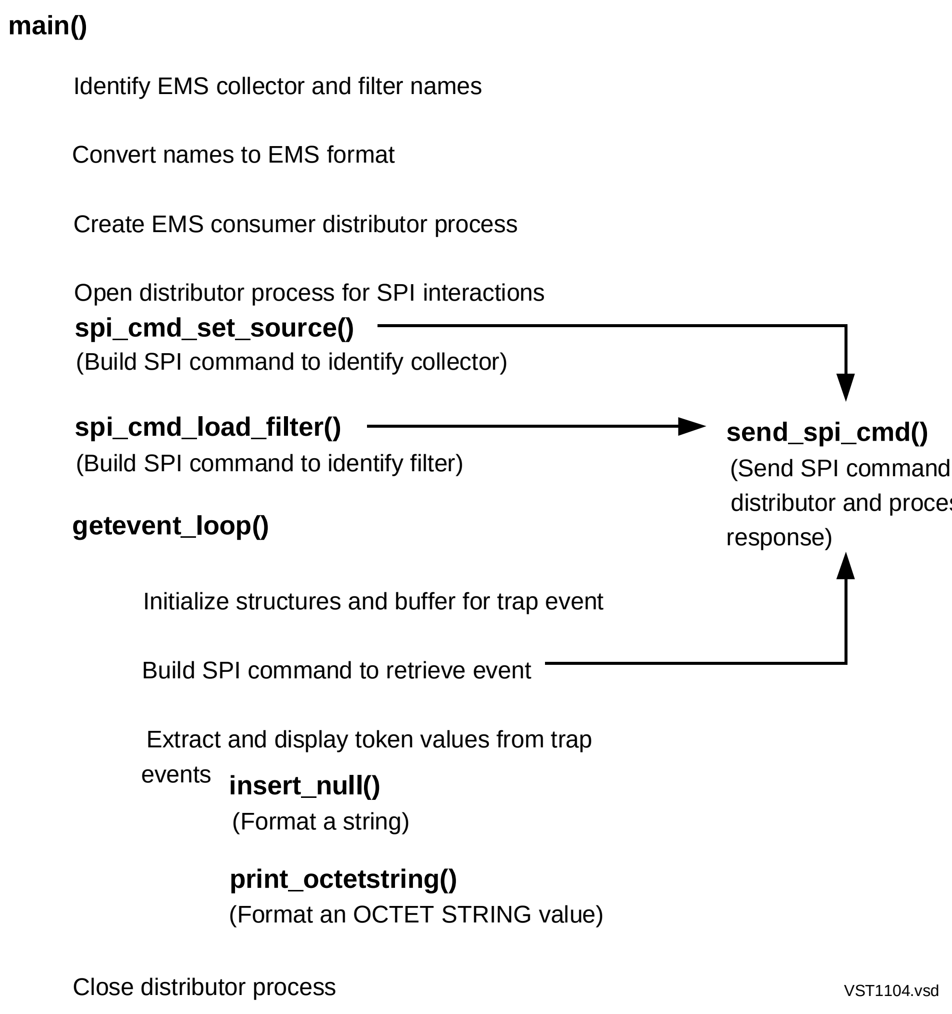
Application Control Flow

Figure 12-4 summarizes the control flow of the sample application:

- The main control block establishes communication with an EMS consumer distributor process, then calls spi_cmd_set_source().
- The spi_cmd_set_source() function builds a SPI command that identifies the EMS collector from which to obtain trap events and then calls send_spi_cmd() to process the command.
- The send_spi_cmd() function sends the SPI command to the EMS consumer distributor and processes the response. This general-purpose function is called from various functions whenever a SPI command is ready for processing.
- The spi_cmd_load_filter() function builds a SPI command that identifies the filter object file and then calls send_spi_cmd() to process the command.

- The getevent_loop() function retrieves trap events and then extracts and displays their token values, calling insert_null() or print_octetstring() functions to format strings and OCTET STRING values for display.

Figure 12-4. Sample Application Control Flow


main()
Identify EMS collector and filter names
Convert names to EMS format
Create EMS consumer distributor process
Open distributor process for SPI interactions
spi_cmd_set_source()
(Build SPI command to identify collector)
spi_cmd_load_filter()
(Build SPI command to identify filter)
getevent_loop()
Initialize structures and buffer for trap event
Build SPI command to retrieve event
Extract and display token values from trap events
insert_null()
(Format a string)
print_octetstring()
(Format an OCTET STRING value)
Close distributor process
send_spi_cmd()
(Send SPI command to distributor and process response)
VST1104.vsd

Application Source Code

The source code for the sample application, shown Example 12-3 at the end of this section, contains numbers that correspond to the following observations:

1. Several C compiler pragmas are specified.
2. Files needed to support the C library calls, Guardian procedure calls, and SPI procedure calls used by the application are included.
3. Functions defined by the application are declared.
4. The C language versions of the DDL files defining trap event tokens are included.

5. The function named send_spi_cmd() sends SPI commands built by other functions to the distributor and processes the response.
6. The function named spi_cmd_set_source() builds SPI commands to identify the EMS collector from which events are to be retrieved and then calls send_spi_cmd().
7. The function named spi_cmd_load_filter() builds SPI commands to identify the EMS filter to use and then calls send_spi_cmd().
8. The function named insert_null() terminates string values with a NULL character so that printf can be used to display these values.
9. The function named print_octetstring() formats an OCTET STRING value by replacing nonprintable characters with a period.
10. The function named getevent_loop() is an infinite loop that uses SPI commands to retrieve events and parse their tokens and then displays token values.
11. Storage for SNMP data is declared. Most values are allocated storage by using compatible ZSPI data types. Values that the application treats as strings (OCTET STRING, OBJECT IDENTIFIER, IpAddress, and Opaque values) are stored in a 1024-byte buffer named t_buffer. The following table summarizes the SPI and C types you can use when declaring storage for trap data:

Trap Data	**SPI Data Type**	**C (Wide Model) Data Type**
SNMP version number	zspi_ddl_int2_def	int
Community value	zspi_ddl_char_def	char*
Enterprise value	zspi_ddl_char_def	char*
IP address	zspi_ddl_char_def	char*
Generic trap type	zspi_ddl_int2_def	int
Specific trap type	zspi_ddl_int2_def	int
Timestamp value	zspi_ddl_int2_def	int
Varbind count	zspi_ddl_int_def	short
Varbind data type	zspi_ddl_enum_def	short
Varbind object identifier	zspi_ddl_char_def	char*
INTEGER value	zspi_ddl_int2_def	int
Counter value	zspi_ddl_int2_def	int
Gauge value	zspi_ddl_int2_def	int
OBJECT IDENTIFIER value	zspi_ddl_char_def	char
OctetString value	zspi_ddl_char_def	char*
Opaque value	zspi_ddl_char_def	char*
IpAddress value	zspi_ddl_char_def	char*
NULL value	zspi_tdt_undef	
TimeTicks value	zspi_ddl_int2_def	int

12. A buffer named event_buf is allocated for holding a trap event. ZEMS_VAL_BUFLEN provides 4200 bytes for this purpose.
13. A SPI command is built to initialize a buffer named spi_buf. This buffer, declared in the main block, is large enough to hold the longest SPI response.
14. The SPI command is processed by calling send_spi_cmd().
15. The EMS event information is extracted from spi_buf and stored in event_buf for token parsing.
16. The beginning of event_buf is identified.
17. The subsystem ID is retrieved and examined to determine whether it matches the subsystem ID of the Trap Multiplexer.
18. The event number is examined to determine whether it is a trap event.
19. The original BER-encoded trap is copied into trap_buf. SPI inserts the size of this value in the first two bytes of the buffer.
20. The value of ZTMX-TKN-TRAP-VERSION, the SNMP version in the trap PDU, is extracted and displayed.
21. The value of ZTMX-TKN-TRAP-COMMUNITY, the community name in the trap PDU, is extracted and displayed after insert_null() is called.
22. The value of ZTMX-TKN-TRAP-ENTERPRISE, the enterprise name in the trap PDU, is extracted and displayed after insert_null() is called.
23. The value of ZTMX-TKN-TRAP-NETADDR, the IP address of the trap originator, is extracted and displayed after insert_null() is called.
24. The value of ZTMX-TKN-TRAP-GENERIC, the generic trap type in the trap PDU, is extracted and displayed.
25. The value of ZTMX-TKN-TRAP-SPECIFIC, the specific trap type in the trap PDU, is extracted and displayed.
26. The value of ZTMX-TKN-TRAP-TIMETICKS, the timestamp in the trap PDU, is extracted and displayed.
27. The token list enclosing varbind information is accessed.
28. The value of ZTMX-TKN-TRAP-VARBIND-COUNT, the number of varbinds in the trap PDU, is extracted and displayed.
29. This “for” loop is executed once for each varbind.
30. The inner list delimiting one varbind is accessed.
31. The value of ZTMX-TKN-TRAP-VARBIND-OID, the object identifier of the varbind, is extracted and displayed after insert_null() is called.
32. The value of ZTMX-TKN-TRAP-VARBIND-DATATYPE, the SNMP data type of the varbind, is extracted.

33. The value of the varbind's data type is used to determine which statements to execute in order to extract and display the varbind's value. For example, if ZTMX-TKN-TRAP-VARBIND-DATATYPE has a value of ZTMX_TNM_SMI_INTEGER, the token named ZTMX_TKN_SMI_INTEGER is accessed.
34. The main block initiates communication with an EMS consumer distributor and then calls getevent_loop() to process trap events.
35. Any EMS collector and filter names passed in the invocation line are identified.
36. The collector name is resolved to a fully qualified file name.
37. The collector name is formatted for use by EMS procedures.
38. The filter name is resolved to a fully qualified file name.
39. The filter name is formatted for use by EMS procedures.
40. The subsystem ID structures are initialized.
41. The buffer for holding a SPI response is allocated by using ZCOM_VAL_MAX_RSP_BUFLEN to define the largest supported buffer (5120 bytes).
42. An EMS distributor process is started.
43. The process is opened.
44. The process is initialized as a consumer distributor process because event access is programmatic.
45. The process is closed and then re-opened to initiate SPI command/response interactions.
46. The function named spi_cmd_set_source() is called to identify the collector from which trap events are to be received.
47. The function named spi_cmd_load_filter() is called to identify the filter to use.
48. The function getevent_loop() is called to process trap events.

Example 12-3. Sample Application Source Code (page 1 of 14)

```
/************************************************************************

   TMUXAPPC :  Trap Multiplexer Application Program  : Starts an EMS
   consumer distributor using a collector and filter provided by the
   user  on the command-line parameters  "-c" and  "-f" respectively.
   By default, the program assumes $0 as the collector and tmxfo in
   the current subvolme as the filter. All filtered trap events are
   then displayed on the terminal.

   Usage :  run tmuxax [-c <collector-name>] [-f <filter-name>]

   Eg.   :  run tmuxax -c $acol  -f myfilto

   To compile the program :

   c/in tmuxappc,out $s.#tmuxapp/tmuxax;&
   ssv0 "$system.zspidef", ssv1 "$system.system"

   Note : zspidef location (ssv0) may vary according to system configuration.

************************************************************************/

#pragma WIDE                                                             <-- 1
#pragma INSPECT
#pragma SYMBOLS
#pragma NOMAP
#pragma NOLMAP
#pragma RUNNABLE
#pragma XMEM

#include <stdio.h>    nolist                                             <-- 2
#include <string.h>   nolist
#include <stdlib.h>   nolist
#include <memory.h>   nolist
#include <tal.h>      nolist
#include <CEXTDECS (DEBUG, EMSGETTKN, FILE_CLOSE_, FILE_OPEN_,  \
                    FILE_GETINFO_, FILENAME_RESOLVE_,           \
                    FILENAME_TO_OLDFILENAME_, PROCESS_CREATE_,  \
                    SSGETTKN, SSINIT, SSPUTTKN, WRITEREADX)>      nolist

/* function prototypes */

void  send_spi_cmd (char* spi_buf, short distr) ;                        <-- 3
void  spi_cmd_set_source (char* spi_buf, char* coll_name, short distr) ;
void  spi_cmd_load_filter (char* spi_buf, char* filt_name, short distr) ;
char* insert_null (char* pt_buffer, short* len) ;
void  print_octetstring (char* buffer, int buflen, char* title)  ;
void  getevent_loop (char* spi_buf, short distr) ;

#include <zspic>       nolist                                            <-- 4
#include <zemsc>       nolist
#include <ztmxc>       nolist
#include <zcomc>       nolist
```

Example 12-3. Sample Application Source Code (page 2 of 14)

```
/* These defines are used for zems structs to shorten names  */
#define SPIERRDEF zspi_ddl_error_def

/* spi error */
short spi_err;

/* declare the SSIDs using the type defs from the DDL output */
zems_val_ssid_def emsssid;
ztmx_val_ssid_def tmxssid;


/*********************************************************************/
#pragma PAGE "send_spi_cmd()"
/* Determines the  used portion of a SPI buffer  prepared  by another
   procedure. Sends the SPI command to the distributor and checks the
   command response

   Input  : SPI buffer and distributor file number

   Return : None
*/
void send_spi_cmd (char* spi_buf, short distr)                            <-- 5
{
   short ccval;                /* for cc return from writeread */
   short ibuflen  = ZCOM_VAL_MAX_RSP_BUFLEN;
   short used_len;
   short spi_ret_code;

   /* Determine how much buffer was used                       */
   spi_err = SSGETTKN ((short*)spi_buf,         /* buffer      */
                       ZSPI_TKN_USEDLEN,        /* token-id    */
                       (char*)&used_len);       /* token-value */
   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* Send the used part to the distributor                    */
   ccval = WRITEREADX(distr,                    /* file-number */
                      (char*)spi_buf,           /* buffer      */
                      used_len,                 /* write-count */
                      ZCOM_VAL_MAX_RSP_BUFLEN); /* read-count  */
   if (ccval != CCE)
      DEBUG();

   /* reset the buffer */
   spi_err = SSPUTTKN((short*)spi_buf,          /* buffer      */
                       ZSPI_TKN_RESET_BUFFER,   /* token-id    */
                       (char*)ibuflen);         /* token-value */
   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* response is in the buffer, check for the return code */
   spi_err = SSGETTKN ((short*)spi_buf,
                      ZSPI_TKN_RETCODE,
                      (char*)&spi_ret_code, 1);

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   if (spi_ret_code != ZSPI_ERR_OK)
      DEBUG();

} /* send_spi_cmd */
```

Example 12-3. Sample Application Source Code (page 3 of 14)

```
/*********************************************************************/
#pragma PAGE "spi_cmd_set_source()"
/* builds a spi command that directs the distributor to use a collector
   as a source of event messages

   Input  : Spi buffer, Collector name, and Distributor file number
   Return : None

*/
void spi_cmd_set_source (char* spi_buf, char* coll_name, short distr)          <-- 6
{
   /* initialize the spi_buf for distributor control command        */
   spi_err = SSINIT((short*)spi_buf,              /* buffer        */
                    ZEMS_VAL_BUFLEN,              /* buffer-length */
                    (short*)&emsssid,             /* ssid          */
                    ZSPI_VAL_CMDHDR,              /* header-type   */
                    ZEMS_CMD_CONTROL);            /* command       */
   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* place the connect-source-token in buffer                      */
   spi_err = SSPUTTKN((short*)spi_buf,            /* buffer        */
                       ZEMS_TKN_CONNECT_SRC_COLL,  /* token-id      */
                       coll_name);                /* token-value   */

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* send the command to the distributor */
   send_spi_cmd (spi_buf, distr);

} /* spi_cmd_set_source */


/*********************************************************************/
#pragma PAGE "spi_cmd_load_filter()"
/* builds a SPI command that loads a filter into the distributor and
   passes one filter parameter

   Input  : SPI buffer, filter name, and distributor file number
   Return : None

*/
void spi_cmd_load_filter(char* spi_buf, char* filt_name, short distr)          <-- 7
{
   /* initialize the spi_buf for distributor control commmand         */
   spi_err = SSINIT((short*)spi_buf,               /* buffer         */
                    ZEMS_VAL_BUFLEN,               /* buffer-length  */
                    (short*)&emsssid,              /* ssid           */
                    ZSPI_VAL_CMDHDR,               /* header-type    */
                    ZEMS_CMD_CONTROL);             /* command        */
   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* place the load filter token in the buffer                       */
   spi_err = SSPUTTKN((short*)spi_buf,             /* buffer         */
                       ZEMS_TKN_FILTERFILE,        /* token-id       */
                       filt_name);                 /* token-value    */
   if (spi_err != ZSPI_ERR_OK)
      DEBUG();
```

Example 12-3. Sample Application Source Code (page 4 of 14)

```
   /* send the comand to the distributor */
   send_spi_cmd (spi_buf, distr);

} /* spi_cmd_load_filter */


/************************************************************************/
#pragma PAGE "insert_null()"
/* this routine inserts a NULL at the end of an array of characters so
   that printf can print it as a string  and returns the pointer to
   the beginning of the printable string

   Input  : pointer to t_buffer and length of the data
   Return : pointer to the beginning of the string
*/
char* insert_null(char* pt_buffer, short* len)                            <-- 8
{
   char* b_ptr;              /* base pointer                         */

   pt_buffer++;              /* skip the type                        */
   *len = *pt_buffer;        /* get the length from the second byte  */
   pt_buffer++;              /* skip the length                      */
   b_ptr = pt_buffer + *len; /* point to the end of array            */
   *b_ptr = 0;               /* insert a null to print as a string   */
   return pt_buffer;

} /* insert_null */


/************************************************************************/
#pragma PAGE "print_octetstring()"
/* this routine prints an octet string variable.  Nonprintable
   chars are replaced with "."

   Input  : pointer to the buffer, length and title to be displayed
   Return : None

*/
void print_octetstring(char* buffer, int buflen, char* title)             <-- 9
{
   static char printbuf[2048];
   unsigned char c;
   int i;
   memcpy (printbuf, buffer, buflen);
   printbuf[buflen] = '\0';
   for (i = 0; i < buflen; i++)  {
      c = *buffer++;
      if (c < 32 || c >= 127)
         printbuf[i] = '.';
   }
   printf ( "%s %s\n", title, printbuf);

} /* print_octetstring */


/************************************************************************/
#pragma PAGE "getevent_loop()"
/* this procedure contains a loop to retrieve event messages. Each time
   through the loop, the procedure gets all the tokens pertaining to this
   event and displays them on the terminal.
```

Example 12-3. Sample Application Source Code (page 5 of 14)

```
   Input  : pointer to spi_buffer and distributor file number
   Return : None

*/
void getevent_loop(char* spi_buf, short distr)                              <-- 10
{

   char* event_buf;                      /* pointer to event data      */
   short byteoffset, i;

   zspi_ddl_int2_def t_version;          /* trap version               */      <-- 11
   zspi_ddl_int2_def t_generic;          /* generic trap number        */
   zspi_ddl_int2_def t_specific;         /* specific trap number       */
   zspi_ddl_int2_def t_timeticks;        /* timeticks                  */
   zspi_ddl_int_def  t_varbind_count;    /* number of var binds        */
   zspi_ddl_enum_def t_varbind_datatype; /* data type of each var bind */
   zspi_ddl_char_def t_null = 0;
   zspi_ddl_int2_def t_integer;
   zspi_ddl_int2_def t_counter;
   zspi_ddl_int2_def t_gauge;

   short evt_num;
   unsigned char trap_buf[ZEMS_VAL_BUFLEN];
   ztmx_val_ssid_def tmpssid;

   char  t_buffer[1024];            /* temp buffer for string manipulation */
   char* pt_buffer = t_buffer;      /* pointer to the temp buffer          */
   short len;
   short ibuflen  = ZEMS_VAL_BUFLEN;

 /* malloc memory for the event buffer */

   event_buf = malloc (ZEMS_VAL_BUFLEN);                                       <-- 12
   if (event_buf == NULL)
      DEBUG();

   setmem (trap_buf,sizeof(trap_buf),0);  /* initialise the trap buffer */

   /* begin an infinite loop that collects and displays events */
   while (1)
   {

      /* initialize spi_buf for getevent spi command                */
      spi_err = SSINIT((short*)spi_buf,           /* buffer         */       <-- 13
                       ZEMS_VAL_BUFLEN,           /* buffer length  */
                       (short*)&emsssid,          /* ssid           */
                       ZSPI_VAL_CMDHDR,           /* header-type    */
                       ZEMS_CMD_GETEVENT);        /* command        */

      if (spi_err != ZSPI_ERR_OK)
         DEBUG();

      /* send getevent command to distributor */
      printf ("\n\nWaiting for the next Event.....\n\n");
      send_spi_cmd (spi_buf, distr);                                           <-- 14

      /* Extract the event into the buffer  */
      spi_err = SSGETTKN ((short*) spi_buf,                                    <-- 15
                          ZEMS_TKN_EVENT,
                          event_buf,
                          1);

      if (spi_err != ZSPI_ERR_OK)
         DEBUG();
```

Example 12-3. Sample Application Source Code (page 6 of 14)

```
/* reset event buffer  */
   event_buf = event_buf + sizeof(short); /* skip the length field */
   spi_err = SSPUTTKN ((short*)event_buf,                                    <-- 16
                       ZSPI_TKN_RESET_BUFFER,
                       (char*) (ibuflen - sizeof(short)));

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   /* check if the event obtained is from a trap mux ssid       */
   spi_err = EMSGETTKN ((short*)event_buf,      /* buffer        */          <-- 17
                      ZSPI_TKN_SSID,            /* token-id      */
                      (char*)&tmpssid,          /* token-value   */
                      1);                       /* index         */

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   if (strcmp(tmpssid.u_z_filler.z_filler,ZSPI_VAL_TANDEM))
   {
      printf ("A non-HP event has been reported\n");
      continue;
   }

   if (tmpssid.z_number != ZSPI_SSN_ZTMX)
   {
      printf ("A non-Trap-Multiplexer event has been reported\n");
      continue;
   }

   /* check whether the event obtained is a trap event         */
   spi_err = EMSGETTKN ((short*)event_buf,     /* buffer        */           <-- 18
                      ZEMS_TKN_EVENTNUMBER,    /* token-id      */
                      (char*)&evt_num,         /* token-value   */
                      1);                      /* index         */

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();

   if ((evt_num != ZTMX_EVT_TRAP_COLDSTART)    &&
       (evt_num != ZTMX_EVT_TRAP_WARMSTART)    &&
       (evt_num != ZTMX_EVT_TRAP_LINKDOWN )    &&
       (evt_num != ZTMX_EVT_TRAP_LINKUP   )    &&
       (evt_num != ZTMX_EVT_TRAP_AUTH_FAILURE) &&
       (evt_num != ZTMX_EVT_TRAP_EGP_LOSS )    &&
       (evt_num != ZTMX_EVT_TRAP_ENT_SPECIFIC))
   {
      printf ("A non-trap event has been reported\n");
      continue;
   }

   spi_err = EMSGETTKN ((short*)event_buf,     /* buffer         */          <-- 19
                       ZTMX_TKN_TRAP_PDU,      /* token-id       */
                       (char*)&trap_buf,       /* token-value    */
                       1);                     /* index          */

   if (spi_err != ZSPI_ERR_OK)
      DEBUG();
```

Example 12-3. Sample Application Source Code (page 7 of 14)

```
    spi_err = EMSGETTKN ((short*)event_buf,                                <-- 20
                       ZTMX_TKN_TRAP_VERSION,
                       (char*) &t_version,          /* trap version */
                       1);                          /* index        */

    if (spi_err != ZSPI_ERR_OK)
       DEBUG();

    printf ("VersionNumber    : %d \n",t_version);


    spi_err = EMSGETTKN ((short*)event_buf,                                <-- 21
                       ZTMX_TKN_TRAP_COMMUNITY,     /* community    */
                       (char*)&t_buffer,
                       1);                          /* index        */

    if (spi_err != ZSPI_ERR_OK)
       DEBUG();

    pt_buffer = insert_null (t_buffer, &len);
    print_octetstring (pt_buffer, len, "Community        :");


    spi_err = EMSGETTKN ((short*)event_buf,                                <-- 22
                       ZTMX_TKN_TRAP_ENTERPRISE,    /* enterprise   */
                       (char*)&t_buffer,
                       1);                          /* index        */

    if (spi_err != ZSPI_ERR_OK)
       DEBUG();

    pt_buffer = insert_null (t_buffer, &len);
    printf ("Enterprise       : %s \n",pt_buffer);


    spi_err = EMSGETTKN ((short*)event_buf,                                <-- 23
                       ZTMX_TKN_TRAP_NETADDR,       /* net address  */
                       (char*)&t_buffer,
                       1);                          /* index        */

    if (spi_err != ZSPI_ERR_OK)
       DEBUG();

    pt_buffer = insert_null (t_buffer, &len);
    printf ("Network Address  : %s \n",pt_buffer);


    spi_err = EMSGETTKN ((short*)event_buf,                                <-- 24
                       ZTMX_TKN_TRAP_GENERIC,      /* generic trap no */
                       (char*) &t_generic,
                       1);                         /* index           */

    if (spi_err != ZSPI_ERR_OK)
       DEBUG();

    printf ("Generic trap No  : %d \n",t_generic);
```

Example 12-3. Sample Application Source Code (page 8 of 14)

```
spi_err = EMSGETTKN ((short*)event_buf,                                    <-- 25
                   ZTMX_TKN_TRAP_SPECIFIC,    /* specific trap no */
                   (char*) &t_specific,
                   1);                        /* index            */

if (spi_err != ZSPI_ERR_OK)
DEBUG();

printf ("Specific Trap No : %d \n",t_specific);


spi_err = EMSGETTKN ((short*)event_buf,                                    <-- 26
                   ZTMX_TKN_TRAP_TIMETICKS,   /* timeticks    */
                   (char*) &t_timeticks,
                   1);                        /* index        */

if (spi_err != ZSPI_ERR_OK)
   DEBUG();

printf ("Timeticks        : %d \n",t_timeticks);


/* enter the data list to get the next token  */
spi_err = EMSGETTKN ((short*)event_buf,                                    <-- 27
                   ZSPI_TKN_DATALIST);

if (spi_err != ZSPI_ERR_OK)
   DEBUG();

spi_err = EMSGETTKN ((short*)event_buf,                                    <-- 28
                   ZTMX_TKN_TRAP_VARBIND_COUNT, /* varbind count */
                   (char*) &t_varbind_count,
                   1);                          /* index         */

if (spi_err != ZSPI_ERR_OK)
   DEBUG();

printf ("Varbind Count    : %d \n",t_varbind_count);


for(i = 1 ; i <= t_varbind_count ; i++)                                    <-- 29
{
   /* enter the data list to get the varbind  */
   spi_err = EMSGETTKN ((short*)event_buf,                                 <-- 30
                      ZSPI_TKN_DATALIST);

   if (spi_err !=ZSPI_ERR_OK)
      DEBUG();

   printf ("\nVarbind #%d\n",i);
```

Example 12-3. Sample Application Source Code (page 9 of 14)

```
/* get the varbind OID   */
spi_err = EMSGETTKN ((short*)event_buf,                                <-- 31
                   ZTMX_TKN_TRAP_VARBIND_OID,  /* varbind OID */
                   (char*)&t_buffer,
                   1);                         /* index       */

if (spi_err != ZSPI_ERR_OK)
   DEBUG();

pt_buffer = insert_null (t_buffer, &len);
printf ("OID   : %s \n",pt_buffer);

/* get the varbind Data type   */
spi_err = EMSGETTKN ((short*)event_buf,                                <-- 32
                   ZTMX_TKN_TRAP_VARBIND_DATATYPE,
                   (char*) &t_varbind_datatype,
                   1);

if (spi_err != ZSPI_ERR_OK)
   DEBUG();

switch (t_varbind_datatype)                                            <-- 33
{
  case ZTMX_TNM_SMI_NULL          : printf ("Type  : NULL \n");
                                    spi_err =
                                            EMSGETTKN ((short*)
                                            event_buf,
                                            ZTMX_TKN_SMI_NULL,
                                            &t_null,
                                            1);
                                    printf ("Value : %d\n",t_null);
                                    break;

  case ZTMX_TNM_SMI_INTEGER       : printf ("Type  : INTEGER \n");
                                    spi_err =
                                            EMSGETTKN ((short*)
                                            event_buf,
                                            ZTMX_TKN_SMI_INTEGER,
                                            (char*) &t_integer,
                                            1);
                                    printf ("Value : %d\n",t_integer);
                                    break;

  case ZTMX_TNM_SMI_OCTETSTRING   : printf ("Type  : OCTETSTRING \n");
                                    spi_err =
                                            EMSGETTKN ((short*)
                                            event_buf,
                                            ZTMX_TKN_SMI_OCTETSTRING,
                                            (char*)&t_buffer,
                                            1);
                                    pt_buffer = insert_null (t_buffer,&len);
                                    /* Note: Nondisplayable portion
                                       of the string will be replaced with
                                       a special character (".")         */
                                  print_octetstring(pt_buffer,len,"Value:");
                                    break;
```

Example 12-3. Sample Application Source Code (page 10 of 14)

```
case ZTMX_TNM_SMI_OID           : printf ("Type  : OID \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_OID,
                                          (char*)&t_buffer,
                                          1);
                                  pt_buffer = insert_null (t_buffer,&len);
                                  printf ("Value : %s\n",pt_buffer);
                                  break;

case ZTMX_TNM_SMI_IPADDR        : printf ("Type  : IPADDR \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_IPADDR,
                                          (char*)&t_buffer,
                                          1);
                                  pt_buffer = insert_null (t_buffer,&len);
                                  printf ("Value : %s\n",pt_buffer);
                                  break;

case ZTMX_TNM_SMI_COUNTER       : printf ("Type  : COUNTER \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_COUNTER,
                                          (char*) &t_counter,
                                          1);
                                  printf ("Value : %d\n",t_counter);
                                  break;

case ZTMX_TNM_SMI_GAUGE         : printf ("Type  : GAUGE \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_GAUGE,
                                          (char*) &t_gauge,
                                          1);
                                  printf ("Value : %d\n",t_gauge);
                                  break;

case ZTMX_TNM_SMI_TICKS         : printf ("Type  : TIMETICKS \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_TIMETICKS,
                                          (char*) &t_timeticks,
                                          1);
                                  printf ("Value : %d\n",t_timeticks);
                                  break;

case ZTMX_TNM_SMI_OPAQUE        : printf ("Type  : OPAQUE \n");
                                  spi_err =
                                          EMSGETTKN ((short*)
                                          event_buf,
                                          ZTMX_TKN_SMI_OPAQUE,
                                          (char*)&t_buffer,
                                          1);
                                  pt_buffer = insert_null (t_buffer,&len);
                                  printf ("Value : %s\n",pt_buffer);
                                  break;
```

Example 12-3. Sample Application Source Code (page 11 of 14)

```
          default                         : printf ("Type  : UNKNOWN \n");
                                            break;
        }   /* end of case */

        /* exit from the current list  */
        spi_err = EMSGETTKN ((short*)event_buf, ZSPI_TKN_ENDLIST);

        if (spi_err !=ZSPI_ERR_OK)
           DEBUG();

     }  /* end of for loop */

   } /*  end while */

} /* end getevent_loop */


/*************************************************************************/
#pragma PAGE "main program code"
/*
   Starts an EMS consumer distributor and calls procedures that use
   SPI commands to connect to the source collector, to load a filter
   and to  position the  distributor within the log files. Then the
   getevent_loop procedure is called to retrieve and process event
   messages.
*/

main(int argc, char**argv)                                           <-- 34
{

 short error;
 short startup_msg_actual_len;
 startup_msg_type  ci_startup;

 char    *cptr;
 int     ccval = 0;

 #define proc_name_len 40
 char  proc_name[proc_name_len];
 char* whoami,opt;
 short err_detail;
 short proc_len;
 char* startup_default_str;
 char* coll_name = "$0";               /* default collector name  */
 char* filt_name = "tmxfo";            /* default event filter    */

 #define file_name_Dxx_maxlen 36
 char filt_name_Dxx[file_name_Dxx_maxlen]; /* D series filename       */
 char filt_name_Cxx[24];                   /* C series filename       */
 char coll_name_Dxx[file_name_Dxx_maxlen];
 char coll_name_Cxx[24];
 short file_name_len;
 short distr;                              /* distributor file number */
 char* spi_buf;

 char usage[] = "Usage: %s  [-c <collector-name>]  [-f <filter-name>]\n\n";

 /* Get the default vol and subvol to set up the internal filenames later */
 startup_default_str = getenv ("DEFAULTS");
 whoami = *argv++;
 argc--;
```

Example 12-3. Sample Application Source Code (page 12 of 14)

```
/*  extract the command-line parameters  */
while (( argc != 0 ) && ( **argv == '-' ))                                  <-- 35
{
  opt = *( *argv + 1 );
  argv++, argc--;

  switch (opt)
  {

  case 'c' :  coll_name = *argv;
              argv++;
              argc--;
              break;

  case 'f' :  filt_name = *argv;
              argv++;
              argc--;
              break;

  default  :  fprintf (stderr,usage,whoami);
              exit(1);

  } /* end of case */

} /* end of while */

/* this will convert a partial filename into an external filename          */
error = FILENAME_RESOLVE_ (coll_name,                  /* partial name     */ <-- 36
                  (short)strlen(coll_name),            /* length           */
                  coll_name_Dxx,                       /* full name        */
                  file_name_Dxx_maxlen,                /* max length       */
                  &file_name_len,                      /* full name length */
                  ,,,,,
                  startup_default_str,                 /* defaults         */
                  (short)strlen(startup_default_str)); /* length           */
if (error)
{
  printf ("FILENAME_RESOLVE_ returned %d\n", error);
  exit (1);
}

/* this will convert an external filename into an internal filename for
   passing it to the EMS routines                                          */
error = FILENAME_TO_OLDFILENAME_ (coll_name_Dxx,      /* file name        */ <-- 37
                                  file_name_len,      /* length           */
                          (short*)coll_name_Cxx);     /* old filename     */
if (error)
{
   printf ("FILENAME_TO_OLDFILENAME_ returned %d\n", error);
   exit (1);
}

/* this will convert a partial filename into an external filename          */
error = FILENAME_RESOLVE_ (filt_name,                  /* partial name     */ <-- 38
                  (short)strlen(filt_name),            /* length           */
                  filt_name_Dxx,                       /* full name        */
                  file_name_Dxx_maxlen,                /* max length       */
                  &file_name_len,                      /* full name length */
                  ,,,,,
                  startup_default_str,                 /* defaults         */
                  (short)strlen(startup_default_str)); /* length           */
```

Example 12-3. Sample Application Source Code (page 13 of 14)

```
if (error)
{
   printf ("FILENAME_RESOLVE_ returned %d\n", error);
   exit (1);
}

/* this will convert an external filename into an internal filename for
   passing it to the EMS routines                                      */
error = FILENAME_TO_OLDFILENAME_ (filt_name_Dxx,       /* filename         */ <-- 39
                                  file_name_len,       /* length           */
                          (short*)filt_name_Cxx);      /* old filename     */

if (error)
{
   printf ("FILENAME_TO_OLDFILENAME_ returned %d\n", error);
   exit (1);
}


/* initialize sub-system-ids */

cptr = strncpy(emsssid.u_z_filler.z_filler,ZSPI_VAL_TANDEM,8);                <-- 40
emsssid.z_number  = ZSPI_SSN_ZEMS;
emsssid.z_version = ZEMS_VAL_VERSION;

cptr = strncpy(tmxssid.u_z_filler.z_filler, ZSPI_VAL_TANDEM, 8);
tmxssid.z_number  = ZSPI_SSN_ZTMX;
tmxssid.z_version = ZTMX_VAL_VERSION;

/* malloc some memory for spi buffers */

spi_buf = malloc(ZCOM_VAL_MAX_RSP_BUFLEN);                                    <-- 41
if (spi_buf == NULL)
   DEBUG();

/* Initialize the startup message structure */
error = get_startup_msg (&ci_startup, &startup_msg_actual_len);
ci_startup.param[0] = 0;

/* create a new process */
ccval = PROCESS_CREATE_ ("$system.system.emsdist",    /* program file  */     <-- 42
             (short)strlen("$system.system.emsdist"), /* length        */
             ,,,,,,,,,                                /* optional      */
             &err_detail,
             4,                                       /* name option   */
             ,,
             proc_name,                               /* process name  */
             proc_name_len,                           /* max length    */
             &proc_len);                              /* actual length */

if (ccval)
{
   printf ("PROCESS_CREATE_ returned : %d\n", ccval);
   exit(1);
}
```

Example 12-3. Sample Application Source Code (page 14 of 14)

```
 /* Open the new process to pass this startup message */
 ccval = FILE_OPEN_(proc_name,         /* file name    */                   <-- 43
                   proc_len,          /* length       */
                   &distr);           /* file number  */

 if (ccval)
 {
    printf ("FILE_OPEN_ returned %d\n", ccval);
    exit (1);
 }

 /* Add "TYPE CONSUMER" to startup msg to send to EMSDIST */
 strcat (ci_startup.param, "TYPE CONSUMER");                                <-- 44

 /* Now write the startup struct */
 ccval = WRITEREADX (distr,                            /* file number   */
                   (char*)&ci_startup,                /* buffer        */
                   (short)(sizeof(startup_msg_type)    /* write-count   */
                   - sizeof(ci_startup.param)
                   + strlen ("TYPE CONSUMER") + 1),
                   0);                                /* read-count    */

 FILE_GETINFO_ (distr, (short*)&error);
 if ( ! ( error == 0 || error == 70 ))
 {
    printf ("WRITEREADX returned error: %d\n", error);
    exit (1);
 }

 /* We can now close the distributor */
 FILE_CLOSE_ (distr);                                                       <-- 45

 /* Prepare to reopen the distributor using spi-interface */
 proc_name[proc_len] = '\0';
 strcat (proc_name, ".#ZSPI");
 ccval = FILE_OPEN_(proc_name,                          /* file name   */
                  (short)(proc_len+strlen(".#ZSPI")),   /* length      */
                  (short*)&distr);                      /* file number */

 if (ccval)
 {
    printf ("FILE_OPEN_ returned : %d\n", ccval);
    exit(1);
 }

 /* Tell collector to use log files as event-message source */
 spi_cmd_set_source (spi_buf, coll_name_Cxx, distr);                        <-- 46

 /* Load filter and filter parameters into the distributor */
 spi_cmd_load_filter (spi_buf, filt_name_Cxx, distr);                       <-- 47

 /* start collecting the EMS events */
 getevent_loop (spi_buf, distr);                                            <-- 48

 /* close the distributor */
 FILE_CLOSE_ (distr);

} /* main */
```

13 IPX/SPX Subagent (G-Series)

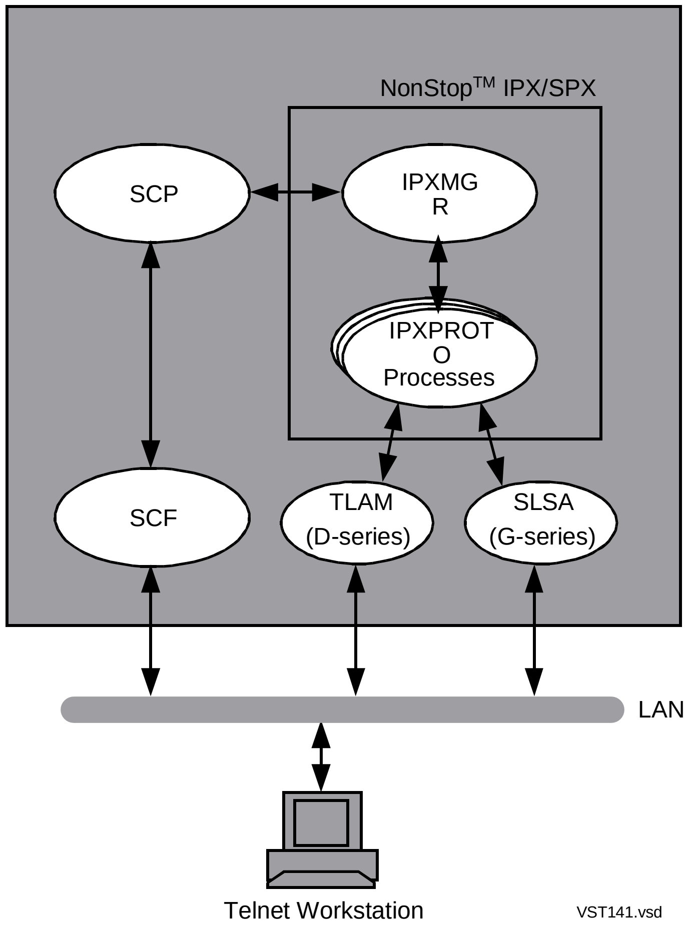
The **IPX/SPX Subagent** facilitates the management of NonStop IPX/SPX processes known as IPX/SPX Protocol (IPXPROTO) processes. **NonStop IPX/SPX** is an HP software product that provides access to NonStop Kernel systems over an Ethernet or Token Ring LAN using Novell's NetWare protocol suite: Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX). **IPXPROTO processes** provide support for NetWare protocols (IPX, SPX, NCP, RIP, and SAP), enabling connectivity, over a local area network (LAN) between NonStop Kernel systems and products using the protocols.

The subagent supports one of the groups in Novell's Management Information Base (MIB) as well as several groups defined by HP containing objects that help you manage IPXPROTO processes and the subagent process itself.

Architectural Overview

As Figure 13-1 indicates, NonStop IPX/SPX consists of one NonStop IPX/SPX Manager (IPXMGR) process and one or more IPXPROTO processes per node.

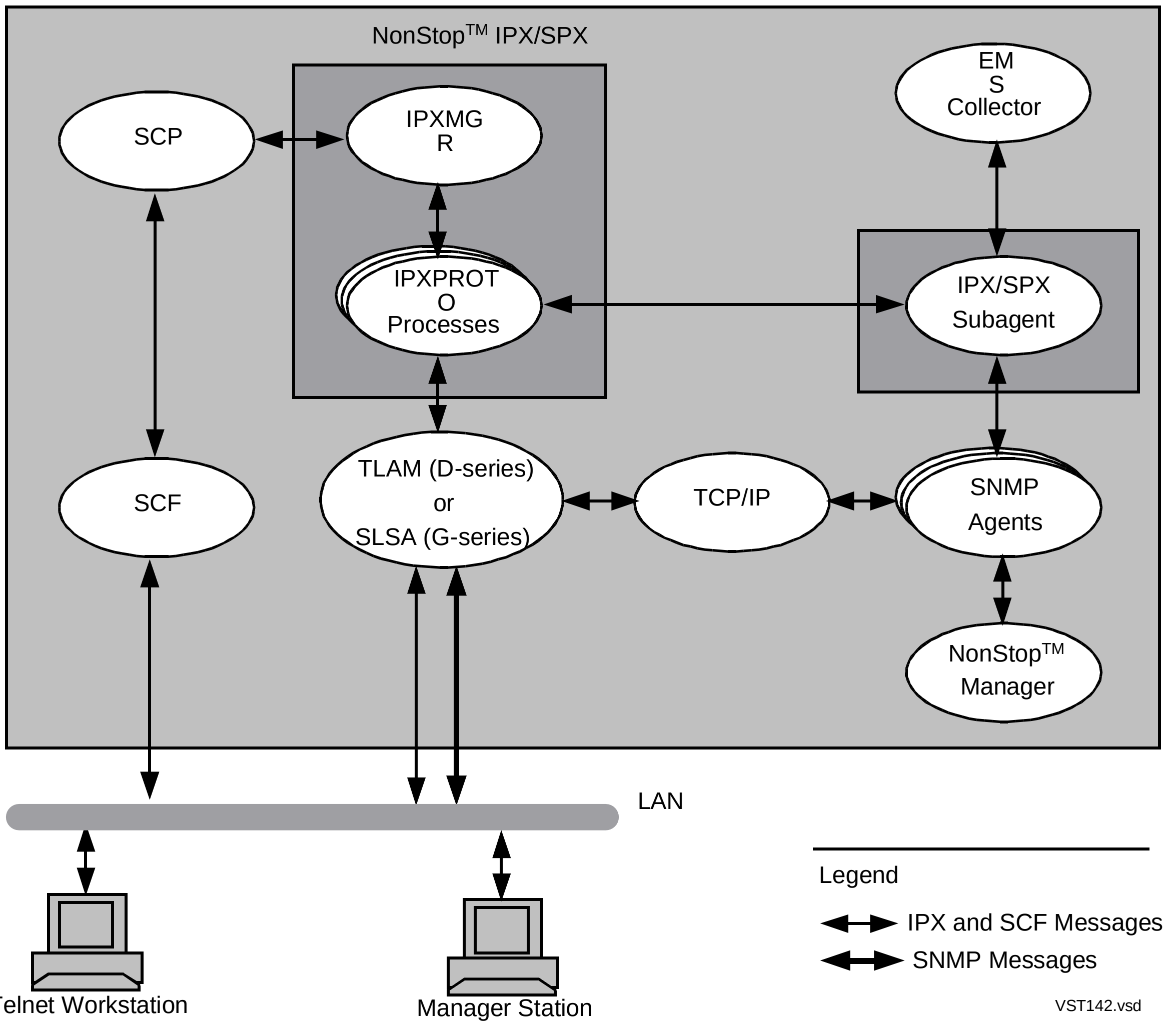
Figure 13-1. NonStop IPX/SPX Interactions



The IPXPROTO processes send IPX/SPX frames to and receive them from the LAN by means of the ServerNet LAN Systems Access (SLSA) subsystem. The IPXMGR process enables management of IPXPROTO processes using the Subsystem Control Facility (SCF).

The IPX/SPX Subagent lets you manage one or more IPXPROTO processes using an SNMP manager. As Figure 13-2 illustrates, the subagent interacts with one or more SNMP agent processes to receive and respond to manager requests. SNMP requests arriving over the LAN are routed using the TCP/IP network protocol. Managers that reside on NonStop Kernel systems can transmit requests to SNMP agents using interprocess communications. The subagent sends Event Management System (EMS) events to an EMS collector when activities of significance occur.

Figure 13-2. IPX/SPX Subagent Interactions



Although not shown in Figure 13-2, the IPX/SPX Subagent can interact over Expand with IPXPROTO processes, SNMP agent processes, and an EMS process running on remote nodes.

The IPX/SPX Subagent MIB

The subagent MIB contains objects that characterize IPXPROTO processes as well as objects that characterize the subagent process:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  novell (23)
                     mibDoc (2)
                        ipx (5)
                           ipxSystem (1)          <— IPXPROTO process objects
                  tandem (169)
                     nonstopsystems (3)
                        zipx (164)                <— IPXPROTO process objects
                           tdmNwInfo (1)
                           tdmNwIPX (2)
                           tdmNwSPX (3)
                           tdmNwNCP (4)
                           tdmNwServer (5)
                        zisa (196)                  <— IPX/SPX Subagent process objects
                           tdmNwSubagent (1)
                              tdmNwSaConfig (1)
                              tdmNwSaAgentNamesTable (2)
                              tdmNwSaIPXPROTONamesTable (3)
```

IPXPROTO Process Objects

The MIB objects that support IPXPROTO process management are located in two groups:

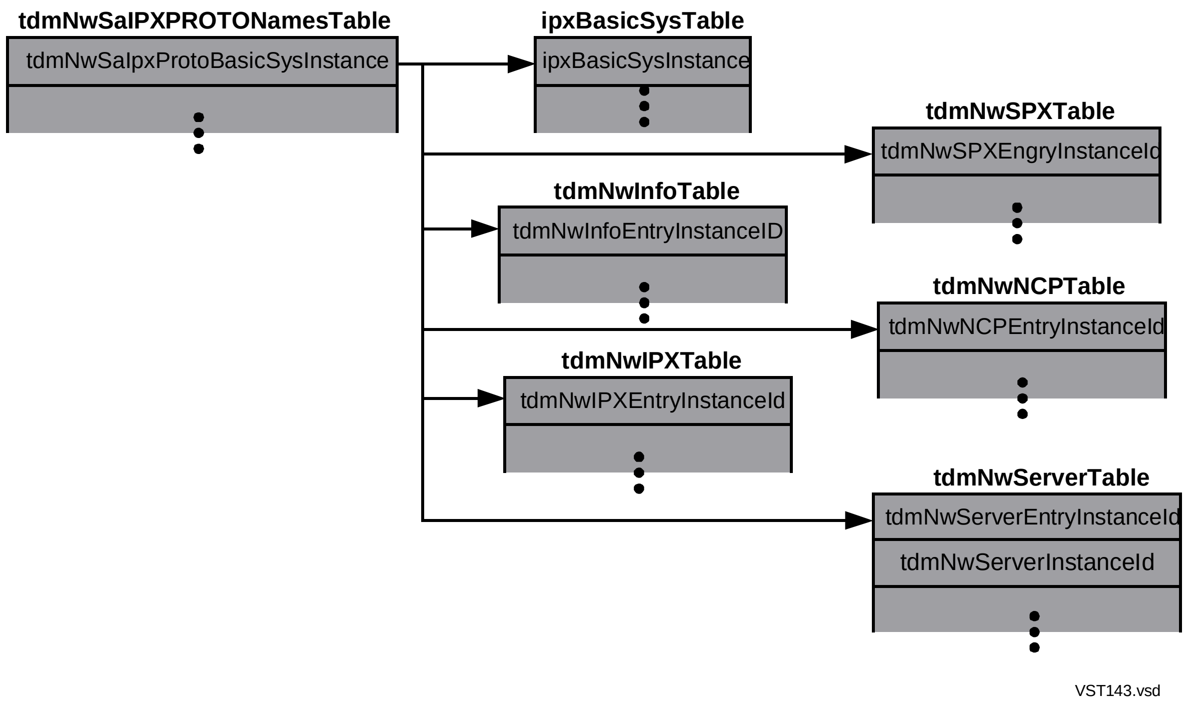
- The ipxSystem group, defined by Novell, provides information about the basic IPX protocol stack. This group consists of one table, the **ipxBasicSysTable**, whose entries characterize IPX packet-handling statistics for IPXPROTO processes. The subagent provides read-only access to all the objects in the ipxSystem group.
- The zipx group, defined by HP, gives read-only access to various IPXPROTO process characteristics. This group contains five groups, each of which consists of one table:
  - The tdmNwInfo group contains the **tdmNwInfoTable**, whose entries describe attributes of IPXPROTO processes, such as the name of the TLAM port or SLSA logical interface (LIF) being used.
  - The tdmNwIPX group contains the **tdmNwIPXTable**, whose entries provide statistical information about the IPX layer of IPXPROTO processes.

- The tdmNwSPX group contains the **tdmNwSPXTable**, whose entries provide statistical information about the SPX layer of IPXPROTO processes.
- The tdmNwNCP group contains the **tdmNwNCPTable**, whose entries provide statistical information about the NCP layer of IPXPROTO processes.
- The tdmNwServer group contains the **tdmNwServerTable**, whose entries describe paths to Novell servers.

The information available in these six tables is derived by subagent interactions with IPXPROTO processes that the subagent process is configured to monitor. Information available from the zipx group tables is also available using SCF commands, as described in zipx Group on page 13-23.

Each IPXPROTO process the subagent is configured to monitor has an entry in the six tables in the ipxSystem and zipx groups and in a table in the zisa group, the tdmNwSaIPXPROTONamesTable. The entries are created and linked by means of an index the subagent derives when an entry for an IPXPROTO process is added to the tdmNwSaIPXPROTONamesTable. Figure 13-3 illustrates this linkage.

Figure 13-3. Indexes Linking MIB Table Entries



IPX/SPX Subagent Process Objects

The zisa group, along with the manager application, facilitates managing the subagent process. The zisa group is a MIB group that contains the following group and two tables:

- The **tdmNwSaConfig group** supports management of subagent process attributes, such as process priority and EMS collector process name.
- The **tdmNwSaAgentNamesTable** entries describe SNMP agents with which the subagent is configured to communicate.
- The **tdmNwSaIPXPROTONamesTable** entries describe IPXPROTO processes that the subagent is configured to monitor.

Many of the MIB objects in this group and tables have corresponding startup parameters, so you have two means by which the subagent process can be managed. More about subagent process configuration appears in Configuration on page 13-12.

Refreshing IPXPROTO Object Values

When the subagent is started, it does not immediately refresh the values of IPXPROTO objects in its MIB. The values of objects in each of the six tables containing IPXPROTO process values are refreshed for the first time after startup when the subagent accesses the table to process an SNMP request:

ipxBasicSysTable
tdmNwInfoTable
tdmNwIPXTable
tdmNwSPXTable
tdmNwNCPTable
tdmNwServerTable

Subsequently, table values are refreshed when a table is accessed only if the number of minutes associated with the subagent's refresh timer have elapsed since the last time an object in the table was accessed. For example, if you access a table at 12:15 and the refresh timer interval is set at 10 minutes, the timer stops running at 12:25. If the table is accessed at 12:30, the table values are refreshed before the request is processed, and the timer is reset. The table will not be refreshed again until after the timer stops running and before the table is accessed again.

At startup, you can specify a refresh timer value using the -r startup parameter:

```
RUN IPXSA /NAME $ZISA, NOWAIT/ -r refresh-interval
```

The refresh interval is the duration, in minutes, of the refresh timer for each of the six tables listed above. By default, the refresh interval is 10 minutes.

After startup, you can change the value of tdmNwSaRefreshTimer in the tdmNwSaConfig group to specify a different interval. Setting the value of this object to zero causes the subagent to retrieve new data from an IPXPROTO process every time it receives a request. You can also force data refresh for all the tables on demand by setting the value of tdmNwSaRefreshNow in the tdmNwSaConfig group to `forceDynamicRefresh`.

The subagent obtains statistical values for zipx group objects from statistics maintained by IPXPROTO processes the subagent is configured to monitor. Two objects in tdmNwInfoTable entries provide information about these statistics:

- The value of tdmNwInfoResetTime indicates the time at which the statistics were last initialized (set to zero). Statistics are initialized by using the SCF STATS PROCESS command with the RESET option.
- The value of tdmNwInfoSampleTime indicates the time at which the statistics were last sampled by the subagent. Use this object to determine whether the IPXPROTO process values are current enough for your needs.
- When the subagent retrieves values from an IPXPROTO process, it waits 10 seconds for a reply from the process. If it receives no reply, the subagent generates a NoSuchName error in response to the request that fostered IPXPROTO process access. The subagent also generates an object-unavailable EMS event and sets the values of two objects in the tdmNwSaIPXPROTONames Table:
- The value of tdmNwSaIpxProtoState is set to `disconnected`.
- The value of tdmNwSaIpxProtoStateReason is set to `resourceproblem`.

The timeout value is not configurable.

Connections Between Subagent and Other Processes

The IPX/SPX Subagent communicates with various processes:

One or more IPXPROTO processes
One or more SNMP agent processes
One EMS collector process

The subagent regularly monitors all the processes with which it communicates to determine whether connections to the processes are intact. When a connection has been lost for any reason other than an SNMP manager request, the subagent attempts to re-establish the connection. This feature minimizes the need for your monitoring the state of subagent connections. The rate at which the subagent attempts to re-establish lost connections, known as the keep-alive rate, is set using the -k startup parameter or its corresponding MIB object, tdmNwSaKeepAliveTimer.

You identify processes with which the subagent should interact when you start the subagent. You can modify or extend process connections after startup by setting the value of objects in the zisa group:

- The -i startup parameter identifies IPXPROTO processes that the subagent should monitor. After startup, you use tdmNwSaIPXPROTONamesTable group objects to control IPXPROTO process connections.

- The -a startup parameter identifies SNMP agent processes that the subagent should accept requests from. After startup, you use tdmNwSaAgentNamesTable objects to control SNMP agent process connections.
- The -c startup parameter identifies the EMS collector process that the subagent should send events to. After startup, you use tdmNwSaConfig group objects to control the EMS collector connection.

Each of the connections has a read-write object state value of `disconnected` or `connected`, which describes whether a connection is currently inactive or active, respectively. Object state values for IPXPROTO, SNMP agent, and EMS processes are associated, respectively, with these state objects:

tdmNwSaIpxProtoState
tdmNwSaAgentState
tdmNwSaEmsCollectorState

When the subagent starts, it establishes connections with IPXPROTO, SNMP agent, and EMS collector processes named in the startup line and sets the value of their MIB state objects to `connected` if a connection could be established. After startup, you can establish additional process connections using Set commands on objects in the zisa group to add processes:

Process	**SNMP Operation**
IPXPROTO process	Add an entry to the tdmNwSaIPXPROTONamesTable.
SNMP agent process	Add an entry to the tdmNwSaAgentNamesTable.
EMS collector process	Change the value of the tdmNwSaEmsCollectorName object in the tdmNwSaConfig group.

If you do not set the state of any new process to `connected`, the subagent attempts to establish a connection when its keep-alive timer runs down.

Each state object has a corresponding state reason object:

tdmNwSaIpxProtoStateReason
tdmNwSaAgentStateReason
tdmNwSaEmsCollectorStateReason

The state reason object is a read-only object that provides an explanation for the current object state value. Values that the state reason object can have are:

State Reason Object Value	**Meaning**
`rowcreate`	The connection was just created by a startup parameter or a Set operation providing the name of a process.
`operatorrequest`	A Set operation on the object state was successful.

`subagentkeepalive`	The subagent successfully connected to a previously connected process that failed. The rate at which the subagent attempts reconnections, called the keep-alive interval, is controlled by the -k startup parameter. By default, the subagent attempts to reconnect every 60 minutes.
`invalidprocessname`	An invalid process name was specified at startup.
`incorrectprocessname`	A process name specified in a startup parameter or Set request does not describe a connection of the type supported by the parameter or MIB object.
`resourceproblem`	The connection is unavailable because of a process failure or interprocess communications failure.

When a connection has a state of `disconnected` and a state reason of `operatorrequest`, the subagent does not attempt to re-establish the connection when the keep-alive interval has elapsed, because the disconnection was deliberate. The subagent attempts to re-establish only connections inadvertently lost.

Resource Utilization

Several subagent attributes affect its CPU time and memory utilization:

CPU time utilization for request processing cannot be tuned. However, you can influence subagent CPU time usage in several ways:

- The lower the value of the -k startup parameter and its corresponding MIB object, tdmNwSaKeepAliveTimer, the greater the amount of CPU time utilized. This value, called the keep-alive interval, is the rate at which the subagent attempts to reconnect to IPXPROTO, SNMP agent, or EMS collector processes when a connection to one of them fails.
- The lower the value of the -r startup parameter and its corresponding MIB object, tdmNwSaRefreshTimer, the greater the amount of CPU time utilized. This value, called the refresh rate, is the frequency at which the subagent refreshes data describing IPXPROTO processes. Updating the tdmNwServerTable, which can be quite large, is the principal factor in CPU time utilization.
- Setting the value of tdmNwSaRefreshNow to `forceDynamicRefresh` increases CPU time utilization while values are updated.
- Running the subagent with a backup process increases CPU time utilization slightly.

Memory utilization cannot be tuned. The amount of memory utilized varies proportionally with the number of IPXPROTO and SNMP agent connections the subagent is managing. Running the subagent with a backup process increases memory usage in the backup CPU.

Subagent Backup Process

The IPX/SPX Subagent can run as a process pair. At startup, you can use the -b startup parameter to start a backup process in the CPU you specify. You can also use the tdmNwSaBackupCPU object to enable or disable the existence of a backup process. If the primary subagent process fails, the backup process takes over.

Several MIB objects are related to the backup process:

- The tdmNwSaBackupCPU value can be assigned a value of -1 to disable a backup process or a value of 0 through 15 to change the CPU in which the backup process runs.
- The tdmNwSaBackupPID value is the process ID of the backup process.
- The tdmNwSaBackupState value determines whether the backup process is connected to or disconnected from the subagent process.
- The tdmNwSaBackupStateReason provides the reason for the value of tdmNwSaBackupState
- The tdmNwSaSwitchToBackupNow object can be assigned the value `forceBackupTakeover` to cause the subagent backup process to take over and a new backup process to be created.

First Failure Data Capture

The IPX/SPX Subagent is instrumented to send messages to the HP

Tandem Failure Data System (TFDS) when it detects internal inconsistencies.

Standards Compliance

The MIB used by the IPX/SPX Subagent supports one of the five groups defined by Novell, the ipxSystem group. All the objects in the ipxSystem group are supported, but only for read-only access. More information on compliance with Novell’s MIB definitions appears in Novell MIB Objects on page 13-18.

All other definitions in the IPX/SPX Subagent’s MIB are defined by HP and fully comply with RFC 1155, RFC 1212, and RFC 1213.

Installation

This subsection describes how to install the IPX/SPX Subagent.

Dependencies

The following products must be configured and running for the subagent to operate properly:

- SNMP agent (D22 or later)
- EMS (D21 or later)
- A G02 or later version of the NonStop IPX/SPX product and the underlying SLSA resources upon which it relies.
- An SNMP manager that can send requests to a SNMP agent process

Installation Steps

The subagent is automatically installed when you install NonStop IPX/SPX.

Ensure that the EMS file named ZISATMPL is installed in the system template file. When you use the SYSGEN utility to generate your system, this file is automatically installed in the system template file; otherwise, you must manually install it. Refer to the *DSM Template Services Manual* for more information.

Load the subagent’s ASN.1 MIB definition files for any manager from which subagent requests will be sent. The MIB definitions are contained in three files:

- Two files contain Novell MIB definitions. If Novell definitions are already represented in a manager’s MIB view, do not install either of the files. Otherwise, install IPXMIB on managers that will be communicating with SNMP agents provided by various vendors or IPXMIBA on managers that will be communicating only with SNMP agents.
    - **IPXMIB** contains definitions identical to those written by Novell.
    - **IPXMIBA** contains the same definitions as IPXMIB, with several exceptions. DESCRIPTION clauses have been modified to provide information specific to HP systems. In addition, ipxBasicSysExistState is a read-only object, not a read-write object, and ipxBasicSysName has a syntax of DisplayString, not OCTET STRING.
- **TDMNWMIB** defines the objects defined by HP.

Configure the SNMP agent to support communications with any managers that you want to interact with the IPX/SPX Subagent, as described in Section 2, Installing and Configuring the SNMP Agent.

The subagent can communicate with multiple SNMP agent processes and multiple IPXPROTO processes running on the same or remote nodes. The subagent can send event messages to an EMS collector process on the same or a remote node. More information about subagent configuration appears in the following subsection.

Configuration

You can configure the behavior of the subagent at startup or by setting the value of MIB objects in the zisa group from an SNMP manager after startup. Table 13-1 summarizes the available options.

Table 13-1. IPX/SPX Subagent Configuration Summary

Subagent Attribute	Startup Parameter	MIB Object	Default Value
SNMP agent connections	-a	tdmNwSaAgentName tdmNwSaAgentState tdmNwSaAgentStateReason	$ZSNMP on subagent’s node
IPXPROTO connections	-i	tdmNwSaIpxProtoName tdmNwSaIpxProtoState tdmNwSaIpxProtoStateReason	$ZNV0 on subagent’s node
EMS collector	-c	tdmNwSaEmsCollectorName tdmNwSaEmsCollectorState tdmNwSaEmsCollectorStateReaso n	$0 on subagent’s node
Subagent backup process	-b	tdmNwSaBackupCPU tdmNwSaBackupState tdmNwSaSwitchToBackupNow	No backup process
Keep-alive interval	-k	tdmNwSaKeepAliveTimer	Every 60 minutes
Refresh interval	-r	tdmNwSaRefreshTimer tdmNwSaRefreshNow	Every 10 minutes
Subagent process priority		tdmNwSaProcessPriority	Assigned by NonStop Kernel operating system

All subagent attributes that can be controlled by setting MIB object values are reinitialized to default values when the subagent is restarted, or to values specified in startup parameters. Refer to Starting and Stopping the Subagent for information on configuration using startup parameters. Refer to subsequent subsections for information on settable MIB objects.

Starting and Stopping the Subagent

The IPXPROTO processes and SNMP agents with which you want the subagent to communicate must be running before the subagent is started. If an agent is not running, start it with the RUN command:

```
RUN SNMPAGT /NAME $agent-process, NOWAIT/
```

Start the IPX/SPX Subagent by entering the RUN command at a terminal that is running the TACL program. The subagent can be run by any user, but must be started as a named process.

```
[RUN] [[$volume.]subvolume.]IPXSA
    [/[NAME $subagent-process] [,other-run-option].../]
    [-a $agent-process [$agent-process...]]
    [-b backup-cpu]
    [-c $ems-collector-process]
    [-i $ipxproto-process [$ipxproto-process...]]
    [-k keep-alive-interval]
    [-r refresh-interval]
```

volume

identifies the volume on which IPXSA resides. You can omit it if IPXSA resides on your current volume. By default, the installation program puts IPXSA into <*dsv*>.ZIPXPRO.

subvolume

identifies the subvolume on which IPXSA resides. You can omit it if it is named in your TACL #PMSEARCHLIST. By default, the installation program puts IPXSA into <*dsv*>.ZIPXPRO.

subagent-process

identifies the IPX/SPX Subagent process. You can specify from one through five alphanumeric characters, but the first character must be alphabetic. HP suggests that you name the subagent process $ZISA.

If you omit the NAME option, the subagent cannot start a backup process.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the following options:

- the NOWAIT option so that you can resume TACL operations once the subagent is started
- the -b option so that you will know where the backup process will run

`agent-process`

identifies a local or remote SNMP agent process from which you want the subagent to receive manager requests. You can specify from one through five alphanumeric characters, but the first character must be alphabetic. If you omit the -a startup parameter, $ZSNMP on the subagent’s node is assumed. If you specify more than one agent, separate agent process names with a space. Only one subagent process can communicate with a particular SNMP agent process at a time.

If the subagent cannot successfully establish an agent connection, it attempts to establish one at the rate specified by the -k startup parameter.

After startup, you can use two MIB objects to control agent process connections. To add a new agent connection, set the value of tdmNwSaAgentName to identify a SNMP agent process. To activate or deactivate an agent connection, set the value of tdmNwSaAgentState to `connected` or `disconnected`, respectively.

`backup-cpu`

identifies a CPU number in which the subagent’s backup process should run. Valid values are -1 or 0 through 15. Specifying -1 is equivalent to omitting the -b startup parameter; no backup process is started. The integers 0 through 15 identify CPUs. If the specified CPU is not available, the subagent attempts to start the backup process at the rate specified by the -k startup parameter. If you do not include the NAME option to name the subagent process, the subagent cannot start a backup process.

After startup, you can use three MIB objects to control the subagent backup process. To create a backup process or change the CPU in which an existing one is running, set the value of tdmNwSaBackupCPU to identify a CPU. To activate or deactivate a backup process, set the value of tdmNwSaBackupState to `connected` or `disconnected`, respectively. To cause the backup process to become the primary process, set the value of tdmNwSaSwitchToBackupNow to `forceBackupTakeover`.

`ems-collector-process`

identifies a local or remote EMS collector process to which the subagent should send EMS events. You can specify from one through five alphanumeric characters, but the first character must be alphabetic. If you omit the -c startup parameter, event messages are sent to $0 on the subagent’s node.

If the subagent cannot establish a connection with the EMS collector process, it generates no events. The subagent attempts to establish a connection at the rate specified by the -k startup parameter.

After startup, you can use two MIB objects to control the EMS collector process. To change the EMS collector process, set the value of tdmNwSaEmsCollectorName to identify the EMS collector process. To activate or deactivate event collection, set the value of tdmNwSaEmsCollectorState to `connected` or `disconnected`, respectively.

`ipxproto-process`

identifies a local or remote IPXPROTO process you want the subagent to monitor. You can specify from one through five alphanumeric characters, but the first character must be alphabetic. If you specify more than one IPXPROTO process, separate process names with a space. If you omit the -i startup parameter, $ZNV0 on the subagent's node is assumed.

If the subagent cannot successfully establish a connection with an IPXPROTO process, it attempts to establish one at the rate specified by the -k startup parameter.

After startup, you can use two MIB objects to control IPXPROTO process connections. To add a new connection, set the value of tdmNwSaProtoName to identify an IPXPROTO process. To activate or deactivate an IPXPROTO process connection, set the value of tdmNwSaIpxProtoState to `connected` or `disconnected`, respectively.

`keep-alive-interval`

specifies how often you want the subagent to try to establish connections with agent, IPXPROTO, EMS collector, or backup processes with which connections have failed. The keep-alive interval is the number of minutes between attempts. Valid values range from 0 through 32000. A value of 0 suppresses reconnections when disconnections occur. If you omit the -k startup parameter, the subagent tries to establish lost connections every 60 minutes.

After startup, you can change the keep-alive interval by setting the value of tdmNwSaKeepAliveTimer to a different interval.

`refresh-interval`

specifies how often you want the subagent to update values for objects in the six tables that describe IPXPROTO processes:

ipxBasicSysTable
tdmNwInfoTable
tdmNwIPXTable
tdmNwSPXTable
tdmNwNCPTable
tdmNwServerTable

When the subagent first accesses a table after startup, the table's values are updated and its refresh timer starts running. The next time the subagent accesses the table, its values are updated only if its refresh timer has run down. The refresh timer runs down when the refresh interval, specified in minutes, has elapsed since the previous table access. Valid values range from 0 through 32000. A value of zero or less causes the subagent to update values every time it receives a request. If you omit the -r startup parameter, the six refresh timers are set to 10 minutes.

After startup, you can use two MIB objects to control MIB value updates. To change the refresh interval, set tdmNwSaRefreshTimer to a new interval. To cause values to be updated on demand, set tdmNwSaRefreshNow to `forceDynamicRefresh`.

You can retrieve the value of tdmNwInfoResetTime to determine the time at which the statistical counters for an IPXPROTO process were last reset to zero; resetting counters is done by using the SCF STATS PROCESS command with the RESET option. You can retrieve the value of tdmNwInfoSampleTime to determine the time at which the statistics were last sampled by the subagent.

To stop the subagent process, provide its name in the TACL STOP command:

```
STOP $ZISA
```

Troubleshooting the Subagent

This subsection contains several suggestions for handling problems you might encounter while using the IPX/SPX Subagent.

EMS Event Messages

Whenever the subagent behaves unexpectedly or an error condition arises, be sure to check for EMS events that have been generated by the subagent. Described in EMS Support on page 13-45, event messages provide information to help you diagnose and fix problems.

Startup Problems

If the syntax of the RUN command is incorrect, the subagent displays a message, then stops. The message has the following form:

```
problem with command line parameter #parameter-number ('parameter-value'):
expected parameter-description
IPXSA - T8170D30_4MAR96_IPXSA_02APR96 - (c) Compaq Computers Incorporated
1996

Usage:
[ -a $<agent-name>     [ ... ] ]  (default: $ZSNMP)
[ -b <backup-cpu-number>       ]  (default: -1 (no backup))  (range: [-
1,0..15])
[ -c $<ems-collector-name>     ]  (default: $0)
[ -i $<ipxproto-name>  [ ... ] ]  (default: $ZNV0)
[ -k <keep-alive-rate>        ]  (default: 60 minutes)  (range: [0..32000])
[ -r <refresh-cache-rate>      ]  (default: 10 minutes)  (range: [0..32000])

Example:
  tacl> run ipxsa /name $ipxsa, nowait/ -a $zsnmp -i $znv0
```

`parameter-number`

identifies which startup parameter in the RUN command is incorrect.

`parameter-value`

identifies the incorrect information detected.

`parameter-description`

describes the information that would be valid for a particular startup parameter. For example, the parameter description for an invalid -r value would be: a `refresh rate [0..32000]`.

If you receive one of these messages, retry the RUN command using the correct startup syntax.

If the subagent stops running shortly after startup, check the EMS event log for unusual conditions. If no EMS events explain the startup difficulty, investigate subagent-agent communications problems.

Any SNMP agent specified at startup needs to be running before you start the subagent. If the agent is not running, try to determine why. Common agent startup problems involve security and port conflicts. Section 2, Installing and Configuring the SNMP Agent, provides complete information on agent startup options.

Another common cause for startup problems is that a SNMP agent process specified at startup is already in use by another IPX/SPX Subagent process. Only one IPX/SPX Subagent can communicate with a particular agent at any time.

Timeouts

You may experience these kinds of timeouts:

- The subagent receives no response from an IPXPROTO process within 10 seconds.
- The time it takes to refresh MIB object values exceeds a manager's timeout value.

IPXPROTO Timeouts

When the subagent retrieves values from an IPXPROTO process, it waits 10 seconds for a reply from the process. If it receives no reply, the subagent generates a NoSuchName error in response to the request that fostered IPXPROTO process access. The subagent also generates an object-unavailable EMS event and sets the values of two objects in the tdmNwSaIPXPROTONames Table:

- The value of tdmNwSaIpxProtoState is set to `disconnected`.
- The value of tdmNwSaIpxProtoStateReason is set to `resourceproblem`.

The timeout value is not configurable.

Manager Timeouts

The greater the number of MIB values maintained, the longer it takes the subagent to refresh the values. You can minimize data refresh wait time by providing a long refresh interval in the -r startup parameter at startup, then forcing value updates when required after startup by setting the value of tdmNwSaRefreshNow to `forceDyamicRefresh`. You can also configure some managers so that their timeouts and retries accommodate longer response times; setting the timeout value to 5 to 9 seconds and the retry value to 0 or 1 may be sufficient.

Novell MIB Objects

The IPX/SPX Subagent supports the objects in one of the five groups defined by Novell: the ipxSystem group. This group contains a table whose entries each represent a single instance of the IPX/SPX protocol stack. The IPX/SPX Subagent supports the objects identified by a check mark in the following list, which shows all the groups in Novell’s MIB:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            novell (23)
              mibDoc (2)
                ipx (5)
                  ipxSystem (1) √
                      ipxBasicSysTable (1)  √
                        ipxBasicSysEntry (1) √
                          ipxBasicSysInstance (1) √
                          ipxBasicSysExistState (2) √
                          ipxBasicSysNetNumber (3) √
                          ipxBasicSysNode (4) √
                          IpxBasicSysName (5) √
                          ipxBasicSysInReceives (6) √
                          ipxBasicSysInHdrErrors (7) √
                          ipxBasicSysInUnknownSockets (8) √
                          ipxBasicSysInDiscards (9) √
                          ipxBasicSysInBadChecksums (10) √
                          ipxBasicSysInDelivers (11) √
                          ipxBasicSysNoRoutes (12) √
                          ipxBasicSysOutRequests (13) √
                          ipxBasicSysOutMalformedRequests (14) √
                          ipxBasicSysOutDiscards (15) √
                          ipxBasicSysOutPackets (16) √
                          ipxBasicSysConfigSockets (17) √
```

ipxBasicSysOpenSocketFails (18) √
ipxCircuit (2)
ipxForwarding (3)
ipxServices (4)
ipxTraps (5)

MIB Objects

Table 13-2 describes how the IPX/SPX Subagent supports objects in entries of the ipxBasicSysTable of the ipxSystem group.

Table 13-2. ipxBasicSysEntry Objects Supported by the Subagent (page 1 of 3)

Object and Attributes	**Definition**	**Derivation of Value**
ipxBasicSysInstance 1.3.6.1.4.1.23.2.5.1.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. The value serves as an index to entries in the ipxBasicSysTable. This value corresponds to the value of tdmNwSaIpxProtoBasicSys-Instance in the tdmNwSa-IPXPROTONamesTable.	Assigned by subagent
ipxBasicSysExistState 1.3.6.1.4.1.23.2.5.1.1.1.2 read-only INTEGER {off(1), on(2)}	The value 0 is always returned.	Calculated by IPXPROTO process
ipxBasicSysNetNumber 1.3.6.1.4.1.23.2.5.1.1.1.3 read-only OCTET STRING (SIZE(4))	The network number portion of the IPX address of this system.	Calculated by IPXPROTO process
ipxBasicSysNode 1.3.6.1.4.1.23.2.5.1.1.1.4 read-only OCTET STRING (SIZE(6))	The node number portion of the IPX address of this system.	Calculated by IPXPROTO process
ipxBasicSysName 1.3.6.1.4.1.23.2.5.1.1.1.5 read-only DisplayString (SIZE (0..48))	A string of length 0 is always returned.	Calculated by IPXPROTO process
ipxBasicSysInReceives 1.3.6.1.4.1.23.2.5.1.1.1.6 read-only Counter	The total number of IPX packets received, including those received in error.	Calculated by IPXPROTO process

Table 13-2. ipxBasicSysEntry Objects Supported by the Subagent (page 2 of 3)

Object and Attributes	**Definition**	**Derivation of Value**
ipxBasicSysInHdrErrors 1.3.6.1.4.1.23.2.5.1.1.1.7 read-only Counter	The number of IPX packets discarded because of errors in their headers, including any IPX packets with a size smaller than the minimum of 30 bytes.	Calculated by IPXPROTO process
ipxBasicSysInUnknownSockets 1.3.6.1.4.1.23.2.5.1.1.1.8 read-only Counter	The number of IPX packets discarded because the destination socket was not open.	Calculated by IPXPROTO process
ipxBasicSysInDiscards 1.3.6.1.4.1.23.2.5.1.1.1.9 read-only Counter	The number of IPX packets received but discarded for reasons other than those accounted for by ipxBasicSysInHdrErrors or ipxBasicSysInUnknownSockets.	Calculated by IPXPROTO process
ipxBasicSysInBadChecksums 1.3.6.1.4.1.23.2.5.1.1.1.10 read-only Counter	The number of IPX packets received with incorrect checksums.	Calculated by IPXPROTO process
ipxBasicSysInDelivers 1.3.6.1.4.1.23.2.5.1.1.1.11 read-only Counter	The total number of IPX packets delivered locally, including packets from local applications.	Calculated by IPXPROTO process
ipxBasicSysNoRoutes 1.3.6.1.4.1.23.2.5.1.1.1.12 read-only Counter	The number of times no route to a destination was found.	Calculated by IPXPROTO process
ipxBasicSysOutRequests 1.3.6.1.4.1.23.2.5.1.1.1.13 read-only Counter	The number of IPX packets supplied locally for transmission.	Calculated by IPXPROTO process
ipxBasicSysOutMalformedRequests 1.3.6.1.4.1.23.2.5.1.1.1.14 read-only Counter	The number of IPX packets supplied locally that contained errors in their structure.	Calculated by IPXPROTO process
ipxBasicSysOutDiscards 1.3.6.1.4.1.23.2.5.1.1.1.15 read-only Counter	The number of outgoing IPX packets discarded for reasons other than those accounted for in ipxBasicSysOutMalformed-Requests.	Calculated by IPXPROTO process

Table 13-2. ipxBasicSysEntry Objects Supported by the Subagent (page 3 of 3)

Object and Attributes	Definition	Derivation of Value
ipxBasicSysOutPackets 1.3.6.1.4.1.23.2.5.1.1.1.16 read-only Counter	The total number of IPX packets transmitted.	Calculated by IPXPROTO process
ipxBasicSysConfigSockets 1.3.6.1.4.1.23.2.5.1.1.1.17 read-only INTEGER	The configured maximum number of IPX sockets that can be open at one time.	Calculated by IPXPROTO process
ipxBasicSysOpenSocketFails 1.3.6.1.4.1.23.2.5.1.1.1.18 read-only Counter	The number of IPX socket open calls that failed.	Calculated by IPXPROTO process

ipxBasicSysTable Maintenance

The subagent creates an entry in the ipxBasicSysTable whenever it creates an entry in the tdmNwSaIPXPROTONAMESTable. Entries cannot be deleted; to deactivate an entry, set the value of tdmNwSaIpxProtoState to `disconnected`. Values in ipxBasicSysTable entries are updated when the ipxBasicSysTable refresh timer has run down; the refresh timer minutes are specified by using the -r startup parameter or setting the value of tdmNwSaRefreshTimer. Values are also updated when the value of tdmNwSaRefreshNow is set to `forceDynamicRefresh`.

Compliance With Novell MIB

Table 13-3 summarizes compliance of ipxBasicSys group support with Novell's definitions.

Table 13-3. Compliance With Novell ipxBasicSys Group Definitions (page 1 of 2)

Object Descriptor	Compliance	Explanation
ipxBasicSysInstance	Yes	See Table 13-2.
ipxBasicSysExistState	Partial	This object is read-only, not read-write.
ipxBasicSysNetNumber	Yes	See Table 13-2.
ipxBasicSysNode	Yes	See Table 13-2.
ipxBasicSysName	Yes	This object is a DisplayString, not an OCTET STRING.
ipxBasicSysInReceives	Yes	See Table 13-2.
ipxBasicSysInHdrErrors	Yes	See Table 13-2.
ipxBasicSysInUnknownSockets	Yes	See Table 13-2.

Table 13-3. Compliance With Novell ipxBasicSys Group Definitions (page 2 of 2)

Object Descriptor	Compliance	Explanation
ipxBasicSysInDiscards	Partial	The values of ipxAdvSysInDiscards and ipxAdvSysInCompressDiscards are not taken into account.
ipxBasicSysInBadChecksums	Yes	See Table 13-2.
ipxBasicSysInDelivers	Yes	See Table 13-2.
ipxBasicSysNoRoutes	Yes	See Table 13-2.
ipxBasicSysOutRequests	Partial	The value of ipxAdvForwPackets is not taken into account.
ipxBasicSysOutMalformedRequests		See Table 13-2.
ipxBasicSysOutDiscards	Partial	The values of ipxAdvSysOutFiltered and ipxAdvSysOutCompressDiscards are not taken into account.
ipxBasicSysOutPackets	Yes	See Table 13-2.
ipxBasicSysConfigSockets	Yes	See Table 13-2.
ipxBasicSysOpenSocketFails	Yes	See Table 13-2.

HP MIB Objects

The IPX/SPX Subagent's MIB contains two groups defined by HP that characterize IPX/SPX resources on NonStop Kernel systems: the zipx group and the zisa group:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  tandem (169)
                     nonstopsystems (3)
                        zipx (164)
                        zisa (196)
```

zipx Group

The zipx group, which contains objects providing information about IPXPROTO processes, is named after the subsystem abbreviation for NonStop IPX/SPX (ZIPX). The zipx group objects are organized into five tables:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zipx (164)
                  tdmNwInfo (1)
                    tdmNwInfoTable (1)
                  tdmNwIPX (2)
                    tdmNwIPXTable (1)
                  tdmNwSPX (3)
                    tdmNwSPXTable (1)
                  tdmNwNCP (4)
                    tdmNwNCPTable (1)
                  tdmNwServer (5)
                    tdmNwServerTable (1)
```

MIB Objects

This subsection contains five tables, Table 13-4 through Table 13-8, which describe the MIB objects in each of the five zipx tables. The zipx tables are indexed by instance identifiers that have corresponding values in the ipxBasicSysTable and the ipxNwSaIPXPROTONamesTable. Instance identifiers are assigned by the subagent when new entries are added to the ipxNwSaIPXPROTONamesTable.

The objects in the zipx table entries provide information equivalent to that retrieved when using SCF commands for an IPXPROTO process. Table 13-4 through Table 13-8 identify the SCF commands that provide equivalent information for each object.

tdmNwInfoTable Entries

Table 13-4 describes tdmNwInfoTable entries. This table contains an entry for each IPXPROTO process the subagent is configured to monitor:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zipx (164)
                  tdmNwInfo (1)
                    tdmNwInfoTable (1)
                      tdmNwInfoEntry (1)
                        tdmNwInfoEntryInstanceID (1)
                        tdmNwInfoProgramFileName (2)
                        tdmNwInfoQIOLibFileName (3)
                        tdmNwInfoPrimaryCPU (4)
                        tdmNwInfoBackupCPU (5)
                        tdmNwInfoIOPortName (6)
                        tdmNwInfoIOPortType (7)
                        tdmNwInfoIOPortAddress (8)
                        tdmNwInfoQIOLimit (9)
                        tdmNwInfoQIOUsage (10)
                        tdmNwInfoResetTime (11)
                        tdmNwInfoSampleTime (12)
                        tdmNwInfoNumberOfOpens (13)
                        tdmNwInfoVersion (14)
```

Table 13-4. tdmNwInfoEntry Objects Supported by the Subagent (page 1 of 3)

Object and Attributes	Definition	Description of Value
tdmNwInfoEntryInstanceID 1.3.6.1.4.1.169.3.164.1.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. This value, which serves as the index to entries in the tdmNwInfoTable, corresponds to the value of tdmNwSaIpxProto-BasicSysInstance in the tdmNwSaIPXPROTO-NamesTable.	Assigned by subagent.

Table 13-4. tdmNwInfoEntry Objects Supported by the Subagent (page 2 of 3)

Object and Attributes	Definition	Description of Value
tdmNwInfoProgramFileName 1.3.6.1.4.1.169.3.164.1.1.1.2 read-only DisplayString (SIZE (0..50))	The name of the IPXPROTO process program name.	`\`*`system`*`.$`*`volume`*`.` *`subvolume`*`.`*`file-name`* obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoQIOLibFileName 1.3.6.1.4.1.169.3.164.1.1.1.3 read-only DisplayString (SIZE (0..50))	The name of the QIO library file.	`\`*`system`*`.$`*`volume`*`.` *`subvolume`*`.`*`file-name`* obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoPrimaryCPU 1.3.6.1.4.1.169.3.164.1.1.1.4 read-only INTEGER (0..15)	The CPU in which the primary IPXPROTO process is running.	Integer ranging from 0 through 15 obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoBackupCPU 1.3.6.1.4.1.169.3.164.1.1.1.5 read-only INTEGER (-1..15)	The CPU in which the backup IPXPROTO process is running.	Integers of 0 through 15 identify CPUs; -1 indicates no backup CPU exists. Value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoIOPortName 1.3.6.1.4.1.169.3.164.1.1.1.6 read-only DisplayString (SIZE (0..50))	The name of the TLAM port or the SLSA LIF that provides LAN access for the IPXPROTO process.	`$`*`line`*`.#`*`port`* for TLAM or *`LIF_name`*.*`protocol`* for SLSA obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoIOPortType 1.3.6.1.4.1.169.3.164.1.1.1.7 read-only DisplayString (SIZE (0..25))	The type of Ethernet frame TLAM or SLSA is using.	Possible values are Ethernet, SNAP, 802.2, 802.3, and `unknown` (##). Value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoIOPortAddress 1.3.6.1.4.1.169.3.164.1.1.1.8 read-only DisplayString (SIZE (0..25))	The address of the TLAM port or SLSA filter.	Two 16-bit words, expressed as a hexadecimal string: `0`*`xnnnn`* `0`*`xnnnn`*. Value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.

Table 13-4. tdmNwInfoEntry Objects Supported by the Subagent (page 3 of 3)

Object and Attributes	Definition	Description of Value
tdmNwInfoQIOLimit 1.3.6.1.4.1.169.3.164.1.1.1.9 read-only INTEGER	The maximum percentage of QIO memory to be used by the IPXPROTO process.	Integer obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwInfoQIOUsage 1.3.6.1.4.1.169.3.164.1.1.1.10 read-only Gauge (0..100)	The percentage of total QIO memory currently in use by the IPXPROTO process.	Integer obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwInfoResetTime 1.3.6.1.4.1.169.3.164.1.1.1.11 read-only DisplayString (SIZE (0..25))	The time at which the statistical counters for the IPXPROTO process were last initialized (set to 0) using the SCF STATS PROCESS command with the RESET option.	`YYYY/MM/DD hh:mm:ss` obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwInfoSampleTime 1.3.6.1.4.1.169.3.164.1.1.1.12 read-only DisplayString (SIZE (0..25))	The time at which the statistics were last sampled by the subagent.	`YYYY/MM/DD hh:mm:ss`.
tdmNwInfoNumberOfOpens 1.3.6.1.4.1.169.3.164.1.1.1.13 read-only Counter	The current number of openers of the IPXPROTO process.	Integer obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwInfoVersion 1.3.6.1.4.1.169.3.164.1.1.1.14 read-only DisplayString (SIZE (0..50))	The version of the IPXPROTO process.	String containing the version number, product name, product number, and release date, for example: `IPXPROTO (T8678D30^31OCT94^D30_V09_BASE).` Value obtained using a mechanism equivalent to the SCF VERSION PROCESS command.

tdmNwIPXTable Entries

Table 13-5 describes tdmNwIPXTable entries. Each entry in the table contains objects that provide statistical information about the IPX layer of an IPXPROTO process the subagent is configured to monitor:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zipx (164)
                  tdmNwIPX (2)√
                    tdmNwIPXTable (1)
                      tdmNwIPXEntry (1)
                        tdmNwIPXEntryInstanceId (1)
                        tdmNwIPXInboundPacketCnt (2)
                        tdmNwIPXOutboundPacketCnt (3)
                        tdmNwIPXMaxSockets (4)
                        tdmNwIPXOpenSocketsCnt (5)
                        tdmNwIPXOpenSocketFailureCnt (6)
                        tdmNwIPXFindRouteFailureCnt (7)
                        tdmNwIPXDroppedCnt (8)
                        tdmNwIPXTotalErrors (9)
```

Table 13-5. tdmNwIPXEntry Objects Supported by the Subagent (page 1 of 2)

Object and Attributes	Definition	Description of Value
tdmNwIPXEntryInstanceId 1.3.6.1.4.1.169.3.164.2.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. This value, which serves as the index to entries in the tdmNwIPXTable, corresponds to the value of tdmNwSaIpxProto-BasicSysInstance in the tdmNwSaIPX-PROTONamesTable.	Assigned by subagent.
tdmNwIPXInboundPacketCnt 1.3.6.1.4.1.169.3.164.2.1.1.2 read-only Counter	The total number of inbound packets received by the IPX layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.

Table 13-5. tdmNwIPXEntry Objects Supported by the Subagent (page 2 of 2)

Object and Attributes	Definition	Description of Value
tdmNwIPXOutboundPacketCnt 1.3.6.1.4.1.169.3.164.2.1.1.3 read-only Counter	The total number of outbound packets sent from the IPX layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwIPXMaxSockets 1.3.6.1.4.1.169.3.164.2.1.1.4 read-only Counter	The maximum number of IPX sockets that can be open simultaneously.	Integer value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwIPXOpenSocketsCnt 1.3.6.1.4.1.169.3.164.2.1.1.5 read-only Counter	The current number of open sockets for IPX.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwIPXOpenSocketFailureCnt 1.3.6.1.4.1.169.3.164.2.1.1.6 read-only Counter	The number of times an IPX socket was unavailable.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwIPXFindRouteFailureCnt 1.3.6.1.4.1.169.3.164.2.1.1.7 read-only Counter	The number of times the IPXPROTO process was unable to find a route to the network.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwIPXDroppedCnt 1.3.6.1.4.1.169.3.164.2.1.1.8 read-only Counter	The number of packets dropped by the QIO interface of the IPXPROTO process.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwIPXTotalErrors 1.3.6.1.4.1.169.3.164.2.1.1.9 read-only Counter	The sum of the values of tdmNwIPXOpen-SocketFailureCnt, tdmNwIPXFind-RouteFailureCnt, and tdmNwIPX-DroppedCnt.	Computed by the IPXPROTO process.

tdmNwSPXTable Entries

Table 13-6 describes tdmNwSPXTable entries. Each entry in the table contains objects that provide statistical information about the SPX layer of an IPXPROTO process the subagent is configured to monitor:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  tandem (169)
                     nonstopsystems (3)
                        zipx (164)
                           tdmNwSPX (3)√
                              tdmNwSPXTable (1)
                                 tdmNwSPXEntry (1)
                                    tdmNwSPXEntryInstanceId (1)
                                    tdmNwSPXMaxBinds (2)
                                    tdmNwSPXMaxConn (3)
                                    tdmNwSPXMaxResend (4)
                                    tdmNwSPXKeepAliveTimer (5)
                                    tdmNwSPXInboundPacketCnt (6)
                                    tdmNwSPXOutboundPacketCnt (7)
                                    tdmNwSPXOpenSocketsCnt (8)
                                    tdmNwSPXSendFailureCnt (9)
                                    tdmNwSPXBadIncomingPacketCnt (10)
                                    tdmNwSPXSuppressedPacketCnt (11)
                                    tdmNwSPXTotalErrors (12)
```

Table 13-6. tdmNwSPXEntry Objects Supported by the Subagent (page 1 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSPXEntryInstanceId 1.3.6.1.4.1.169.3.164.3.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. This value, which serves as the index to entries in the tdmNwSPXTable, corresponds to the value of tdmNwSaIpxProto-BasicSysInstance in the tdmNwSaIPX-PROTONamesTable .	Assigned by subagent.

Table 13-6. tdmNwSPXEntry Objects Supported by the Subagent (page 2 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSPXMaxBinds 1.3.6.1.4.1.169.3.164.3.1.1.2 read-only INTEGER	The maximum number of concurrent explicit binds to the IPXPROTO process.	Integer value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwSPXMaxConn 1.3.6.1.4.1.169.3.164.3.1.1.3 read-only INTEGER	The maximum number of simultaneously active SPX connections.	Integer value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwSPXMaxResend 1.3.6.1.4.1.169.3.164.3.1.1.4 read-only INTEGER	The maximum number of times that SPX will resend.	Integer value obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwSPXKeepAliveTimer 1.3.6.1.4.1.169.3.164.3.1.1.5 read-only INTEGER	The number of minutes between pings. SPX connections are regularly pinged to verify the existence of the remote SPX endpoint.	Integer value between 1 and 60 obtained using a mechanism equivalent to the SCF INFO PROCESS DETAIL command.
tdmNwSPXInboundPacketCnt 1.3.6.1.4.1.169.3.164.3.1.1.6 read-only Counter	The total number of inbound packets received by the SPX layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwSPXOutboundPacketCnt 1.3.6.1.4.1.169.3.164.3.1.1.7 read-only Counter	The total number of outbound packets sent from the SPX layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwSPXOpenSocketsCnt 1.3.6.1.4.1.169.3.164.3.1.1.8 read-only Counter	The current number of open sockets for SPX.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwSPXSendFailureCnt 1.3.6.1.4.1.169.3.164.3.1.1.9 read-only Counter	The number of times SPX has reached the maximum number of resends.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.

Table 13-6. tdmNwSPXEntry Objects Supported by the Subagent (page 3 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSPXBadIncomingPacketCnt 1.3.6.1.4.1.169.3.164.3.1.1.10 read-only Counter	The number of times SPX received a bad packet.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwSPXSuppressedPacketCnt 1.3.6.1.4.1.169.3.164.3.1.1.11 read-only Counter	The number of times a connection request has not reached the LAN.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwSPXTotalErrors 1.3.6.1.4.1.169.3.164.3.1.1.12 read-only Counter	The sum of the values of tdmNwSPXSend-FailureCnt, tdmNw-SPXBadIncoming-PacketCnt, and tdmNwSPX-SuppressedPacket-Cnt.	Computed by the IPXPROTO process.

tdmNwNCPTable Entries

Table 13-7 describes tdmNwNCPTable entries. Each entry in the table contains objects that provide statistical information about the NCP layer of an IPXPROTO process the subagent is configured to monitor:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zipx (164)
                  tdmNwNCP (4)
                    tdmNwNCPTable (1)
                      tdmNwNCPEntry (1)
                        tdmNwNCPEntryInstanceId (1)
                        tdmNwNCPInboundPacketCnt (2)
                        tdmNwNCPOutboundPacketCnt (3)
                        tdmNwNCPInboundPacketErrors (4)
                        tdmNwNCPOutboundPacketErrors (5)
                        tdmNwNCPOpenCnt (6)
```

tdmNwNCPMessageCnt (7)
tdmNwNCPTotalErrors (8)

Table 13-7. tdmNwNCPEntry Objects Supported by the Subagent (page 1 of 2)

Object and Attributes	**Definition**	**Description of Value**
tdmNwNCPEntryInstanceId 1.3.6.1.4.1.169.3.164.4.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. This value, which serves as the index to entries in the tdmNwNCPTable, corresponds to the value of tdmNwSaIpxProto-BasicSysInstance in the tdmNwSaIPX-PROTONamesTable.	Assigned by subagent.
tdmNwNCPInboundPacketCnt 1.3.6.1.4.1.169.3.164.4.1.1.2 read-only Counter	The total number of inbound packets received by the NCP layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwNCPOutboundPacketCnt 1.3.6.1.4.1.169.3.164.4.1.1.3 read-only Counter	The total number of outbound packets sent from the NCP layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwNCPInboundPacketErrors 1.3.6.1.4.1.169.3.164.4.1.1.4 read-only Counter	The total number of times the NCP layer was unable to get a packet from IPX.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwNCPOutboundPacketErrors 1.3.6.1.4.1.169.3.164.4.1.1.5 read-only Counter	The total number of times the NCP layer was unable to send a packet to IPX.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwNCPOpenCnt 1.3.6.1.4.1.169.3.164.4.1.1.6 read-only Counter	The current number of NCP opens.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.

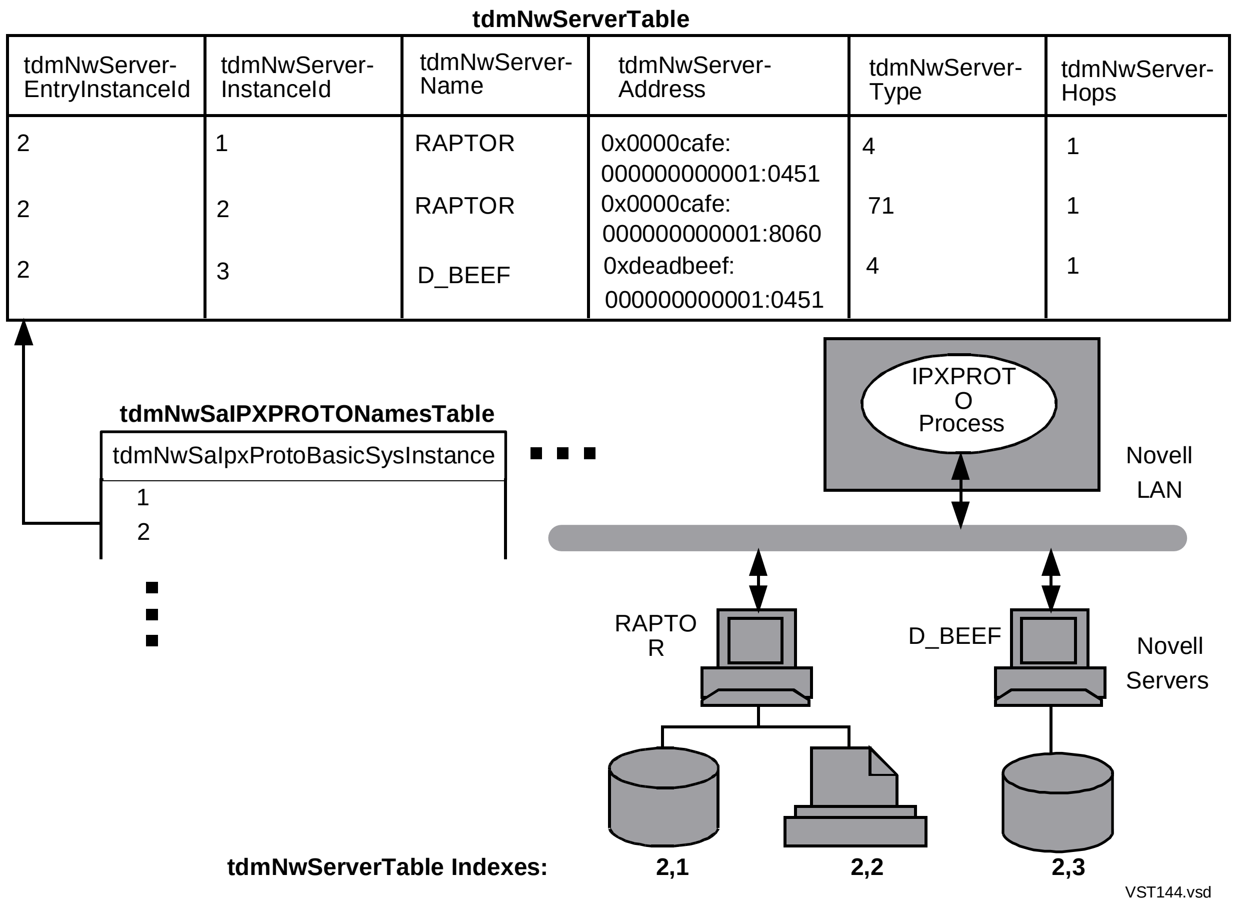
Table 13-7. tdmNwNCPEntry Objects Supported by the Subagent (page 2 of 2)

Object and Attributes	Definition	Description of Value
tdmNwNCPMessageCnt 1.3.6.1.4.1.169.3.164.4.1.1.7 read-only Counter	The number of times a user request was received by the NCP layer.	Integer value obtained using a mechanism equivalent to the SCF STATS PROCESS command.
tdmNwNCPTotalErrors 1.3.6.1.4.1.169.3.164.4.1.1.8 read-only Counter	The sum of the values of tdmNCP-InboundPacket-Errors and tdmNwNCP-OutboundPacket-Errors.	Computed by the IPXPROTO process.

tdmNwServerTable Entries

Each tdmNwServerTable entry corresponds to a path to a Novell server that can be reached from an IPXPROTO process the subagent is configured to monitor. Because an IPXPROTO process can be linked to more than one server, two values are used to index entries in the tdmNwServerTable: an IPXPROTO process identifier (tdmNwServerEntryInstanceId) and a server identifier (tdmNwServerInstanceId), as Figure 13-4 illustrates. The value of the server identifier is derived using the values of the server name, address, and type objects.

Figure 13-4. Derivation of tdmNwServerTable Index Values



tdmNwServerTable

tdmNwServer-EntryInstanceId	tdmNwServer-InstanceId	tdmNwServer-Name	tdmNwServer-Address	tdmNwServer-Type	tdmNwServer-Hops
2	1	RAPTOR	0x0000cafe: 000000000001:0451	4	1
2	2	RAPTOR	0x0000cafe: 000000000001:8060	71	1
2	3	D_BEEF	0xdeadbeef: 000000000001:0451	4	1

tdmNwSaIPXPROTONamesTable
tdmNwSaIpxProtoBasicSysInstance
1
2
IPXPROTO Process
Novell LAN
RAPTOR
D_BEEF
Novell Servers
tdmNwServerTable Indexes:
2,1
2,2
2,3
VST144.vsd

Table 13-8 describes tdmNwServerTable entries:

iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zipx (164)
                  tdmNwServer (5)√
                    tdmNwServerTable (1)
                      tdmNwServerEntry (1)
                        tdmNwServerEntryInstanceId (1)
                        tdmNwServerInstanceId (2)
                        tdmNwServerName (3)
                        tdmNwServerAddress (4)

tdmNwServerType (5)
tdmNwServerHops (6)

Table 13-8. tdmNwServerEntry Objects Supported by the Subagent (page 1 of 2)

Object and Attributes	**Definition**	**Description of Value**
tdmNwServerEntryInstanceId 1.3.6.1.4.1.169.3.164.5.1.1.1 read-only INTEGER	The unique identifier of the instance of an IPXPROTO process. This value, which along with tdmNwServerInstance Id serves as the index to entries in the tdmNwServerTable, corresponds to the value of tdmNwSaIpxProto-BasicSysInstance in the tdmNwSaIPX-PROTONamesTable.	Assigned by subagent.
tdmNwServerInstanceId 1.3.6.1.4.1.169.3.164.5.1.1.2 read-only INTEGER	The unique identifier of the instance of the tuple tdmNwServer-Name, tdmNwServer-Address, and tdmNw-ServerType.This identifier in combination with the value of tdmNw-ServerEntryInstanceI dserve as the index to entries in the tdmNwServerTable.	Assigned by subagent.
tdmNwServerName 1.3.6.1.4.1.169.3.164.5.1.1.3 read-only DisplayString (SIZE (0..48))	The name of the server discovered.	Value obtained using a mechanism equivalent to the SCF STATUS SERVER command.
tdmNwServerAddress 1.3.6.1.4.1.169.3.164.52.1.1.4 read-only DisplayString (SIZE (0..28))	The IPX network address assigned to the server on a Novell LAN.	`0x`*`network`*`:`*`node`*`:` *`socket`*, where *`network`* is 8 hexademical digits, *`node`* is 12 hexadecimal digits, and *`socket`* is 4 hexadecimal digits. Value derived using a mechanism equivalent to the SCF STATUS SERVER command.

Table 13-8. tdmNwServerEntry Objects Supported by the Subagent (page 2 of 2)

Object and Attributes	Definition	Description of Value
tdmNwServerType 1.3.6.1.4.1.169.3.164.5.1.1.5 read-only INTEGER	The type of the server.	Some well-known server types include: Unknown (0), User (1), User Group (2), Print Queue (3), File Server (4), Job Server (5), Gateway (6), Print Server (7), Archive Queue (8), Archive Server (9), Job Queue (10), Administration (11), Remote Bridge Server (36), Advertising Print Server (71), NetWare 386 (263), HP ODBC Server (1140), HP Telserv Server (1479), reserved by HP (1480..1488), reserved by each customer site (1489..1490). Value derived using a mechanism equivalent to the SCF STATUS SERVER command.
tdmNwServerHops 1.3.6.1.4.1.169.3.164.5.1.1.6 read-only INTEGER	The number of hops (one hop equals one network) to get to the related server.	Integer value obtained using a mechanism equivalent to the SCF STATUS SERVER command.

zipx Group Table Maintenance

The subagent creates an entry in the zipx group tables whenever it creates an entry in the tdmNwSaIPXPROTONamesTable. Entries cannot be deleted; to deactivate an entry, set the value of tdmNwSaIpxProtoState to `disconnected`. Values in any zipx group table are updated when its refresh timer has run down; the refresh timer minutes are specified by using the -r startup parameter or setting the value of tdmNwSaRefreshTimer. Values are also updated when the value of tdmNwSaRefreshNow is set to `forceDynamicRefresh`.

zisa Group

The zisa group contains objects that describe IPX/SPX Subagent process attributes. This group consists of a subtree named tdmNwSubagent, which contains a group and two tables:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  tandem (169)
                     nonstopsystems (3)
                        zisa (196)
                           tdmNwSubagent (1)
                              tdmNwSaConfig (1)
                              tdmNwSaAgentNamesTable (2)
                              tdmNwSaIPXPROTONamesTable (3)
```

MIB Objects

This subsection contains three tables, Table 13-9 through Table 13-11, which describe the MIB objects in the group and tables in the tdmNwSubagent subtree.

tdmNwSaConfig Group Objects

Table 13-9 describes tdmNwSaConfig group objects. This group contains a collection of scalar objects characterizing attributes of the subagent process:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zisa (196)
                  tdmNwSubagent (1)
                    tdmNwSaConfig (1)
                      tdmNwSaProcessName (1)
                      tdmNwSaProcessVersion (2)
                      tdmNwSaProcessPriority (3)
                      tdmNwSaProcessPAID (4)
                      tdmNwSaPrimaryPID (5)
                      tdmNwSaBackupPID (6)
                      tdmNwSaBackupCPU (7)
                      tdmNwSaBackupState (8)
                      tdmNwSaBackupStateReason (9)
                      tdmNwSaSwitchToBackupNow (10)
                      tdmNwSaEmsCollectorName (11)
                      tdmNwSaEmsCollectorState (12)
                      tdmNwSaEmsCollectorStateReason (13)
                      tdmNwSaKeepAliveTimer (14)
                      tdmNwSaRefreshTimer (15)
                      tdmNwSaRefreshNow (16)
```

Table 13-9. tdmNwSaConfig Objects Supported by the Subagent (page 1 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSaProcessName 1.3.6.1.4.1.169.3.196.1.1.1 read-only DisplayString (SIZE (0..20))	The name of the subagent process.	`\`*`system`*`.$`*`process`*, where *`system`* and *`process`* are 1 to 7 alphanumeric characters, starting with a letter.
tdmNwSaProcessVersion 1.3.6.1.4.1.169.3.196.1.1.2 read-only DisplayString (SIZE (0..32))	The version of the subagent process object file.	This value is the same as the VPROC value.
tdmNwSaProcessPriority 1.3.6.1.4.1.169.3.196.1.1.3 read-write INTEGER (0..255)	The process priority at which the subagent is running.	An integer returned by a PROCESS_GETINFO_ procedure call.
tdmNwSaProcessPAID 1.3.6.1.4.1.169.3.196.1.1.4 read-only DisplayString (SIZE (0..10))	The process accessor ID (PAID) of the subagent process.	*`group`*,*`user`*, where *`group`* and *`user`* have a value from 0 through 255. The value is returned by a PROCESS_GETINFO_ procedure call.
tdmNwSaPrimaryPID 1.3.6.1.4.1.169.3.196.1.1.5 read-only DisplayString (SIZE (0..15))	The process ID (PID) of the subagent's primary process.	`cpu,pin`, returned by a PROCESSHANDLE_ GETMINE_ procedure call.
tdmNwSaBackupPID 1.3.6.1.4.1.169.3.196.1.1.6 read-only DisplayString (SIZE (0..15)	The process ID (PID) of the subagent's backup process.	`cpu,pin`, returned by a PROCESS_CREATE_ procedure call.
tdmNwSaBackupCPU 1.3.6.1.4.1.169.3.196.1.1.7 read-write INTEGER (-1..15)	The CPU in which the subagent's backup process runs.	A value of -1 disables the backup process. Changing the value to 0 through 15 stops any current backup process and starts a new backup process in the specified CPU. The value can be controlled by a manager or by the -b startup parameter.
tdmNwSaBackupState 1.3.6.1.4.1.169.3.196.1.1.8 read-write INTEGER { disconnected (1), connected (2)}	The state of the subagent's backup process.	One of the values shown in the first column.

Table 13-9. tdmNwSaConfig Objects Supported by the Subagent (page 2 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSaBackupStateReason 1.3.6.1.4.1.169.3.196.1.1.9 read-only INTEGER { rowcreate (1), operatorrequest (2), subagentkeepalive (3), invalidprocessname (4), incorrectprocessname (5), resourceproblem (6)}	The reason for the value of tdmNwSa-BackupState.	One of the values shown in the first column.
tdmNwSaSwitchToBackupNow 1.3.6.1.4.1.169.3.196.1.1.10 read-write INTEGER { autoProcessPairControl (1), forceBackupTakeover (2)}	A switch that causes the subagent's backup process to take over when set to 2.	The value retrieved is always 1. When you set the value to 2, an SNMP genErr is sent to the requesting manager, even though the backup process actually takes over.
tdmNwSaEmsCollectorName 1.3.6.1.4.1.169.3.196.1.1.11 read-write DisplayString (SIZE (0..20))	The process name of the EMS collector to which the subagent sends its EMS events.	`[\`*`system`*`.]$`*`ems-collector`*, where *`system`* and *`ems-collector`* are alphanumeric strings starting with a letter. The default value is $0, but it can be changed with the -c startup parameter or from a manager.
tdmNwSaEmsCollectorState 1.3.6.1.4.1.169.3.196.1.1.12 read-write INTEGER { disconnected (1), connected (2)}	The state of the connection with the EMS collector process.	One of the values shown in the first column.
tdmNwSaEmsCollectorStateReason 1.3.6.1.4.1.169.3.196.1.1.13 read-only INTEGER { rowcreate (1), operatorrequest (2), subagentkeepalive (3), invalidprocessname (4), incorrectprocessname (5), resourceproblem (6)}	The reason for the value of tdmNwSaEms-CollectorState.	One of the values shown in the first column.

Table 13-9. tdmNwSaConfig Objects Supported by the Subagent (page 3 of 3)

Object and Attributes	Definition	Description of Value
tdmNwSaKeepAliveTimer 1.3.6.1.4.1.169.3.196.1.1.14 read-write INTEGER (0..32000)	The number of minutes between subagent attempts to connect to a disconnected backup process, SNMP agent process, IPXPROTO process, or EMS collector process.	The value can be set at startup by using the -k startup parameter or from a manager. A value of 0 suppresses retries when disconnections occur.
tdmNwSaRefreshTimer 1.3.6.1.4.1.169.3.196.1.1.15 read-write INTEGER (0..32000)	The number of minutes that must pass between subagent accesses of the following tables before the subagent refreshes the table's values: ipxBasicSysTable tdmNwInfoTable tdmNwIPXTable tdmNwSPXTable tdmNwNCPTable tdmNwServerTable	The value can be set at startup by using the -r startup parameter or from a manger. The default value is 10. A value of 0 causes the subagent to refresh data values by retrieving new data from an IPXPROTO process every time it receives a request.
tdmNwSaRefreshNow 1.3.6.1.4.1.169.3.196.1.1.16 read-write INTEGER { autoDynamicRefresh (1), forceDynamicRefresh (2)}	A switch that can be used to force the subagent to refresh its MIB data.	The value retrieved is always 1.

tdmNwSaAgentNamesTable Entries

Table 13-10 describes the objects in each tdmNwSaAgentNamesTable entry. Each entry describes a SNMP agent with which the IPX/SPX Subagent is configured to communicate:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  tandem (169)
                     nonstopsystems (3)
                        zisa (196)
                           tdmNwSubagent (1)
                              tdmNwSaAgentNamesTable (2)
                                 tdmNwSaAgentNamesEntry (1)
                                    tdmNwSaAgentName (1)
                                    tdmNwSaAgentState (2)
                                    tdmNwSaAgentStateReason (3)
```

Table 13-10. tdmNwSaAgentNamesEntry Objects Supported by the Subagent

Object and Attributes	Definition	Description of Value
tdmNwSaAgentName 1.3.6.1.4.1.169.3.196.1.2.1.1 read-write DisplayString (SIZE (0..20))	The name of a SNMP agent process.	`[\`*`system`*`.]$`*`process`*
tdmNwSaAgentState 1.3.6.1.4.1.169.3.196.1.2.1.2 read-write INTEGER { disconnected (1), connected (2)}	The state of the connection between the subagent and the SNMP agent processes.	One of the values in the first column.
tdmNwSaAgentStateReason 1.3.6.1.4.1.169.3.196.1.2.1.3 read-only INTEGER { rowcreate (1), operatorrequest (2), subagentkeepalive (3), invalidprocessname (4), incorrectprocessname (5), resourceproblem (6)}	The reason for the value of tdmNwSa-AgentState.	One of the values in the first column.

tdmNwSaIPXPROTONamesTable Entries

Table 13-11 describes the objects in each tdmNwSaIPXPROTONamesTable entry. Each entry describes an IPXPROTO process the subagent is configured to monitor:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprises (1)
            tandem (169)
              nonstopsystems (3)
                zisa (196)
                  tdmNwSubagent (1)
                    tdmNwSaIPXPROTONamesTable (3)
                      tdmNwSaIPXPROTONamesEntry (1)
                        tdmNwSaIpxProtoName (1)
                        tdmNwSaIpxProtoBasicSysInstance (2)
                        tdmNwSaIpxProtoState (3)
                        tdmNwSaIpxProtoStateReason (4)
```

Table 13-11. tdmNwSaIPXPROTONamesEntry Objects Supported by the Subagent (page 1 of 2)

Object and Attributes	Definition	Description of Value

tdmNwSaIpxProtoName
1.3.6.1.4.1.169.3.196.1.3.1.1
read-write
DisplayString (SIZE (0..15))
	The name of the IPXPROTO process.	`[\`*`system`*`.]`
		`$`*`process`*

tdmNwSaIpxProtoBasicSysInstance 1.3.6.1.4.1.169.3.196.1.3.1.2 read-only INTEGER	The unique identifier of the instance of the IPXPROTO process. This value is used as the index to the tables in the zipx group and the ipxBasicSysTable.	Assigned by subagent.
tdmNwSaIpxProtoState 1.3.6.1.4.1.169.3.196.1.3.1.3 read-write INTEGER { disconnected (1), connected (2)}	The state of the connection between the subagent and the IPXPROTO process.	One of the values in the first column.

Table 13-11. tdmNwSaIPXPROTONamesEntry Objects Supported by the Subagent (page 2 of 2)

Object and Attributes	Definition	Description of Value
tdmNwSaIpxProtoStateReason 1.3.6.1.4.1.169.3.196.1.3.1.4 read-only INTEGER { rowcreate (1), operatorrequest (2), subagentkeepalive (3), invalidprocessname (4), incorrectprocessname (5), resourceproblem (6)}	The reason for the value of tdmNwSaIpx-ProtoState.	One of the values in the first column.

zisa Group Table Maintenance

The subagent creates an entry in the tdmNwSaAgentNamesTable when a SNMP agent connection is configured. It creates an entry in the tdmNwSaIPXPROTONamesTable when an IPXPROTO process connection is configured. SNMP agent and IPXPROTO connections are configured by using the -a and -i startup parameters, respectively, or by setting the value of tdmNwSaAgentName or tdmNwSaIpxProtoName, respectively.

When the subagent starts, it establishes connections with IPXPROTO and SNMP agent processes named in the startup line and sets the value of their MIB state objects to connected if a connection could be established. After startup, you can establish additional process connections by adding entries to the tdmNwSaIPXPROTONamesTable and the tdmNwSaAgentNamesTable:

Process	SNMP Operation
IPXPROTO process	Add an entry to the tdmNwSaIPXPROTONamesTable, setting the value of tdmNwSaIpxProtoName to the IPXPROTO process name.
SNMP agent process	Add an entry to the tdmNwSaAgentNamesTable, setting the value of tdmNwSaAgentName to the SNMP agent process name.

If you do not set the state object associated with a new entry to `connected`, the subagent attempts to establish a connection when its keep-alive timer runs down.

Entries in the zisa group tables cannot be deleted; to deactivate an entry, set the value of its state object to `disconnected`.

Values in the zisa group tables are updated whenever an event occurs that causes them to change.

EMS Support

The events generated by the IPX/SPX Subagent are listed in Table 13-12. All the events are standard events, as the table indicates. Refer to the *EMS Manual* for a complete description of standard events.

Table 13-12. IPX/SPX Subagent Event Messages

Number	Event	Standard Event
1001	ZISA-EVT-SUBAGENT-AVAILABLE	Object Available
1002	ZISA-EVT-SUBAGENT-UNAVAILABLE	Object Unavailable
1003	ZISA-EVT-AGENT-AVAILABLE	Object Available
1004	ZISA-EVT-AGENT-UNAVAILABLE	Object Unavailable
1005	ZISA-EVT-BACKUP-AVAILABLE	Object Available
1006	ZISA-EVT-BACKUP-UNAVAILABLE	Object Unavailable
1007	ZISA-EVT-EMSCOLL-AVAILABLE	Object Available
1008	ZISA-EVT-EMSCOLL-UNAVAILABLE	Object Unavailable
1009	ZISA-EVT-IPXPROTO-AVAILABLE	Object Available
1010	ZISA-EVT-IPXPROTO-UNAVAILABLE	Object Unavailable
1011	ZISA-EVT-MEMORY-ALLOC-FAILURE	Transient Fault
1012	ZISA-EVT-PROCESS-TRAPPED	Transient Fault

Each event is described individually in Event Message Descriptions on page 13-49.

Many of the tokens, structures, and values appearing in the events are defined in the ZSPI and ZEMS data definition files ZSPIDDL and ZEMSDDL and their associated language-specific files. Data elements defined by the IPX/SPX Subagent are in the file ZISADDL and associated language-specific files:

ZISADDL	Data Definition Language (DDL) definitions from which the language-specific definitions are derived
ZISAC	Definitions for C programs
ZISACOB	Definitions for COBOL programs
ZISAPAS	Definitions for Pascal programs
ZISASEGF	Definitions for TACL segment file
ZISATACL	Definitions for TACL programs
ZISATAL	Definitions for TAL programs

All these files except ZISASEGF are located in $<*dsv*>.ZSPIDEF. ZISASEGF is located in $<*dsv*>.ZSPISEGF. The source template file name for the subagent is $<*dsv*>.ZTEMPL.SISATMPL, and the compiled template file name is $<*dsv*>.ZTEMPL.ZISATMPL. All SPI definition files are usually located in the ZSPIDEF subvolume of the NonStop Kernel operating system installation volume. The

SPI Programming Manual has information on the ZSPI data definitions and data definition files in general. The *EMS Manual* has information on ZEMS data definitions.

Subsystem ID

The subsystem ID that the subagent uses to identify itself as the source of event messages is:

```
CONSTANT ZISA-VAL-VERSION             VALUE VERSION "D30".
CONSTANT ZSPI-SSN-ZISA                VALUE 196.

DEFINITION ZISA-VAL-SSID              TACL   SSID.
  02 z-filler  TYPE character 8       VALUE is ZSPI-VAL-
TANDEM.
  02 z-owner   TYPE ZSPI-DDL-CHAR8    REDEFINES z-filler.
  02 z-number  TYPE ZSPI-DDL-INT      VALUE IS ZSPI-SSN-ZISA.
  02 z-version TYPE ZSPI-DDL-UINT     VALUE is ZISA-VAL-
VERSION.
END
```

Tokens in ZISA Event Messages

This subsection lists the tokens that the subagent includes in the event messages that it generates.

IPX Subagent (ZISA) Tokens

Table 13-13 lists all the tokens defined by the subagent. These tokens are defined in the ZISADDL definition file, and are described where they appear in the event message descriptions later in this section.

Table 13-13. ZISA Tokens in ZISA Event Messages (page 1 of 2)

Token	Contents
ZISA-TKN-SUBAGENT	Name of a subagent process
ZISA-TKN-AGENT	Name of a SNMP agent process
ZISA-TKN-BACKUP	Name of a subagent process whose backup process started or stopped
ZISA-TKN-EMSCOLLECTOR	Name of an EMS collector process
ZISA-TKN-IPXPROTO	Name of an IPXPROTO process

ZISA-TKN-OBJ-STATE	The state object value, which can be one of these:
	`connected`
	`disconnected`

Table 13-13. ZISA Tokens in ZISA Event Messages (page 2 of 2)

Token	Contents

ZISA-TKN-OBJ-STATE-REASON	The state reason object, which can be one of the following:
	`rowcreate`
	`operatorrequest`
	`subagentkeepalive`
	`invalidprocessname`
	`incorrectprocessname`
	`resourceproblem`

ZISA-TKN-REFRESH-RATE	The refresh interval
ZISA-TKN-KEEPALIVE-RATE	The keep-alive interval
ZISA-TKN-MEM-ALLOC-SIZE	The number of bytes the subagent tried to allocate unsuccessfully

Standard SPI Tokens

Table 13-14 lists the SPI-defined tokens that are used by the subagent. The standard SPI tokens are defined in the ZSPIDDL definition file. See the *SPI Programming Manual* for more information on these tokens and related data definitions.

Table 13-14. ZSPI Tokens in ZISA Event Messages

Token	Contents
ZSPI-TKN-MAX-FIELD-VERSION	The latest version of any extensible structured token in the EMS buffer
ZSPI-TKN-SSID	The ZISA subsystem ID
ZSPI-TKN-USEDLEN	The number of used bytes in the EMS buffer

Standard EMS Tokens

Table 13-15 lists the EMS-defined tokens that are used by the subagent. These tokens are defined in the ZEMSDDL definition file. See the *EMS Manual* for more information on these tokens and related data definitions.

Table 13-15. ZEMS Tokens in ZISA Event Messages (page 1 of 2)

Token	Contents
ZEMS-TKN-BATCHJOB-ID	Batch job ID of process
ZEMS-TKN-CONTENT-STANDARD	Type of standard event
ZEMS-TKN-CONTENT-USER	Type of user-defined event, always null
ZEMS-TKN-CPU	CPU number of event sender
ZEMS-TKN-EMPHASIS	Critical/Noncritical event flag
ZEMS-TKN-EVENTNUMBER	Event number

Table 13-15. ZEMS Tokens in ZISA Event Messages (page 2 of 2)

Token	Contents
ZEMS-TKN-GENTIME	Event generation time
ZEMS-TKN-LOGTIME	Event log time
ZEMS-TKN-NAME-MANAGER	Manager process name
ZEMS-TKN-NODENUM	System number of event sender
ZEMS-TKN-PIN	PIN of event sender
ZEMS-TKN-SUBJECT-MARK	Event subject marker
ZEMS-TKN-SUPPRESS-DISPLAY	Display/do not display event flag, always “display”
ZEMS-TKN-USERID	User ID of event sender
ZEMS-TKN-CHANGE-REASON	Object state change reason. Private enumerations for Object Available/Object Unavailable events (1001-1010): ZISA-VAL-CHRSN-ROWCREATE ZISA-VAL-CHRSN-OPERATOR-RQST ZISA-VAL-CHRSN-SUBAGENT-KEEP ZISA-VAL-CHRSN-INVALID-PROC ZISA-VAL-CHRSN-INCORR-PROC ZISA-VAL-CHRSN-RESOURCE-PROB ZISA-VAL-CHRSN-PROCESS-STARTED Private enumerations for Transient Fault event 1012: ZISA-VAL-CR-PROCESS-STOPPED ZISA-VAL-CR-PROCESS-SIGILL ZISA-VAL-CR-PROCESS-SIGABRT ZISA-VAL-CR-PROCESS-SIGFPE ZISA-VAL-CR-PROCESS-SIGSEGV ZISA-VAL-CR-PROCESS-SIGSTK ZISA-VAL-CR-PROCESS-SIGTIMEOUT ZISA-VAL-CR-PROCESS-SIGTERM ZISA-VAL-CR-PROCESS-NOMEM
ZEMS-TKN-STATE-CURRENT	Current object state; private enumerations: ZISA-VAL-STATE-DISCONNECTED ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-STATE-PREVIOUS	Previous object state; private enumerations: ZISA-VAL-STATE-DISCONNECTED ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-TXFAULT-TYPE	Type of transient fault; private enumerations: ZISA-VAL-TF-IO ZISA-VAL-TF-MEM ZISA-VAL-TF-PROG

Event Message Descriptions

ZISA event messages are described in order by event number. Each description includes:

- A list of tokens that are included in the event message. Tokens listed as “unconditional” always appear in the message. Tokens listed as “conditional” are included only under described conditions.

  Standard ZSPI and ZEMS tokens are listed only if they contain information that appears in the printed message test or if they contain ZISA-defined values.

- The event-message text that is generated when the contents of the event message are displayed according to the standard event message template defined in the file ZEMSTMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the standard event templates in SEMSTMPL.

- Descriptions of listed tokens or values.
- The cause of the event, the conditions that prompted the subagent to generate the event message.
- The effects associated with or resulting from the cause.
- Recovery procedures that you can follow to solve the problem reported by the event message.
- An example of the formatted message.

1001: ZISA-EVT-SUBAGENT-AVAILABLE

A subagent process has started successfully.

Unconditional Tokens **Value**

```
ZISA-TKN-SUBAGENT              <1> IPX/SPX Subagent process
ZEMS-TKN-EVENTNUMBER           name
ZEMS-TKN-CHANGE-REASON         <2> ZISA-EVT-SUBAGENT-AVAILABLE
ZEMS-TKN-STATE-PREVIOUS        <3> ZISA-VAL-CHRSN-reason
ZEMS-TKN-STATE-CURRENT         <4> ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-CONTENT-USER          <5> ZISA-VAL-STATE-CONNECTED
ZISA-TKN-OBJ-STATE             <6> undefined (null)
ZISA-TKN-OBJ-STATE-REASON          ZISA-VAL-CONNECTED
                                   ZISA-VAL-reason
```

Conditional Tokens

None

Message Text

```
Object available  IPX/SPX-subagent-process - <1>, event
number:  <2>, reason:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-SUBAGENT`

identifies the subject of the event, always the IPX/SPX Subagent process. The DDL heading of this token (“IPX/SPX-subagent-process”) and the token’s value (the name of the subagent process) are inserted in the message text.

`ZISA-EVT-SUBAGENT-AVAILABLE`

is the event number (1001). The DDL AS clause of this value (“subagent-process-available”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “reason:”:

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-SUBAGENT-KEEP	“subagent-keepalive”

`ZISA-VAL-STATE-DISCONNECTED`

is the previous state. The DDL AS clause of this value (“disconnected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-CONNECTED`

is the current state. The DDL AS clause of this value (“connected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-CONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-SUBAGENT-KEEPALIVE

Cause. A subagent process has started successfully.

Effect. The subagent is available for processing requests.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001001 Object
                              available IPX/SPX-SNMP-subagent-process -
                              \COMM.$ZISA, event number:
                              subagent-process-available, reason:
                              operatorrequest, previous state:
                              disconnected, current state: connected, user
                              content: undefined
```

1002: ZISA-EVT-SUBAGENT-UNAVAILABLE

A subagent process has stopped.

Unconditional Tokens	Value
ZISA-TKN-SUBAGENT	<1> *IPX/SPX Subagent process name*
ZEMS-TKN-EVENTNUMBER	<2> ZISA-EVT-SUBAGENT-UNAVAILABLE
ZEMS-TKN-CHANGE-REASON	<3> ZISA-VAL-CHRSN-OPERATOR-RQST
ZEMS-TKN-STATE-PREVIOUS	<4> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-STATE-CURRENT	<5> ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-CONTENT-USER	<6> undefined (null)
ZISA-TKN-OBJ-STATE	ZISA-VAL-DISCONNECTED
ZISA-TKN-OBJ-STATE-REASON	ZISA-VAL-OPERATOR-REQUEST

Conditional Tokens

None

Message Text

```
Object unavailable  IPX/SPX-subagent-process - <1>, event
number:  <2>, cause:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-SUBAGENT`

identifies the subject of the event, always the IPX/SPX Subagent process. The DDL heading of this token (“IPX/SPX-subagent-process”) and the token’s value (the name of the subagent process) are inserted in the message text.

`ZISA-EVT-SUBAGENT-UNAVAILABLE`

is the event number (1002). The DDL AS clause of this value (“subagent-process-unavailable”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-OPERATOR-RQST`

is the change reason. The DDL AS clause of this value (“operatorrequest”) appears in the message text following “cause:”.

`ZISA-VAL-STATE-CONNECTED`

is the previous state. The DDL AS clause of this value (“connected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-DISCONNECTED`

is the current state. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-DISCONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-OPERATOR-REQUEST`

is the value assigned to the ZISA-TKN-OBJ-STATE-REASON token.

Cause. A subagent primary process was intentionally stopped.

Effect. The backup process takes over if it exists.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001002 Object
                              unavailable IPX/SPX-SNMP-subagent-process -
                              \COMM.$ZISA, event number:
                              subagent-unavailable, cause:
                              operatorrequest, previous state: connected,
                              current state: disconnected, user content:
                              undefined
```

1003: ZISA-EVT-AGENT-AVAILABLE

The subagent has successfully connected to a SNMP agent process.

Unconditional Tokens		Value
ZISA-TKN-AGENT	<1>	*SNMP agent process name*
ZEMS-TKN-EVENTNUMBER	<2>	ZISA-EVT-AGENT-AVAILABLE
ZEMS-TKN-CHANGE-REASON	<3>	ZISA-VAL-CHRSN-*reason*
ZEMS-TKN-STATE-PREVIOUS	<4>	ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-STATE-CURRENT	<5>	ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER	<6>	undefined (null)
ZISA-TKN-OBJ-STATE		ZISA-VAL-CONNECTED
ZISA-TKN-OBJ-STATE-REASON		ZISA-VAL-*reason*

Conditional Tokens

None

Message Text

```
Object available  agent-resource-object - <1>, event number:
<2>, reason:  <3>, previous state: <4>, current state: <5>,
user content:  <6>
```

`ZISA-TKN-AGENT`

identifies the subject of the event, always a SNMP agent process. The DDL heading of this token (“agent-resource-object”) and the token’s value (the name of the SNMP agent process) are inserted in the message text.

`ZISA-EVT-AGENT-AVAILABLE`

is the event number (1003). The DDL AS clause of this value (“agent-available”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “reason:”:

Token Value	**DDL AS Clause**
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-SUBAGENT-KEEP	“subagent-keepalive”

`ZISA-VAL-STATE-DISCONNECTED`

is the previous state. The DDL AS clause of this value (“disconnected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-CONNECTED`

is the current state. The DDL AS clause of this value (“connected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-CONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-SUBAGENT-KEEPALIVE

Cause. The subagent has successfully connected with a SNMP agent process.

Effect. The subagent can receive requests from the SNMP agent process.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001003 Object
                              available agent-resource-object - $ZSNMP,
                              event number: agent-available, reason:
                              subagent-keepalive, previous state:
                              disconnected, current state: connected, user
                              content: undefined
```

1004: ZISA-EVT-AGENT-UNAVAILABLE

The subagent cannot establish a connection with a SNMP agent process.

Unconditional Tokens		Value
ZISA-TKN-AGENT	<1>	*SNMP agent process name*
ZEMS-TKN-EVENTNUMBER	<2>	ZISA-EVT-AGENT-UNAVAILABLE
ZEMS-TKN-CHANGE-REASON	<3>	ZISA-VAL-CHRSN-*reason*
ZEMS-TKN-STATE-PREVIOUS	<4>	ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-STATE-CURRENT	<5>	ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-CONTENT-USER	<6>	undefined (null)
ZISA-TKN-OBJ-STATE		ZISA-VAL-DISCONNECTED
ZISA-TKN-OBJ-STATE-REASON		ZISA-VAL-*reason*

Conditional Tokens

None

Message Text

```
Object unavailable  agent-resource-object - <1>, event
number:  <2>, cause:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-AGENT`

identifies the subject of the event, always a SNMP agent process. The DDL heading of this token (“agent-resource-object”) and the token’s value (the name of the SNMP agent process) are inserted in the message text.

`ZISA-EVT-AGENT-UNAVAILABLE`

is the event number (1004). The DDL AS clause of this value (“agent-unavailable”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “cause:”:

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-INVALID-PROC	“invalid-processname”
ZISA-VAL-CHRSN-INCORR-PROC	“incorrect-processname”
ZISA-VAL-CHRSN-RESOURCE-PROB	“resource-problem”

`ZISA-VAL-STATE-CONNECTED`

is the previous state. The DDL AS clause of this value (“connected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-DISCONNECTED`

is the current state. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-DISCONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-INVALID-PROCESSNAME
ZISA-VAL-INCORRECT-PROCESSNAME
ZISA-VAL-RESOURCE-PROBLEM

Cause. The cause depends on the reason for the disconnection:

Change Reason	Cause
operatorrequest	The value of tdmNwSaAgentState object for the SNMP agent process was deliberately set to `disconnected`.

Change Reason	Cause
invalid-processname	A Set operation on tdmNwSaAgentName specified a syntactically incorrect value.
incorrect-processname	A SNMP agent process was stopped, and a non-agent process by the same name was started. When the subagent attempted to reconnect with the stopped agent process, the name of the process running did not identify a SNMP agent process.
resource-problem	A SNMP agent process failed.

Effect. The subagent cannot receive requests or transmit responses using the SNMP agent process.

Recovery. Correcting the situation depends on the reason for the disconnection:

Change Reason	Recovery
operatorrequest	No action is required because the stoppage was deliberate.
invalid-processname	Resubmit the Set request using the correct syntax for a NonStop Kernel process name.
incorrect-processname	Start a SNMP agent process using a unique process name.
resource-problem	Try to set the value of tdmNwSaAgentState to `connected`. If the operation fails, refer to Section 7, Troubleshooting the SNMP Agent, for other ideas.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001004 Object
                              unavailable agent-resource-object - $ZSNMP,
                              event number: agent-unavailable, cause:
                              resource-problem, previous state: connected,
                              current state: disconnected, user content:
                              undefined
```

1005: ZISA-EVT-BACKUP-AVAILABLE

The subagent's backup process started and the subagent checkpointed its state to the backup process.

Unconditional Tokens	Value
ZISA-TKN-BACKUP	<1> *IPX/SPX subagent process name*
ZEMS-TKN-EVENTNUMBER	<2> ZISA-EVT-BACKUP-AVAILABLE
ZEMS-TKN-CHANGE-REASON	<3> ZISA-VAL-CHRSN-*reason*
ZEMS-TKN-STATE-PREVIOUS	<4> ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-STATE-CURRENT	<5> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER	<6> undefined (null)
ZISA-TKN-OBJ-STATE	ZISA-VAL-CONNECTED
ZISA-TKN-OBJ-STATE-REASON	ZISA-VAL-*reason*

Conditional Tokens

None

Message Text

```
Object available  backup-process-resource-object - <1>, event
number:  <2>, reason:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-BACKUP`

identifies the subject of the event, always an IPX/SPX Subagent process. The DDL heading of this token ("backup-process-resource-object") and the token's value (the name of the subagent process) are inserted in the message text.

`ZISA-EVT-BACKUP-AVAILABLE`

is the event number (1005). The DDL AS clause of this value ("backup-available") appears in the message text following "event number:".

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following "reason:":

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	"operatorrequest"
ZISA-VAL-CHRSN-SUBAGENT-KEEP	"subagent-keepalive"

`ZISA-VAL-STATE-DISCONNECTED`

is the previous state. The DDL AS clause of this value ("disconnected") appears in the message text following "previous state:".

`ZISA-VAL-STATE-CONNECTED`

is the current state. The DDL AS clause of this value (“connected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-CONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-SUBAGENT-KEEPALIVE

Cause. The subagent’s backup process has successfully started.

Effect. If the primary subagent process fails, the backup process takes over as the primary subagent process and a new backup process is created.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001005 Object
                              available backup-process-resource-object -
                              \COMM.$ZISA, event number: backup-available,
                              reason: subagent-keepalive, previous state:
                              disconnected, current state: connected,
                              usercontent: undefined
```

1006: ZISA-EVT-BACKUP-UNAVAILABLE

The subagent's backup process stopped.

Unconditional Tokens **Value**

```
ZISA-TKN-BACKUP              <1> IPX/SPX subagent process
ZEMS-TKN-EVENTNUMBER         name
ZEMS-TKN-CHANGE-REASON       <2> ZISA-EVT-BACKUP-AVAILABLE
ZEMS-TKN-STATE-PREVIOUS      <3> ZISA-VAL-CHRSN-reason
ZEMS-TKN-STATE-CURRENT       <4> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER        <5> ZISA-VAL-STATE-DISCONNECTED
ZISA-TKN-OBJ-STATE           <6> undefined (null)
ZISA-TKN-OBJ-STATE-REASON        ZISA-VAL-DISCONNECTED
                                 ZISA-VAL-reason
```

Conditional Tokens

None

Message Text

```
Object unavailable  backup-process-resource-object - <1>,
event number:  <2>, cause:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-BACKUP`

identifies the subject of the event, always an IPX/SPX subagent process. The DDL heading of this token ("backup-process-resource-object") and the token's value (the name of the subagent process) are inserted in the message text.

`ZISA-EVT-BACKUP-UNAVAILABLE`

is the event number (1006). The DDL AS clause of this value ("backup-unavailable") appears in the message text following "event number:".

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following "reason:":

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	"operatorrequest"
ZISA-VAL-CHRSN-RESOURCE-PROB	"resource-problem"

`ZISA-VAL-STATE-CONNECTED`

is the previous state. The DDL AS clause of this value ("connected") appears in the message text following "previous state:".

`ZISA-VAL-STATE-DISCONNECTED`

is the current state. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-DISCONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-RESOURCE-PROBLEM

Cause. The cause depends on the reason for the stoppage:

Change Reason	**Cause**
operatorrequest	The value of tdmNwSaBackupState object was deliberately set to `disconnected`.
resource-problem	The CPU in which the backup process was running failed, or a critical resource was not available.

Effect. The subagent is no longer running in fault-tolerant mode.

Recovery. Correcting the situation depends on the reason for the stoppage:

Change Reason	**Recovery**
operatorrequest	No action is required because the stoppage was deliberate.
resource-problem	If a CPU failure caused the problem, use a Set operation to change the value of tdmNwSaBackupCPU to an available CPU, then Set the value of tdmNwSaBackupState to `connected`. You can also reload the CPU that failed, and the subagent backup process automatically restarts.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001006 Object
                              unavailable backup-process-resource-object
                              - \COMM.$ZISA, event number:
                              backup-unavailable, cause: operatorrequest,
                              previous state: connected, current state:
                              disconnected, user content: undefined
```

1007: ZISA-EVT-EMSCOLL-AVAILABLE

The subagent has successfully connected to an EMS collector process.

Unconditional Tokens **Value**

```
ZISA-TKN-EMSCOLLECTOR         <1> EMS collector process name
ZEMS-TKN-EVENTNUMBER          <2> ZISA-EVT-EMSCOLL-AVAILABLE
ZEMS-TKN-CHANGE-REASON        <3> ZISA-VAL-CHRSN-reason
ZEMS-TKN-STATE-PREVIOUS       <4> ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-STATE-CURRENT        <5> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER         <6> undefined (null)
ZISA-TKN-OBJ-STATE                ZISA-VAL-CONNECTED
ZISA-TKN-OBJ-STATE-REASON         ZISA-VAL-reason
```

Conditional Tokens

None

Message Text

```
Object available  emscollector-resource-object - <1>, event
number:  <2>, reason:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-EMSCOLLECTOR`

identifies the subject of the event, always an EMS collector process. The DDL heading of this token (“emscollector-resource-object”) and the token’s value (the name of the EMS collector process) are inserted in the message text.

`ZISA-EVT-EMSCOLL-AVAILABLE`

is the event number (1007). The DDL AS clause of this value (“emscollector-available”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “reason:”:

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-SUBAGENT-KEEP	“subagent-keepalive”

`ZISA-VAL-STATE-DISCONNECTED`

is the previous state. The DDL AS clause of this value (“disconnected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-CONNECTED`

is the current state. The DDL AS clause of this value ("connected") appears in the message text following "current state:".

`ZEMS-TKN-CONTENT-USER`

is assigned the value "undefined" in the message text following "user content:".

`ZISA-VAL-CONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-SUBAGENT-KEEPALIVE

Cause. The subagent has successfully connected with an EMS collector process.

Effect. The subagent can send EMS event messages to the EMS collector process.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001007 Object
                              available emscollector-resource-object -
                              $0, event number: emscollector-available,
                              reason: subagent-keepalive, previous state:
                              disconnected, current state: connected,
                              user content: undefined
```

1008: ZISA-EVT-EMSCOLL-UNAVAILABLE

The subagent cannot establish a connection with an EMS collector process.

Unconditional Tokens **Value**

```
ZISA-TKN-EMSCOLLECTOR        <1> EMS collector process name
ZEMS-TKN-EVENTNUMBER         <2> ZISA-EVT-EMSCOLL-UNAVAILABLE
ZEMS-TKN-CHANGE-REASON       <3> ZISA-VAL-CHRSN-reason
ZEMS-TKN-STATE-PREVIOUS      <4> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-STATE-CURRENT       <5> ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-CONTENT-USER        <6> undefined (null)
ZISA-TKN-OBJ-STATE               ZISA-VAL-DISCONNECTED
ZISA-TKN-OJB-STATE-REASON        ZISA-VAL-reason
```

Conditional Tokens

None

Message Text

```
Object unavailable  agent-resource-object - <1>, event
number:  <2>, cause:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

ZISA-TKN-EMSCOLLECTOR

identifies the subject of the event, always an EMS collector process. The DDL heading of this token (“emscollector-resource-object”) and the token’s value (the name of the EMS collector process) are inserted in the message text.

ZISA-EVT-EMSCOLL-UNAVAILABLE

is the event number (1008). The DDL AS clause of this value (“emscollector-unavailable”) appears in the message text following “event number:”.

ZISA-VAL-CHRSN-*reason*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “cause:”:

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-INVALID-PROC	“invalid-processname”
ZISA-VAL-CHRSN-INCORR-PROC	“incorrect-processname”
ZISA-VAL-CHRSN-RESOURCE-PROB	“resource-problem”

ZISA-VAL-STATE-CONNECTED

is the previous state. The DDL AS clause of this value (“connected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-DISCONNECTED`

is the current state. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-DISCONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-INVALID-PROCESSNAME
ZISA-VAL-INCORRECT-PROCESSNAME
ZISA-VAL-RESOURCE-PROBLEM

Cause. The cause depends on the reason for the disconnection:

Change Reason	**Cause**
operatorrequest	The value of tdmNwSaEmsCollectorState object for the EMS collector process was deliberately set to `disconnected`.
invalid-processname	A Set operation on tdmNwSaEmsCollectorName specified a syntactically incorrect value.
incorrect-processname	An EMS collector process was stopped, and a non-collector process by the same name was started. When the subagent attempted to reconnect with the stopped collector process, the name of the process running did not identify an EMS collector process.
resource-problem	Resources needed to support communications between the subagent and the EMS collector process are not available.

Effect. The subagent cannot send additional EMS events to the EMS collector until a connection is re-established.

Recovery. Correcting the situation depends on the reason for the disconnection:

Change Reason	Recovery
operatorrequest	No action is required because the stoppage was deliberate.
invalid-processname	Resubmit the Set request using the correct syntax for a NonStop Kernel process name.
incorrect-processname	Start an EMS collector process using a unique process name.
resource-problem	Try to set the value of tdmNwSaEmsCollectorState to `connected`. If the operation fails, refer to the *EMS Manual* for other ideas on making the EMS collector process available.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001008 Object
                              unavailable emscollector-resource-object -
                              $0, event number: emscollector-unavailable,
                              cause: operatorrequest, previous state:
                              connected, current state: disconnected, user
                              content: undefined
```

1009: ZISA-EVT-IPXPROTO-AVAILABLE

The subagent has successfully connected to an IPXPROTO process.

Unconditional Tokens / **Value**

Unconditional Tokens		Value
ZISA-TKN-IPXPROTO	<1>	*IPXPROTO process name*
ZEMS-TKN-EVENTNUMBER	<2>	ZISA-EVT-IPXPROTO-AVAILABLE
ZEMS-TKN-CHANGE-REASON	<3>	ZISA-VAL-CHRSN-*reason*
ZEMS-TKN-STATE-PREVIOUS	<4>	ZISA-VAL-STATE-DISCONNECTED
ZEMS-TKN-STATE-CURRENT	<5>	ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER	<6>	undefined (null)
ZISA-TKN-OBJ-STATE		ZISA-VAL-CONNECTED
ZISA-TKN-OBJ-STATE-REASON		ZISA-VAL-*reason*

Conditional Tokens

None

Message Text

```
Object available  ipxproto-resource-object - <1>, event
number:  <2>, reason:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-IPXPROTO`

identifies the subject of the event, always an IPXPROTO process. The DDL heading of this token (“ipxproto-resource-object”) and the token’s value (the name of the IPXPROTO process) are inserted in the message text.

`ZISA-EVT-EMSCOLL-AVAILABLE`

is the event number (1009). The DDL AS clause of this value (“ipxproto-available”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “reason:”:

Token Value	**DDL AS Clause**
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-SUBAGENT-KEEP	“subagent-keepalive”

`ZISA-VAL-STATE-DISCONNECTED`

is the previous state. The DDL AS clause of this value (“disconnected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-CONNECTED`

is the current state. The DDL AS clause of this value (“connected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-CONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-SUBAGENT-KEEPALIVE

Cause. The subagent has successfully connected with an IPXPROTO process.

Effect. The subagent can retrieve information from the IPXPROTO process to populate MIB tables that describe IPXPROTO processes.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001009 Object
                              available ipxproto-resource-object - $ZNV0,
                              event number: ipxproto-available, reason:
                              subagent-keepalive, previous state:
                              disconnected, current state: connected,
                              user content: undefined
```

1010: ZISA-EVT-IPXPROTO-UNAVAILABLE

The subagent cannot establish a connection with an IPXPROTO process.

Unconditional Tokens **Value**

```
ZISA-TKN-IPXPROTO             <1> IPXPROTO process name
ZEMS-TKN-EVENTNUMBER          <2> ZISA-EVT-IPXPROTO-
ZEMS-TKN-CHANGE-REASON        UNAVAILABLE
ZEMS-TKN-STATE-PREVIOUS       <3> ZISA-VAL-CHRSN-reason
ZEMS-TKN-STATE-CURRENT        <4> ZISA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER         <5> ZISA-VAL-STATE-DISCONNECTED
ZISA-TKN-OBJ-STATE            <6> undefined (null)
ZISA-TKN-OBJ-STATE-REASON         ZISA-VAL-DISCONNECTED
                                  ZISA-VAL-reason
```

Conditional Tokens

None

Message Text

```
Object unavailable  ipxproto-resource-object - <1>, event
number:  <2>, cause:  <3>, previous state: <4>, current
state: <5>, user content:  <6>
```

`ZISA-TKN-IPXPROTO`

identifies the subject of the event, always an IPXPROTO process. The DDL heading of this token (“ipxproto-resource-object”) and the token’s value (the name of the IPXPROTO process) are inserted in the message text.

`ZISA-EVT-EMSCOLL-UNAVAILABLE`

is the event number (1010). The DDL AS clause of this value (“ipxproto-unavailable”) appears in the message text following “event number:”.

`ZISA-VAL-CHRSN-`*`reason`*

is the change reason. One of the following DDL AS clauses is inserted in the message text following “cause:”:

Token Value	DDL AS Clause
ZISA-VAL-CHRSN-OPERATOR-RQST	“operatorrequest”
ZISA-VAL-CHRSN-INVALID-PROC	“invalid-processname”
ZISA-VAL-CHRSN-INCORR-PROC	“incorrect-processname”
ZISA-VAL-CHRSN-RESOURCE-PROB	“resource-problem”

`ZISA-VAL-STATE-CONNECTED`

is the previous state. The DDL AS clause of this value (“connected”) appears in the message text following “previous state:”.

`ZISA-VAL-STATE-DISCONNECTED`

is the current state. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-DISCONNECTED`

is the value assigned to the ZISA-TKN-OBJ-STATE token.

`ZISA-VAL-`*`reason`*

is one of the following values, assigned to the ZISA-TKN-OBJ-STATE-REASON token:

ZISA-VAL-OPERATOR-REQUEST
ZISA-VAL-INVALID-PROCESSNAME
ZISA-VAL-INCORRECT-PROCESSNAME
ZISA-VAL-RESOURCE-PROBLEM

Cause. The cause depends on the reason for the disconnection:

Change Reason	Cause
operatorrequest	The value of tdmNwSaIpxProtoState object for the IPXPROTO process was deliberately set to `disconnected`.
invalid-processname	A Set operation on tdmNwSaIpxProtoName specified a syntactically incorrect value.
incorrect-processname	A Set operation on tdmNwSaIpxProtoName specified a name for a process that is not an IPXPROTO process.
resource-problem	Resources needed to support communications between the subagent and the IPXPROTO process are not available.

Effect. The subagent cannot retrieve information from the IPXPROTO process.

Recovery. Correcting the situation depends on the reason for the disconnection:

Change Reason	Recovery
operatorrequest	No action is required because the stoppage was deliberate.
invalid-processname	Resubmit the Set request using the correct syntax for a NonStop Kernel process name.
incorrect-processname	Resubmit the Set request using the name of an existing IPXPROTO process.
resource-problem	Try to set the value of tdmNwSaIpxProtoState to `connected`. If the operation fails, refer to the *IPX/SPX Configuration and Management Manual* for ideas about how to handle unavailable IPXPROTO processes.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001010 Object
                              unavailable ipxproto-resource-object -
                              $ZNV0, event number: ipxproto-unavailable,
                              cause: incorrect-processname, previous
                              state: connected, current state:
                              disconnected, user content: undefined
```

1011: ZISA-EVT-MEMORY-ALLOC-FAILURE

The subagent could not allocate the memory required for an operation.

Unconditional Tokens / **Value**

```
ZISA-TKN-SUBAGENT          <1> IPX/SPX subagent process
ZEMS-TKN-EVENTNUMBER       name
ZEMS-TKN-TXFAULT-TYPE      <2> ZISA-EVT-MEMORY-ALLOC-
ZEMS-TKN-CONTENT-USER      FAILURE
ZISA-TKN-MEM-ALLOC-SIZE    <3> ZISA-VAL-TF-MEM
                           <4> undefined (null)
                               ZISA-TNM-MEM-ALLOC-SIZE
```

Conditional Tokens

```
None
```

Message Text

```
Transient Fault  IPX/SPX-SNMP-subagent-process - <1>, event
number:  <2>, fault type:  <3>, user content:  <4>
```

`ZISA-TKN-SUBAGENT`

identifies the subject of the event, always an IPX/SPX Subagent process. The DDL heading of this token (“IPX/SPX-SNMP-subagent-process”) and the token’s value (the name of the subagent process) are inserted in the message text.

`ZISA-EVT-MEMORY-ALLOC-FAILURE`

is the event number (1011). The DDL AS clause of this value (“memory-allocation-failure”) appears in the message text following “event number:”.

`ZISA-VAL-TF-MEM`

characterizes the type of transient fault that occurred. The DDL AS clause of this value (“Memory-allocation”) is inserted in the message text following “fault type:”.

`ZEMS-TKN-CONTENT-USER`

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-TNM-MEM-ALLOC-SIZE`

is the number of bytes the subagent tried to allocate.

Cause. The subagent could not allocate the memory needed to perform an operation. The subagent allocates memory for various operations, including responding to SNMP requests and adding a new entry to the tdmNwSaIPXPROTONamesTable, the tdmNwSaAgentNamesTable, and the tdmNwServerTable.

Effect. The operation could not be completed.

Recovery. Stop the subagent if necessary; try using another disk as a swap device. If the problem persists, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as EMS logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001011 Transient Fault
                              IPX/SPX-SNMP-subagent-process - \COMM.$ZISA,
                              event number: memory-allocation-failure,
                              fault type: Memory-allocation, user content:
                              undefined
```

1012: ZISA-EVT-PROCESS-TRAPPED

The subagent process trapped.

Unconditional Tokens	**Value**
ZISA-TKN-SUBAGENT	<1> *IPX/SPX subagent process name*
ZEMS-TKN-EVENTNUMBER	<2> ZISA-EVT-PROCESS-TRAPPED
ZEMS-TKN-TXFAULT-TYPE	<3> ZISA-VAL-TF-PROG
ZEMS-TKN-CONTENT-USER	<4> undefined (null)
ZEMS-TKN-CHANGE-REASON	ZISA-VAL-CR-*reason*

Conditional Tokens

None

Message Text

```
Transient Fault  IPX/SPX-SNMP-subagent-process - <1>, event
number:  <2>, fault type:  <3>, user content:  <4>
```

ZISA-TKN-SUBAGENT

identifies the subject of the event, always an IPX/SPX subagent process. The DDL heading of this token (“IPX/SPX-SNMP-subagent-process”) and the token’s value (the name of the subagent process) are inserted in the message text.

ZISA-EVT-PROCESS-TRAPPED

is the event number (1012). The DDL AS clause of this value (“process-trapped”) appears in the message text following “event number:”.

ZISA-VAL-TF-PROG

characterizes the type of transient fault that occurred. The DDL AS clause of this value (“Internal-error”) appears in the message text following “fault type:”.

ZEMS-TKN-CONTENT-USER

is assigned the value “undefined” in the message text following “user content:”.

`ZISA-VAL-CR-`*`reason`*

is the change reason, which is one of the following values:

Token Value	**DDL AS Clause**
ZISA-VAL-CR-PROCESS-SIGILL	“Instruction-failure”
ZISA-VAL-CR-PROCESS-SIGABRT	“Process-aborted”
ZISA-VAL-CR-PROCESS-SIGFPE	“Arithmetic-overflow”
ZISA-VAL-CR-PROCESS-SIGSEGV	“Illegal-address-reference”
ZISA-VAL-CR-PROCESS-SIGSTK	“Stack-overflow”
ZISA-VAL-CR-PROCESS-SIGTIMEOUT	“Loop-timer-timeout”
ZISA-VAL-CR-PROCESS-SIGTERM	“Termination-request”
ZISA-VAL-CR-PROCESS-SIGNOMEM	“Memory-space-exhausted”

Cause. The subagent process trapped.

Effect. The subagent process stops.

Recovery. If the trap occurred in the primary process, any existing backup process takes over, so no recovery is needed. If a backup process was not running, try restarting the subagent. Regardless of whether the subagent is able to recover, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as EMS logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Sample Message

```
96-01-30 16:50:50 \COMM.$ZISA TANDEM.IPXSA.D30 001012 Transient Fault
                              IPX/SPX-SNMP-subagent-process - \COMM.$ZISA,
                              event number:process-trapped,
                              fault type: Internal-error, user content:
                              undefined
```

14 Ethernet Subagent

This section describes the Ethernet Subagent and the MIBs it supports.The Ethernet Subagent allows Ethernet resources on NonStop systems to be monitored by SNMP managers. The Ethernet Subagent provides statistics about Ethernet interfaces (E4SA, GESA, and G4SA interfaces) with the ServerNet LAN System Access (SLSA) subsystem, which supports parallel LAN I/O in a G-series ServerNet-based system. The Ethernet Subagent also supports additional private MIB objects defined by HP that let you monitor and manage the subagent itself.

Architectural Overview

The Ethernet Subagent supports management of Ethernet resources and the subagent in the following ways:

- The Ethernet Subagent implements the dot3StatsTable, defined in RFC 1643, which provides statistics about Ethernet-like interface types and which is part of the dot3 group within the MIB-II Transmission group.
- Additional private objects in the Ethernet Subagent's MIB defined by HP allow the Ethernet Subagent to be monitored and managed through SNMP.

The interaction between the SNMP agent, the Ethernet Subagent, and the TCP/IP subsystem whose Ethernet interfaces are being monitored is illustrated in Figure 14-1, Architectural Overview of the Ethernet Subagent, on page 14-2.

The Ethernet Subagent and Its Managed Resources

Each Ethernet Subagent monitors the Ethernet interfaces for one HP TCP/IP subsystem. When you start the Ethernet Subagent, you define:

- The TCP/IP subsystem whose Ethernet interfaces are to be monitored
- The SCP process with which the Ethernet Subagent communicates

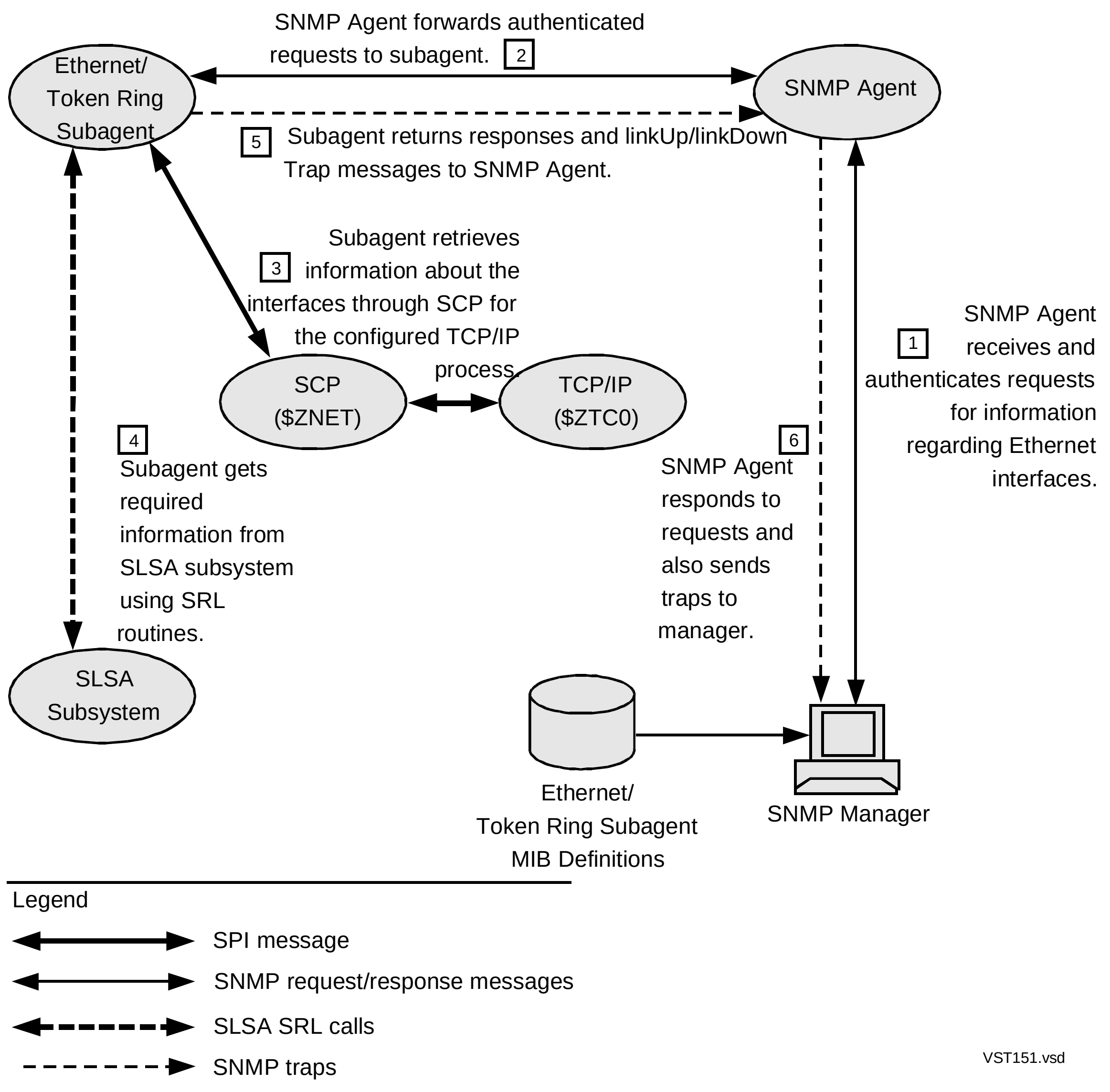
By default, the Ethernet Subagent monitors the TCP/IP process $ZTC0 running on the local node, and communicates through the SCP process $ZNET.

The Ethernet Subagent and the SNMP Agent

Any SNMP agent process can communicate with only one Ethernet Subagent process. Because each Ethernet Subagent can monitor only one TCP/IP subsystem, you must start one SNMP agent/Ethernet Subagent pair for each TCP/IP subsystem whose Ethernet interfaces you want to monitor.

You define the SNMP agent with which the Ethernet Subagent communicates when you start the subagent. By default, the Ethernet Subagent communicates with the SNMP agent $ZSNMP on the same node as the subagent.

Figure 14-1. Architectural Overview of the Ethernet Subagent


SNMP Agent forwards authenticated requests to subagent. 2
Ethernet/ Token Ring Subagent
SNMP Agent
5 Subagent returns responses and linkUp/linkDown Trap messages to SNMP Agent.
Subagent retrieves 3 information about the interfaces through SCP for the configured TCP/IP process.
SNMP Agent 1 receives and authenticates requests for information regarding Ethernet interfaces.
SCP ($ZNET)
TCP/IP ($ZTC0)
4 Subagent gets required information from SLSA subsystem using SRL routines.
6 SNMP Agent responds to requests and also sends traps to manager.
SLSA Subsystem
Ethernet/ Token Ring Subagent MIB Definitions
SNMP Manager
Legend
SPI message
SNMP request/response messages
SLSA SRL calls
SNMP traps
VST151.vsd

dot3 Statistics Group Supported by the Ethernet Subagent

The Ethernet Subagent supports the dot3 Statistics group for Ethernet-like Interface types, defined in RFC 1643, identified by a check mark in the following list:

```
iso (1)
      identified-organization (3)
        dod (6)
          internet (1)
            mgmt (2)
              mib-II (1)
                system (1)
                interfaces (2)
                at (3)
                ip (4)
                icmp (5)
                tcp (6)
                udp (7)
                egp (8)
                cmot (9)
                transmission (10)
                  .
                  .
                  dot3 (7)  √
                  .
                  .
                snmp (11)
```

The Ethernet Subagent MIB

The Ethernet Subagent MIB supports private MIB objects that:

- Provide information about the subagent process itself, such as:
  - The name of the subagent process
  - The processor in which the subagent’s primary process is running
- Allow you to control certain aspects of the subagent’s behavior, such as:
  - Causing a backup process to take over and a new backup process to be created
  - Controlling the rate at which MIB object values are updated
  - Requesting that the subagent refresh MIB object values on demand

These private MIB objects are defined by HP and reside in the zesa subtree within the nonstopsystems subtree registered to HP:

```
iso (1)
   org (3)
      dod (6)
         internet (1)
            private (4)
               enterprises (1)
                  tandem (169)
                     nonstopsystems (3)
                        zesa (217)                  <— Ethernet Subagent
                           zesaAgentName (1)            process objects
                           zesaAgentState (2)
                           zesaProcessName (3)
                           zesaProcessVersion (4)
                           zesaProgramFile (5)
                           zesaProcessPriority (6)
                           zesaProcessGroupID (7)
                           zesaProcessUserID (8)
                           zesaPrimaryCPU (9)
                           zesaPrimaryPIN (10)
                           zesaBackupCPU (11)
                           zesaBackupPIN (12)
                           zesaBackupState (13)
                           zesaSwitchtoBackupNow (14)
                           zesaEmsCollectorName (15)
                           zesaEmsCollectorState (16)
                           zesaKeepAliveTimer (17)
                           zesaRefreshNow (18)
                           zesaRefreshTimer (19)
                           zesaTcpIpProcess (20)
                           zesaTcpIpState (21)
                           zesaSCPProcess (22)
                           zesaTrap (23)
```

The zesa objects supported by the Ethernet Subagent are described in detail in Table 14-3 on page 14-12.

Installing the Ethernet Subagent

Run the Distributed Systems Management/Software Configuration Manager (DSM/SCM) product to install the SNMP agent and the Ethernet Subagent.

Then load the ASN.1 source code for the Ethernet Subagent's private MIB onto any SNMP manager station from which you want to monitor Ethernet resources. The MIB definitions are contained in two files:

- RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*, contains definitions for the dot3 Statistics group MIB objects.
- The ZESAMIB file contains definitions for the private Ethernet Subagent MIB objects defined by HP.

Compile the ZESAMIB file (and, if necessary, the RFC1643 file) as described in the documentation provided with your SNMP manager. Compiling the MIB makes it possible for the SNMP manager to display the names (rather than only numeric object identifiers) of MIB objects.

Once the MIB definition files have been compiled, you can start the Ethernet Subagent.

Before Starting the Ethernet Subagent

The SNMP agent must be running before you start the Ethernet Subagent. Start the SNMP agent with the RUN command, as described in Section 2, Installing and Configuring the SNMP Agent. For example:

```
RUN SNMPAGT /NAME $agent-process, NOWAIT/
```

If the SNMP agent is not available when the Ethernet Subagent is started, the subagent fails to start. If the SNMP agent disappears after the Ethernet Subagent starts, the subagent enters a loop trying to reconnect to the SNMP agent at the rate specified in its zesakeepAliveTimer object.

In addition, the SCP process ($ZNET, or the SCP process specified for the -s startup parameter) must be running. If the SCP process is not available, the Ethernet Subagent enters a loop trying to reconnect to it at the rate specified in its zesaKeepAliveTimer object.

Note. You can access the Ethernet Subagent's private MIB objects regardless of the status of the SCP process.

Starting the Ethernet Subagent

Start the Ethernet Subagent using the TACL RUN command. Be sure to specify the name of a running SNMP agent. Following is an example of starting an Ethernet Subagent named $ZESA, with a backup process running in processor 6, that communicates with SNMP agent $ZSNMP:

```
RUN ETHSA /NAME $ZESA, NOWAIT/ -a $ZSNMP -b 6
```

Syntax of the startup options follows established conventions used in the Guardian environment.

```
[RUN] [[$volume.]subvolume.]ETHSA
   / NAME $eth-tr-subagent-process [,other-run-option]... /
   [ startup-parameter [startup-parameter]...]
```

volume

identifies the volume on which the Ethernet Subagent program file (ETHSA) resides. You can omit it if ETHSA resides on your current subvolume. By default, the DSM/SCM program puts ETHSA into $SYSTEM.

subvolume

identifies the subvolume on which ETHSA resides. You can omit it if it is named in your TACL #PMSEARCHLIST. By default, the DSM/SCM program puts ETHSA into SYSTEM.

eth-tr-subagent-process

identifies the Ethernet Subagent process. You can specify from one through five alphanumeric characters, but the first character must be alphabetic.

other-run-option

is any of the TACL RUN command options. Refer to the *TACL Reference Manual* for more information about these options. HP recommends using at least the NOWAIT option so that you can resume TACL operations once the subagent is started.

startup-parameter

is one of the parameters listed next with which you can control attributes of the Ethernet Subagent process.

Note. Each of the Ethernet Subagent startup parameters corresponds to one of its private (zesa) MIB objects, as indicated in the following syntax diagram and described in Configuring a Running Ethernet Subagent on page 14-9.

```
-a [\node.]$agent-process          | zesaAgentName
-b backup-cpu-number               | zesaBackupCPU
-c [\node.]$collector-process      | zesaEmsCollectorName
-k keep-alive-timer                | zesaKeepAliveTimer
-r refresh-timer                   | zesaRefreshTimer
-s $scp-process                    | zesaScpProcess
-t $tcpip-process                  | zesaTcpIpProcess
```

-a [\node.]$*agent-process*

is the name of the SNMP agent process with which you want the Ethernet Subagent to communicate.

The default value is $ZSNMP on the same node as the subagent.

-b *backup-cpu-number*

is the processor number in which a backup process is to be created.

The default value is -1, indicating that no backup is to be created.

-c [\node.]$*collector-process*

is the name of the collector to which the subagent is to send EMS event messages it generates.

The default is $0, the primary EMS collector, on the local node.

Note. EMS event generation is always enabled at process startup. To disable the sending of events, set the value of the zesaEmsCollectorState object to disabled (3) once the Ethernet Subagent is running.

-k *keep-alive-timer*

specifies the time interval, in seconds, between attempts by the Ethernet Subagent to reconnect to a TCP/IP, SNMP agent, backup, or EMS collector process that is in the disconnected state.

The default value is 60.

Note. This timer value does not apply to objects that are placed in the disabled state by an SNMP Set request. See State Object/Resource Object Pairs on page 14-16 for information on the possible states that can be set for processes used by the Ethernet Subagent.

`-r` *`refresh-timer`*

specifies, in seconds, how often the subagent should refresh its cached values for the MIB.

The default is 60 seconds.

`-s $`*`scp-process`*

is the name of the SCP process to be used for SPI requests for deriving MIB object values. The SCP process named must be running and on the same node as the subagent process.

The default is $ZNET.

`-t $`*`tcpip-process`*

is the name of the TCP/IP process whose Ethernet interfaces are to be monitored by the subagent. The subagent communicates with the SLSA subsystem using Shared Resource Library (SRL) calls for each interface to get statistics.

The default is $ZTC0.

Stopping the Ethernet Subagent

Issue a TACL STOP command to stop an Ethernet Subagent process. For example:

```
STOP $ZESA
```

Initiating Backup Process Takeover

The Ethernet Subagent can be run as a process pair to achieve basic fault-tolerance. The support includes a persistent subagent process and checkpointing of the startup parameters.

To run the Ethernet Subagent as a process pair and have the backup process take over in case the primary process fails, include the -b startup parameter when you start the subagent. For example:

```
RUN ETHSA /NAME $ZESA, NOWAIT/ -b 6
```

Note. You must specify a backup CPU. Otherwise the Ethernet Subagent does not run as a process pair.

You can force the backup process to take over and a new backup process to be created by setting the value of the zesaSwitchToBackupNow object to 1 (forceBackupTakeover) from an SNMP manager.

Configuring a Running Ethernet Subagent

Once an Ethernet Subagent is running, you can issue SNMP Get and Set requests from an SNMP manager to query and control the subagent.

Querying a Running Ethernet Subagent

You can find out the information listed in Table 14-1 about a running Ethernet Subagent by issuing SNMP Get requests from an SNMP manager.

Table 14-1. Querying an Ethernet Subagent Through SNMP

You can find out this about the Ethernet Subagent...	**By issuing an SNMP Get request against this zesa object...**
SNMP agent process with which subagent is communicating	zesaAgentName
State of the connection between subagent and SNMP agent	zesaAgentState
Name of subagent process	zesaProcessName
Version of subagent process object file	zesaProcessVersion
Name of subagent program file	zesaProgramFile
Current process priority of subagent process	zesaProcessPriority
Group ID under which subagent was started	zesaProcessGroupID
User ID under which subagent was started	zesaProcessUserID
Processor in which the subagent's primary process is running	zesaPrimaryCPU
Process number of subagent's primary process	zesaPrimaryPIN
Processor in which subagent's backup process is running or will start	zesaBackupCPU
Process number of subagent's backup process	zesaBackupPIN
State of subagent's backup process	zesaBackupState
EMS collector process to which subagent sends events	zesaEmsCollectorName
State of the connection to the EMS collector	zesaEmsCollectorState
Rate at which subagent makes reconnection attempts	zesaKeepAliveTimer
Rate at which subagent updates its cached values for MIB objects	zesaRefreshTimer
SCP process used for SPI requests	zesaScpProcess
TCP/IP process whose Ethernet interfaces are being monitored	zesaTcpIpProcess
State of the connection between subagent and monitored TCP/IP process	zesaTcpIpState

These objects are described in detail in Table 14-3 on page 14-12.

Controlling a Running Ethernet Subagent

In addition to getting information about a running Ethernet Subagent, you can control certain aspects of its behavior by issuing SNMP Set requests from an SNMP manager, as described in Table 14-2 on page 14-10.

All subagent attributes that can be controlled by setting MIB object values are reinitialized when the subagent is restarted to default values or values specified in startup parameters.

Table 14-2. Controlling an Ethernet Subagent Through SNMP (page 1 of 2)

You can control this attribute of an Ethernet Subagent...	**By issuing SNMP Set requests against this object...**	**Initially set with this startup parameter...**	**With this default value...**
Current process priority of subagent process	zesaProcessPriority		None; determined by operating system.
Processor in which subagent's backup process is running or will start	zesaBackupCPU	-b	-1 (no backup process) The zesaBackupState* object must be set to disabled (3) before the zesaBackupCPU object can be set.
State of the subagent's backup process	zesaBackupState		None.
Switching to the backup subagent process	zesaSwitchToBackupNow		0 (autoProcessPairControl) at startup.
EMS collector process to which subagent sends events	zesaEmsCollectorName	-c	$0 on the same node as the subagent. zesaEmsCollectorState* must be set to disabled (3) before the zesaEmsCollectorName object can be set.
State of the connection to the EMS collector	zesaEmsCollectorState		None; in normal circumstances, value is "connected."

* See State Object/Resource Object Pairs on page 14-16 for information about Ethernet Subagent objects that control or indicate the internal processing state of processes used by the subagent.

Table 14-2. Controlling an Ethernet Subagent Through SNMP (page 2 of 2)

You can control this attribute of an Ethernet Subagent...	**By issuing SNMP Set requests against this object...**	**Initially set with this startup parameter...**	**With this default value...**
Rate at which subagent makes reconnection attempts	zesaKeepAliveTimer	-k	60 seconds between attempts made by the subagent to connect to a SNMP agent, TCP/IP, or to restart the backup process.
Cause subagent to update MIB values in cache	zesaRefreshNow		0 (autoDynamicRefresh) at startup.
Rate at which cached values for MIB objects are refreshed	zesaRefreshTimer	-r	60 seconds.
TCP/IP process whose Ethernet interfaces are being monitored	zesaTcpIpProcess	-t	$ZTC0 on the same node as the subagent. zesaTcpIpState* must be set to disabled (3) before the zesaTcpIpProcess object can be set.
State of the connection to the monitored TCP/IP process	zesaTcpIpState		None: in normal circumstances, value is “connected.”

* See State Object/Resource Object Pairs on page 14-16 for information about Ethernet Subagent objects that control or indicate the internal processing state of processes used by the subagent.

These read-write objects are described in detail in Table 14-3 on page 14-12.

ZESA MIB Objects

The collection of scalar objects in the zesa subtree, described in Table 14-3, provide information about the Ethernet Subagent and allow you to control some aspects of its behavior.

Table 14-3. Private (ZESA) MIB Objects Supported by the Ethernet Subagent (page 1 of 5)

Object and Attributes	Definition	Format and Derivation of Value
zesaAgentName 1.3.6.1.4.1.169.3.217.1 read-only DisplayString (SIZE (0..32))	The name of the SNMP agent process with which the Ethernet Subagent is communicating.	\node.$*agent-process* specified for the -a startup parameter. If none specified, $ZSNMP on the local node.
zesaAgentState 1.3.6.1.4.1.169.3.217.2 read-only INTEGER	The status of the interface between the Ethernet Subagent and the SNMP agent.	Following are valid states for the subagent process: connected (1) disconnected (2) A value of connected (1) is always returned in a response message to a management request, indicating that the Ethernet Subagent is communicating with the SNMP agent.
zesaProcessName 1.3.6.1.4.1.169.3.217.3 read-only DisplayString (SIZE (0..49))	The fully qualified NonStop Kernel process name of this subagent.	\node.$*process:seq-no* identifying the process name specified for the TACL NAME option when the Ethernet Subagent was started, and the sequence number assigned to the process by the NonStop Kernel.
zesaProcessVersion 1.3.6.1.4.1.169.3.217.4 read-only DisplayString (SIZE (0..32))	The version of the Ethernet Subagent process object file.	T0326G*nn_date* -- HP SNMP Ethernet Subagent, where G*nn* is the product version of the Ethernet Subagent, for example, G06.
zesaProgramFile 1.3.6.1.4.1.169.3.217.5 read-only DisplayString (SIZE (0..32))	The name of the Ethernet Subagent program file.	\node.$*vol.subvol.filename*.
zesaProcessPriority 1.3.6.1.4.1.169.3.217.6 read-only INTEGER (0..199)	The current NonStop Kernel process priority of the Ethernet Subagent process.	Initially determined by the NonStop Kernel or from a TACL run option. After startup, the value can be set by an SNMP manager.
zesaProcessGroupID 1.3.6.1.4.1.169.3.217.7 read-only INTEGER (0..255)	The NonStop Kernel process group ID part of the process access ID (PAID) for the Ethernet Subagent process.	*group-ID-number*

Table 14-3. Private (ZESA) MIB Objects Supported by the Ethernet Subagent (page 2 of 5)

Object and Attributes	Definition	Format and Derivation of Value
zesaProcessUserID 1.3.6.1.4.1.169.3.217.8 read-only INTEGER (0..255)	The NonStop Kernel process user ID part of the process access ID (PAID) for the Ethernet Subagent process.	*`user-ID-number`*
zesaPrimaryCPU 1.3.6.1.4.1.169.3.217.9 read-only INTEGER (0..255)	The number of the processor in which the primary Ethernet Subagent process is running.	*`cpu-number`*
zesaPrimaryPIN 1.3.6.1.4.1.169.3.217.10 read-only INTEGER (0..255)	The process identification number (PIN) of the primary Ethernet Subagent process.	*`PIN-number`*
zesaBackupCPU 1.3.6.1.4.1.169.3.217.11 read-write INTEGER (0..255)	The number of the processor in which the backup process will start or is running.	*`cpu-number`* specified for the -b startup parameter. After startup, the value can be set by an SNMP manager. A value of -1 indicates the backup is not configured. Setting this object is permitted only if zesaBackupState is set to disabled (3). Once zesaBackupCPU has been set to a new value, zesaBackupState must be set to enabled (4) for the new value to take effect.
zesaBackupPIN 1.3.6.1.4.1.169.3.217.12 read-only INTEGER (0..255)	The process identification number (PIN) of the backup Ethernet Subagent process.	*`PIN-number`* A value of -1 indicates that the backup is not running.
zesaBackupState 1.3.6.1.4.1.169.3.217.13 read-write INTEGER	The state of the Ethernet Subagent backup process.	Following are valid states for the backup process: connected (1) disconnected (2) disabled (3) enabled (4) This value can be set with an SNMP request as illustrated by the state change diagram in Figure 14-2 on page 14-18.

Table 14-3. Private (ZESA) MIB Objects Supported by the Ethernet Subagent (page 3 of 5)

Object and Attributes	Definition	Format and Derivation of Value
zesaSwitchToBackupNow 1.3.6.1.4.1.169.3.217.14 read-write INTEGER	Causes the Ethernet Subagent backup process to take over and a new backup process to be created.	One of these values: autoProcessPairControl (0) forceBackupTakeover (1) At startup, the subagent sets this value to 0, but it can be changed by an SNMP manager if the Ethernet Subagent was started using the -b startup parameter. Setting this object to 1 causes the Ethernet Subagent backup process to take over and a new backup process to be created. Setting this object is permitted only if zesaBackupState is connected (1).
zesaEmsCollectorName 1.3.6.1.4.1.169.3.217.15 read-write DisplayString (SIZE (0..15))	The name of the EMS collector process to which the Ethernet Subagent sends events it generates.	\node.$*ems-collector* specified for the -c startup parameter. After startup, the value can be set by an SNMP manager. Setting this object is permitted only if zesaEmsCollectorState is disabled (3).
zesaEmsCollectorState 1.3.6.1.4.1.169.3.217.16 read-write INTEGER	The state of the connection between the Ethernet Subagent and the EMS collector.	Following are valid values for the state of the connection: connected (1) disconnected (2) disabled (3) enabled (4) The connected and disconnected states are set internally. The disabled and enabled states are set with an SNMP request as illustrated by the state change diagram in Figure 14-2 on page 14-18.
zesaKeepAliveTimer 1.3.6.1.4.1.169.3.217.17 read-write INTEGER (0..32000)	The number of seconds between attempts the subagent makes to connect to a lost resource or to restart the backup process.	A value in the range 0 to 32000 specified for the -k startup parameter. After startup, the value can be set by an SNMP manager. The default is 60 seconds.

Table 14-3. Private (ZESA) MIB Objects Supported by the Ethernet Subagent (page 4 of 5)

Object and Attributes	Definition	Format and Derivation of Value
zesaRefreshNow 1.3.6.1.4.1.169.3.217.18 read-write INTEGER	A means of forcing an update of the MIB tables and generating traps (if needed).	One of these values: autoDynamicRefresh (0) forceDynamicRefresh (1) This object normally reads 0, indicating that the MIB values are refreshed at the rate indicated by the value of the zesaRefreshTImer Setting this object to 1 causes the subagent to update the MIB tables and generate traps (if needed). The object returns a value of 1 until the refresh is complete.
zesaRefreshTimer 1.3.6.1.4.1.169.3.217.19 read-write INTEGER (0..32000)	The cache update interval (in seconds) for MIB objects.	A value in the range 0 to 32000 specified for the -r startup parameter. The default is 60 seconds. After startup, the value can be set by an SNMP manager. Value 0 disables cache timing so that updates occur to objects only when zesaRefreshNow is set to forceDynamicRefresh (1).
zesaTcpIpProcess 1.3.6.1.4.1.169.3.217.20 read-write DisplayString (SIZE (0..15))	The name of the TCP/IP process whose Ethernet interfaces are being monitored by the subagent.	$*tcpip-process* specified for the -t startup parameter. The default is $ZTC0. After startup, the value can be set by an SNMP manager. Setting this object is permitted only if zesaTcpIpState is disabled (3).

Table 14-3. Private (ZESA) MIB Objects Supported by the Ethernet Subagent (page 5 of 5)

Object and Attributes	Definition	Format and Derivation of Value
zesaTcpIpState 1.3.6.1.4.1.169.3.217.21 read-write INTEGER	The state of the connection between the Ethernet Subagent and the TCP/IP process whose Ethernet interfaces are being monitored.	The settable values for this object are: disabled (3) enabled (4) The following states are set by the subagent to reflect the running state of the TCP/IP resource interface: connected (1) disconnected (2)
zesaSCPProcess 1.3.6.1.4.1.169.3.217.22 read-only DisplayString (SIZE (0..15))	The name of the SCP process used for SPI requests.	$*scp-process* specified for the -s startup parameter. The default is $ZNET.
zesaTrap 1.3.6.1.4.1.169.3.217.23 read-only DisplayString (SIZE (0..25))	Used internally while generating traps.	N.A.

State Object/Resource Object Pairs

The private Ethernet Subagent MIB objects that control or indicate the internal processing state of other objects are referred to as “state objects.” The objects whose internal processing states are indicated or controlled by state objects are referred to as “resource objects.”

Table 14-4 lists the resource objects and their associated state objects and state values.

Table 14-4. Ethernet Subagent State Object/Resource Object Pairs

The operational state of this resource object...	Is indicated or controlled by this state object...	Which may contain one of the following states...
zesaAgentName	zesaAgentState	Connected (1) Disconnected (2)
zesaBackupCPU zesaBackupPIN	zesaBackupState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)
zesaTcpIpProcess	zesaTcpIpState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)
zesaEmsCollectorName	zesaEmsCollectorState	Connected (1) Disconnected (2) Disabled (3) Enabled (4)

State objects cannot be set to “Connected” and “Disconnected” with a Set request; these objects are internally generated states indicating the state of the connection between the Ethernet Subagent process and the process represented by the resource object. “Disabled” and “Enabled” are SNMP Set request-generated states used when resource objects are being reconfigured.

The meanings of each state is as follows:

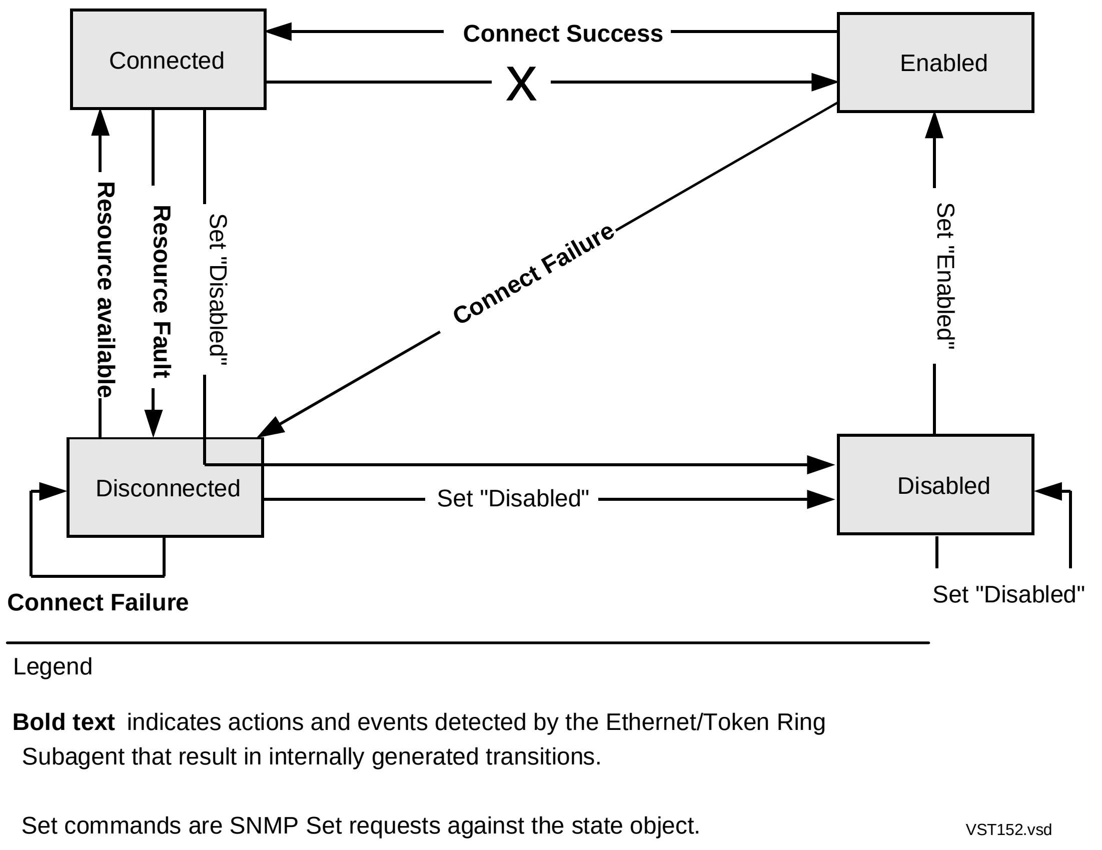
Connected	The subagent is functioning normally.
Disconnected	Subagent function dependent on the process or interface is not available.
Enabled	The subagent will immediately attempt to establish connections with the process or interface.
Disabled	The process or interface has been taken out of service explicitly by an operator.

State Transitions for Resource Objects

Figure 14-2 illustrates how resource objects move between states. Note that:

- Setting a “Connected” object to “Disabled” forces a transition through the “Disconnected” state. This permits the orderly shutdown of the resource and the generation of a state change event for the disconnected state.
- “Connect Success,” “Connect Failure,” and “Resource Fault” all relate to actions and events detected by the subagent. Internally generated transitions produce an EMS event indicating the state change, as described in Event Message Descriptions on page 14-33.
- An SNMP general error is returned if the Ethernet Subagent receives a Set “Enabled” request against a state object that is in the “Connected” state.

Figure 14-2. Ethernet Subagent Resource Object States


Connected
Connect Success
X
Enabled
Resource available
Resource Fault
Set "Disabled"
Connect Failure
Set "Enabled"
Disconnected
Set "Disabled"
Disabled
Connect Failure
Set "Disabled"
Legend
Bold text indicates actions and events detected by the Ethernet/Token Ring Subagent that result in internally generated transitions.
Set commands are SNMP Set requests against the state object.
VST152.vsd

Modifying the Value of a Resource Object

To modify the value of a resource object:

1. Set the value of the resource object’s associated state object to “Disabled.”
2. Set the resource object to a new value.

3. Set the value of the resource object’s associated state object to “Enabled.”

When the state object has been placed in the “Enabled” state, the Ethernet Subagent attempts to connect to the underlying process.

- If the connection attempt is successful, the object enters the “Connected” state.
- If the connection attempt fails, the object enters the “Disconnected” state. The Ethernet Subagent continues to attempt to reconnect at the rate specified for the zesaKeepAliveTimer object.

Ethernet-Like Statistics dot3Group

The Ethernet Subagent supports the dot3StatsTable within the dot3 Ethernet-like Statistics group, as defined in RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types.*

```
iso (1)
    identified-organization (3)
      dod (6)
        internet (1)
          mgmt (2)
            mib-II (1)
              .
              .
              transmission (10)
                .
                .
                dot3 (7)
                  dot3StatsTable (2)
                    dot3StatsEntry (1)
                      dot3StatsIndex (1)
                      dot3StatsAlighmentErrors (2)
                      dot3StatsFCSErrors (3)
                      dot3StatsSingleCollisionFrames (4)
                      dot3StatsMultipleCollisionFrames (5)
                      dot3StatsSQETestErrors (6)
                      dot3StatsDeferredTransmissions (7)
                      dot3StatsLateCollisions (8)
                      dot3StatsExcessiveCollisions (9)
                      dot3StatsInternalMacTransmitErrors (10)
                      dot3StatsCarrierSenseErrors (11)
                      dot3StatsFrameTooLongs (13)
                      dot3StatsInternalMacReceiveErrors (16)
                      dot3StatsEtherChipSet (17)
```

MIB Objects

Table 14-5 describes how the Ethernet Subagent supports objects in the dot3StatsTable.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 1 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsTable Objects:	Entries describing the Ethernet interfaces of the TCP/IP resources being monitored.	Refer to dot3StatsTable Maintenance on page 14-28 for information on how entries are created and updated.
dot3StatsIndex 1.3.6.1.2.1.10.7.2.1 read-only INTEGER	An index value that uniquely identifies an interface to an ethernet-like medium. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex for the ifTable that is managed by the TCP/IP Subagent.	This object is supported for E4SA, GESA. and G4SA. It is an integer ranging from 1 to the number of interfaces associated with the TCP/IP subsystem. This value is updated as each interface associated with the TCP/IP subsystem is obtained using the SPI procedure calls.
dot3StatsAlignmentErrors 1.3.6.1.2.1.10.7.2.2 read-only Counter	Number of frames received on a particular interface that are not an integral number of octets in length.	This value is supported for E4SA, GESA, and G4SA. It is a value obtained from: ● the ‘RXalec’ field of the structure LME4SATypeStatsType for E4SA, ● the dot3StatsAlignmentErrors field of the structure LMCLGESARxStatsType for GESA ● the alignmentErrors field of the structure LMCLG4SARxStatsType for G4SA.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 2 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsFCSErrors 1.3.6.1.2.1.10.7.2.3 read-only Counter	Number of frames discarded because of frame checksum errors.	This object is supported for E4SA, GESA and the G4SA. It is a value obtained from: ● the 'RXcrec' field of the structure LME4SATypeStatsType for E4SA ● the dot3StatsFCSErrors field of the structure LMCLGESARxStatsType for GESA ● the FCSErrors field of the structure LMCLG4SARxStatsType for G4SA.
dot3StatsSingleCollisionFrames 1.3.6.1.2.1.10.7.2.4 read-only Counter	Number of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision.	This object is supported for GESA and G4SA, but not for E4SA. The value is obtained from: ● the dot3StatsSingle CollisionFrames field of the structure LMCLGESATxStatsType for GESA ● the SingleCollisionFrames field of the structure LMCLG4SATxStatsType for G4SA. For E4SA, the value 0 is returned when this object is accessed.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 3 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsMultipleCollision Frames 1.3.6.1.2.1.10.7.2.5 read-only Counter	The number of succesfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision.	This object is supported for GESA and G4SA, but not for E4SA. The value is obtained from: ● the dot3StatsMultipleCollisionFrames field of the structure LMCLGESATxStatsType ● the multipleCollisionFrames field of the structure LMCLG4SATxStatsType for G4SA. For E4SA, the value 0 is returned when this object is accessed.
dot3StatsSQETestErrors 1.3.6.1.2.1.10.7.2.6 read-only Counter	Number of times a heartbeat error occurred.	This object is supported for E4SA and G4SA, but not for GESA. The value is obtained from: ● the ‘TXheartbeat’ field of the structure LME4SATypeStatsType for E4SA ● the SQE field of the structure LMCLG4SATxStatsType for G4SA.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 4 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsDeferredTransmissions 1.3.6.1.2.1.10.7.2.7 read-only Counter	Number of frames that could not be transmitted immediately due to bus activity.	This object is supported for E4SA, GESA, and G4SA. The value is obtained from: ● the 'TXdefer' field of the structure LME4SATypeStatsType for E4SA ● the dot3StatsDeferredTransmissions field of the structure LMCLGESATxStatsType for GESA ● the field deferredTransmissions of the structure LMCLG4SATxStatsType for G4SA.
dot3StatsLateCollisions 1.3.6.1.2.1.10.7.2.8 read-only Counter	Number of frames that had their transmission aborted because a collision occurred before completion of the retry process.	This object is supported for E4SA, GESA, and G4SA. The value is obtained from: ● the 'TXlatecol' field of the structure LME4SATypeStatsType for E4SA ● the dot3StatsLateCollisions field of the structure LMCLGESATxStatsType for GESA ● the lateCollisions field of the structure LMCLG4SATxStatsType for G4SA.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 5 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsExcessiveCollisions 1.3.6.1.2.1.10.7.2.9 read-only Counter	Number of frames that had their transmission aborted after 15 collisions.	This object is supported for the E4SA, GESA, and G4SA. The value is obtained from: ● the 'TXcolfail' field of the structure LME4SATypeStatsType for E4SA ● the dot3StatsExcessiveCollisions field of the structure LMCLGESATxStatsType for GESA ● the excessiveCollisions field of the structure LMCLG4SATxStatsType for G4SA.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 6 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsInternalMacTransmit Errors 1.3.6.1.2.1.10.7.2.10 read-only Counter	Number of frames for which transmission on a particular interface fails due to an internal MAC sublayer tansmit error.	This object is supported for GESA and G4SA, but not for E4SA. The value is obtained from: ● the dot3StatsInternalMacTransmitErrors field of the structure LCMLGESATxStatsType for GESA ● the field transmitError of the structure LMCLG4SATxStatsType for G4SA. For E4SA, the value 0 is returned when this object is accessed.
dot3StatsCarrierSenseErrors 1.3.6.1.2.1.10.7.2.11 read-only Counter	Number of frames that had their transmission aborted because carrier sense was lost.	This object is supported for E4SA, GESA, and G4SA. The value is obtained from ● the 'TXcslost' field of the structure LME4SATypeStatsType for E4SA ● the dot3StatsCarrierSenseErrors field of the structure LMCLGESATxStatsType for GESA ● the carrierSenseErrors field of the structure LMCLG4SATxStatsType for G4SA.

Table 14-5. dot3StatsTable Objects Supported by Ethernet Subagent (page 7 of 7)

Object and Attributes	Definition	Format and Derivation of Value
dot3StatsFrameTooLong 1.3.6.1.2.1.10.7.2.13 read-only Counter	Number of frames discarded because the length of the frame was greater than the maximum frame size.	This object is supported for E4SA, GESA, and G4SA. The value is obtained from ● the 'RX2big' field of the structure LME4SATypeStatsType for E4SA, ● the dot3StatsFrameTooLong field of the structure LMCLGESARxStatsType for GESA ● the pktTooLong field of the structure LMCLG4SARxStatsType for G4SA.

The Ethernet SubAgent returns only 32-bit values for GESA and G4SA interface statistics, though the SLSA Management library supports64-bit values for GESA and G4SA statistics. Normally, the chances of GESA or G4SA statistics exceeding 32-bit values are minimal.

RFC Compliance

Table 14-6 summarizes compliance of Interfaces group support with RFC 1643.

Table 14-6. Compliance With dot3 Group Definitions in RFC 1643

Object Descriptor	Compliance	Explanation
dot3StatsIndex	Yes	See Table 14-5.
dot3StatsAlignmentErrors	Yes	See Table 14-5.
dot3StatsFCSErrors	Yes	See Table 14-5.
dot3StatsSingleCollisionFrames	Partial	See Table 14-5 (dot3StatsSingleCollisionFrames is supported for GESA, E4SA, and G4SA.)
dot3StatsMultipleCollisionFrames	Partial	See Table 14-5 (dot3statsMultipleCollisionFrames is supported for GESA and G4SA.)
dot3StatsSQETestErrors	Partial	See Table 14-5. (dot3StatsSQETestErrors is supported for E4SA and G4SA.)
dot3StatsDeferredTransmissions	Yes	See Table 14-5.
dot3StatsLateCollisions	Yes	See Table 14-5.
dot3StatsExessiveCollisions	Yes	See Table 14-5.
dot3StatsInternalMacTransmitErrors	Partial	See Table 14-5.
dot3StatsCarrierSenseErrors	Yes	See Table 14-5.
dot3StatsFrameTooLongs	Yes	See Table 14-5.
dot3StatsInternalMacReceiveErrors	Partial	No corresponding HP instrumentation exists; 0 is returned when the object is accessed. (dot3StatsInternalMacReceiveErrors is not supported for E4SA, GESA and G4SA interfaces.)
dot3StatsEtherChipSet	Partial	No corresponding HP instrumentation exists; NULL is returned when the object is accessed. (dot3StatsEtherChipSet is not supported for E4SA, GESA and G4SA Interfaces.)

dot3StatsTable Maintenance

For each subnet found in response to an INFO SPI command for the SUBNET * object of the TCP/IP process, the Ethernet Subagent extracts associated logical interface (LIF) names.

Using this information, the Ethernet Subagent uses the Shared Resource Library (SRL) routine LM_Get_Attributes_() to get the corresponding physical interface (PIF) name and the LIF type.

Using the SRL routine LM_Get_Status_(), the Ethernet Subagent gets the status of the LIF.

- If the LIF type is Ethernet and the access status is UP, the subagent gets PIF statistics by issuing the SRL routine LM_Get_Statistics_() on the corresponding PIF. The subagent then maps the obtained statistics to the Ethernet MIB tabular objects.
- If the LIF type is Ethernet and the status is STOPPED, zeros are assigned to all the Ethernet MIB tabular objects.

When the refresh timer times out, the subagent updates the dot3StatsTable. If a SUBNET is deleted or added, the subagent removes or adds the corresponding table entry during the next MIB refresh.

The subagent assigns a number for each SUBNET defined in TCP/IP. That number represents both the index for the dot3StatsTable and the tabular object instance identifier. This number is assigned once at startup time and never changes while the subagent executes, unless an addition or deletion of interfaces occurs.

Traps Generated by the Ethernet Subagent

The Ethernet Subagent generates traps to inform SNMP managers of the status of the Ethernet interfaces to the configured TCP/IP subsystem. The SNMP managers to which the Ethernet Subagent sends traps are identified by the set of trap destination definitions configured for the SNMP agent with which the Ethernet Subagent is communicating. See Section 2, Installing and Configuring the SNMP Agent, for information on configuring trap destinations.

The Ethernet Subagent monitors the status of the Ethernet interfaces at the time interval specified in its zesaRefereshTimer object. Whenever the status of an Ethernet interface changes, the Ethernet Subagent sends a trap. The Ethernet Subagent only sends traps to broadcast type trap destinations.

Table 14-7 describes traps the Ethernet Subagent generates. The traps are a subset of the generic SNMP traps prescribed in RFC 1157.

Table 14-7. Traps Generated by Ethernet Subagent

Generic Trap Type	Reason for Generating Trap
linkDown	Signals that the Ethernet interface has gone from an operational to a disabled state.
linkUp	Signals that the Ethernet interface has gone from a disabled to an operational state.

EMS Support

This subsection describes the Event Management Service (EMS) events, listed in Table 14-8, generated by the Ethernet Subagent (subsystem abbreviation ZESA).

Table 14-8. Ethernet Subagent Event Messages (page 1 of 2)

Number	ZESA-EVT-	Standard Event and Description
3001	SUBAGENT-AVAIL	Object Available The Ethernet Subagent process (zesaProcessName) has completed process initialization after initial startup or after a takeover by the backup process.
3002	SUBAGENT-UNAVAIL	Object Unavailable The Ethernet Subagent process is terminating for reasons other than an internal fault or invalid startup configuration.
3003	AGENT-OBJ-AVAIL	Object Available The configured SNMP agent process (zesaAgentName) has transitioned into the “connected” state.
3004	AGENT-OBJ-UNAVAIL	Object Unavailable The configured SNMP agent process has transitioned into the “disconnected” state.
3005	BACKUP-OBJ-AVAIL	Object Available The Ethernet Subagent backup process has transitioned into the “connected” state.
3006	BACKUP-OBJ-UNAVAIL	Object Unavailable The Ethernet Subagent backup process has transitioned from the “connected” state into the “disconnected” or “disabled” state.
3007	EMSCOLL-AVAIL	Object Available The configured EMS collector process (zesaEmsCollectorName) has transitioned into the “connected” state.
3008	EMSCOLL-UNAVAIL	Object Unavailable The configured EMS collector process has transitioned into the “disconnected” state.
3009	TCPIP-OBJ-AVAIL	Object Available The configured TCP/IP process (zesaTcpIpProcess) interface has transitioned into a “connected” state.

Table 14-8. Ethernet Subagent Event Messages (page 2 of 2)

Number	ZESA-EVT-	Standard Event and Description
3010	TCPIP-OBJ-UNAVAIL	Object Unavailable The configured TCP/IP process interface has transitioned from a “connected” state into a “disconnected” or “disabled” state.
3011	SRL-CALL-FAILED	Transient Fault A Shared Resource Library (SRL) call to the SLSA has failed.
3012	INTERNAL-FAULT	Object Unavailable The Ethernet Subagent process has terminated after detecting an internal fault condition.
3013	CONFIGURATION-INVALID	Object Unavailable The startup parameters passed to the Ethernet Subagent process at startup are unusable.
3014	TAKEOVER-BY-BACKUP	Transient Fault The Ethernet Subagent primary process has stopped and the backup process is taking over the primary role.

Data Definitions

Because they are standard events, most of the tokens, structures, and values appearing in ZESA events are defined in the ZSPI and ZEMS data definition files ZSPIDDL and ZEMSDDL and their associated language-specific files. Data elements defined by the Ethernet Subagent are in the file ZESADDL and associated language-specific files:

ZESADDL	Data Definition Language (DDL) definitions from which the language-specific definitions are derived
ZESAC	Definitions for C programs
ZESACOB	Definitions for COBOL programs
ZESAPAS	Definitions for Pascal programs
ZESATACL	Definitions for TACL programs
ZESATAL	Definitions for TAL programs

The complete set of SPI definition files is usually located in the ZSPIDEF subvolume of the NonStop Kernel installation volume. See the *SPI Programming Manual* for information on the ZSPI data definitions and data definition files in general. See the *EMS Manual* for information on ZEMS data definitions.

Subsystem ID

The subsystem ID that the Ethernet Subagent uses to identify itself as the source of event messages is:

```
CONSTANT ZESA-VAL-OWNER                VALUE  "TANDEM".
CONSTANT ZESA-VAL-NUMBER               VALUE  217.
CONSTANT ZESA-VAL-VERSION              VALUE  VERSION "G06".
CONSTANT ZESA-VAL-EXTERNAL-SSID        VALUE  "TANDEM.217.G06".

DEFINITION ZESA-VAL-SSID               TACL   SSID.
  02 z-filler  TYPE character 8      VALUE IS ZESA-VAL-OWNER.
  02 z-owner   TYPE ZSPI-DDL-CHAR8   REDEFINES z-filler.
  02 z-number  TYPE ZSPI-DDL-INT     VALUE IS ZESA-VAL-NUMBER.
  02 z-version TYPE ZSPI-DDL-UINT    VALUE IS ZESA-VAL-
VERSION.
END
```

Ethernet Subagent (ZESA) Tokens

Table 14-9 lists all the tokens defined by the Ethernet Subagent. These tokens are defined in the ZESADDL definition file, and are described where they appear in the event message descriptions later in this section.

Table 14-9. ZESA Tokens in ZESA Event Messages

Token	Identifies the Subject of the Event as Being the ...
ZESA-TKN-SUBAGENT	Ethernet Subagent process
ZESA-TKN-AGENT	Interface between the Ethernet Subagent and the SNMP agent with which it communicates
ZESA-TKN-BACKUP	Ethernet Subagent backup process
ZESA-TKN-EMSCOLLECTOR	Interface between the Ethernet Subagent and the EMS collector process to which it sends its event
ZESA-TKN-TCPIP	Interface between the Ethernet Subagent and the TCP/IP process it is monitoring
ZESA-TKN-SCP	Interface between the Ethernet Subagent and the SCP process used for SPI requests

Standard EMS Tokens

Table 14-10 lists the standard EMS tokens that the Ethernet Subagent includes in the event messages it generates. These tokens are defined in the ZEMSDDL definition file. See the *EMS Manual* for more information on these tokens and related data definitions.

Table 14-10. ZEMS Tokens in ZESA Event Messages

Token	Contents
ZEMS-TKN-CONTENT-STANDARD	Type of standard event.
ZEMS-TKN-CONTENT-USER	Type of user-defined event (for the Ethernet Subagent, the value is always ZEMS-VAL-NULL).
ZEMS-TKN-EVENTNUMBER	Event number.
ZEMS-TKN-SUBJECT-MARK	Event subject marker.
ZEMS-TKN-SUPPRESS-DISPLAY	Display/do not display event flag.
ZEMS-TKN-CHANGE-REASON	Reason for object state change.
ZEMS-TKN-STATE-CURRENT	Current object state; private enumerations: ZESA-VAL-STATE-CONNECTED ZESA-VAL-STATE-DISCONNECTED ZESA-VAL-STATE-ENABLED ZESA-VAL-STATE-DISABLED
ZEMS-TKN-STATE-PREVIOUS	Previous object state; private enumerations: ZESA-VAL-STATE-CONNECTED ZESA-VAL-STATE-DISCONNECTED ZESA-VAL-STATE-ENABLED ZESA-VAL-STATE-DISABLED
ZEMS-TKN-TXFAULT-TYPE	Reason for transient fault; private enumerations: ZESA-VAL-TF-SRL ZESA-VAL-TF-TAKEOVER

Event Message Descriptions

ZESA event messages are described in order by event number. Each description includes:

- Token lists. Tokens listed as “unconditional” always appear in the event message. Tokens listed as “conditional” are included only under described conditions.

  Tokens not defined by ZESA are listed only if they contain information that appears in the printed message text or if they contain ZESA-defined values.

- Event message text. The event message text illustrates what is generated when the contents of the event message are displayed according to the message template defined in the file ZESATMPL.

  <*n*> shows where text appears that is derived from a token in the token list.

  For a complete specification of the message, examine the message template source file SESATMPL. The message templates for standard formatted messages are defined in the *EMS Manual*.

- Descriptions of listed tokens or values.

- The cause of the event, the conditions that prompted the Ethernet Subagent to generate the event message.
- The effects associated with or resulting from the cause.
- Recovery procedures for solving the problem reported by the event message.
- An example of the formatted message.

3001: ZESA-EVT-SUBAGENT-AVAIL

The Ethernet Subagent process has completed process initialization after initial startup or after a takeover by the backup process.

Unconditional Tokens		**Value**
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-SUBAGENT
ZESA-TKN-SUBAGENT	<2>	zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-SUBAGENT-AVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-STATE-CURRENT	<6>	ZESA-VAL-STATE-CONNECTED
ZEMS-TKN-CONTENT-USER	<7>	ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state: <6>, user content: <7>
```

`ZESA-TKN-SUBAGENT`

identifies the subject of the event (the Ethernet Subagent process). The DDL heading of this token (“Ethernet-SNMP-subagent”) is inserted in the message text following “Object available.”

`zesaProcessName`

is the Ethernet Subagent process name specified for the subagent’s zesaProcessName MIB object.

`ZESA-EVT-SUBAGENT-AVAIL`

is the event number (3001). The DDL AS clause of this value (“subagent-process-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the Ethernet Subagent process has become available. The DDL AS clause associated with this value is inserted in the message text following "reason."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-UP	"underlying-serv-up" indicates that the Ethernet Subagent process has come up because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	"operator-initiated" indicates that an operator has brought the Ethernet Subagent process up.

`ZESA-VAL-STATE-DISCONNECTED`

indicates the previous state of the Ethernet Subagent process. The DDL AS clause of this value ("disconnected") appears in the message text following "previous state," indicating that the subagent was in a nonoperational state.

`ZESA-VAL-STATE-CONNECTED`

indicates the new state of the Ethernet Subagent process. The DDL AS clause of this value ("connected") appears in the message text following "current state," indicating that the subagent is in a normal operational state.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The Ethernet Subagent process completed process initialization after initial startup or after a takeover by the backup process.

Effect. The startup configuration (or last saved configuration in the case of a takeover by the backup) is used to establish connections with SNMP agent specified for the subagent's private zesaAgentName object and to build MIB values derived from the TCP/IP process specified by the zesaTcpIpProcess object.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
98-07-08 12:21:51 \NOVA.$ZESA      TANDEM.ETHSA.G06    003001
                                   Object available  Ethernet-SNMP-subagent -
                                   \NOVA.$ZESA:384566222,
                                   event number: subagent-process-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined
```

3002: ZESA-EVT-SUBAGENT-UNAVAIL

The Ethernet Subagent process is terminating for reasons other than an internal fault or invalid startup configuration.

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZESA-TKN-SUBAGENT
ZESA-TKN-SUBAGENT	<2> zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZESA-EVT-SUBAGENT-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5> ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7> *code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8> ZEMS-VAL-NULL

Conditional Tokens

ZEMS-TKN-UNDERLYING-OBJ-NAME	<9> zesaProcessName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>, user content:
<8>
```

`ZESA-TKN-SUBAGENT`

identifies the subject of the event (the Ethernet Subagent process). The DDL heading of this token (“Ethernet-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

`zesaProcessName`

is the Ethernet Subagent process name specified for the subagent’s private zesaProcessName MIB object.

`ZESA-EVT-SUBAGENT-OBJ-UNAVAIL`

is the event number (3002). The DDL AS clause of this value ("subagent-process-unavailable") appears in the message text following "event number."

`ZEMS-VAL-`*`reason`*

indicates the reason the Ethernet Subagent process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following "cause."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	"underlying-serv-failed" indicates that an underlying service on which the Ethernet Subagent depends has failed, or an extended memory segment of the subagent process is full and cannot be extended.
ZEMS-VAL-INTERNAL-FAILED	"internal-failed" indicates that an internal error in program logic or data was encountered.

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the Ethernet Subagent process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZESA-VAL-STATE-CONNECTED	"connected" indicates a running state, after initialization is complete.
ZESA-VAL-STATE-DISCONNECTED	"disconnected" indicates startup initialization state.

`ZESA-VAL-STATE-DISCONNECTED`

indicates the current state of the Ethernet Subagent process. The DDL AS clause of this value ("disconnected," defined above) appears in the message text following "previous state."

`zesaProcessName`

is the name of the underlying process whose failure caused the Ethernet Subagent process to go out of service. The name of the Ethernet Subagent process specified in the subagent's private zesaProcessName MIB object is inserted in the message text following "underlying object."

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

code-location/internal-context-text

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The Ethernet Subagent process terminated either:

- For reasons other than an internal fault or invalid startup configuration.
- Because the extended memory segment of the subagent process is full and could not be extended to satisfy a memory allocation request.

Effect. The Ethernet Subagent process is unavailable.

Recovery. In case of a memory allocation failure, free swap space on the subagent process swap volume and restart the subagent.

Otherwise, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- The saveabend file (if generated)
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products.

Sample Messages

```
98-07-08 11:53:20 \NOVA.$ZESA    TANDEM.ETHSA.G06    003002

                                 Object unavailable  Ethernet-SNMP-subagent -
                                 \NOVA.$ZESA:384566222,
                                 event number: subagent-process-unavail,
                                 cause: internal-failed,
                                 previous state: connected,
                                 current state disconnected,
                                 symptom string: T0326G06_25JUN98_ETHSA:
                                 AWAITIOX error -1,
                                 user content: undefined
98-07-08 11:53:20 \NOVA.$ZESA    TANDEM.ETHSA.G06    003002

                                 Object unavailable  Ethernet-SNMP-subagent -
                                 \NOVA.$ZESA:384566222,
                                 event number: subagent-process-unavail,
                                 cause: underlying-serv-failed,
                                 previous state: connected,
                                 current state disconnected,
                                 symptom string: T0326G06_25JUN98_ETHSA:
                                 create_node: malloc failed,
                                 user content: undefined
```

3003: ZESA-EVT-AGENT-OBJ-AVAIL

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZESA-TKN-AGENT
ZESA-TKN-AGENT	<2> zesaAgentName
ZEMS-TKN-EVENTNUMBER	<3> ZESA-EVT-AGENT-OBJ-AVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-UNDERLYING-UP
ZEMS-TKN-STATE-PREVIOUS	<5> ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZESA-VAL-STATE-CONNECTED
ZEMS-TKN-USER-CONTENT	<7> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

ZESA-TKN-AGENT

identifies the subject of the event (the SNMP agent process). The DDL heading of this token (“agent-interface”) is inserted in the message text following “Object available.”

zesaAgentName

is the SNMP agent process name specified for the subagent’s private zesaAgentName MIB object.

ZESA-EVT-AGENT-OBJ-AVAIL

is the event number (3003). The DDL AS clause of this value (“agent-interface-available”) appears in the message text following “event number.”

ZEMS-VAL-UNDERLYING-UP

indicates the reason the SNMP agent process has become available. The DDL AS clause associated with this value (“underlying-serv-up”) is inserted in the message text following “reason,” indicating that the interface with the SNMP agent process is available because an underlying resource on which it depends has become available.

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the SNMP agent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent functions dependent on the interface are not available.

`ZESA-VAL-STATE-CONNECTED`

indicates the new state of the SNMP agent process. The DDL AS clause of this value (“connected,” defined above) appears in the message text following “current state.”

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The Ethernet Subagent reacquired its connection to the configured SNMP agent process.

Effect. Requests can now be made against the MIB objects supported by the Ethernet Subagent.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
98-07-08 12:21:51 \NOVA.$ZESA       TANDEM.ETHSA.G06    003003
                                    Object available  agent-interface -
                                    \NOVA.$ZSNMP,
                                    event number: agent-interface-available,
                                    reason: underlying-serv-up,
                                    previous state: disconnected,
                                    current state: connected,
                                    user content: undefined
```

3004: ZESA-EVT-AGENT-OBJ-UNAVAIL

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZESA-TKN-AGENT
ZESA-TKN-AGENT	<2> zesaAgentName
ZEMS-TKN-EVENTNUMBER	<3> ZESA-EVT-AGENT-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-UNDERLYING-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5> ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-UNDERLYING-OBJ-NAME	<7> zesaAgentName
ZEMS-TKN-SYMPTOM-STRING	<8> *code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<9> ZEMS-VAL-NUL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
underlying object: <7>, symptom string: <8>, user content:
<9>
```

ZESA-TKN-AGENT

identifies the subject of the event (the SNMP agent process). The DDL heading of this token (“agent-interface”) is inserted in the message text following “Object unavailable.”

zesaAgentName

is the SNMP agent process name specified for the subagent’s private zesaAgentName MIB object.

ZESA-EVT-AGENT-OBJ-UNAVAIL

is the event number (3004). The DDL AS clause of this value (“agent-interface-unavailable”) appears in the message text following “event number.”

ZEMS-VAL-UNDERLYING-FAILED

indicates the reason the SNMP agent process has become unavailable. The DDL AS clause “underlying-serv-failed” is inserted in the message text following “cause,” indicating that an underlying service on which the SNMP agent depends has failed.

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the SNMP agent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent functions dependent on the interface are not available.

`ZESA-VAL-STATE-DISCONNECTED`

indicates the new state of the SNMP agent process. The DDL AS clause of this value (“disconnected,” defined above) appears in the message text following “current state.”

`zesaAgentName`

is the name of the underlying process whose failure caused the interface between the Ethernet Subagent and the SNMP agent to go out of service. The name of the SNMP agent process specified for the subagent’s private zesaAgentName MIB object is inserted in the message text following “underlying object.”

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The Ethernet Subagent lost its connection to the configured SNMP agent process.

Effect. SNMP requests cannot be made against the MIB objects supported by the Ethernet Subagent.

Recovery. Check the operating state of the SNMP agent process. If necessary, restart the SNMP agent. The Ethernet Subagent will attempt to reconnect to the SNMP agent at the rate indicated by its zesaKeepAliveTimer object.

Sample Message

```
98-07-08 12:18:39 \NOVA.$ZESA    TANDEM.ETHSA.G06    003004
                                 Object unavailable  agent-interface -
                                 \NOVA.$ZSNMP,
                                 event number: agent-interface-unavailable,
                                 cause: underlying-serv-failed,
                                 previous state: connected,
                                 current state: disconnected,
                                 underlying object: \NOVA.$ZSNMP,
                                 symptom string: T0326G06_25JUN98_ETHSA,
                                 updateAgent.2, agent interface error 201 on
                                 poll request,
                                 user content: undefined
```

3005: ZESA-EVT-BACKUP-OBJ-AVAIL

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZESA-TKN-BACKUP
ZESA-TKN-BACKUP	<2> zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZESA-EVT-BACKUP-OBJ-AVAIL
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5> ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6> ZESA-VAL-STATE-CONNECTED
ZEMS-TKN-USER-CONTENT	<7> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

ZESA-TKN-BACKUP

identifies the subject of the event (the backup Ethernet Subagent process). The DDL heading of this token ("backup-process") is inserted in the message text following "Object available."

zesaProcessName

is the Ethernet Subagent process name specified for the subagent's private zesaProcessName MIB object.

ZESA-EVT-BACKUP-OBJ-AVAIL

is the event number (3005). The DDL AS clause of this value ("backup-process-available") appears in the message text following "event number."

ZEMS-VAL-*reason*

indicates the reason the backup Ethernet Subagent process has become available. The DDL AS clause associated with this value is inserted in the message text following "reason."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-UP	"underlying-serv-up" indicates that the backup Ethernet Subagent process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	"operator-initiated" indicates that an operator has brought the backup Ethernet Subagent process up by setting the zesaBackupState MIB object to "enabled."

ZESA-VAL-STATE-*state*

indicates the previous state of the backup Ethernet Subagent process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	"connected" indicates a normal operational state.
ZESA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state where subagent function dependent on the interface was not available.
ZESA-VAL-STATE-ENABLED	"enabled" indicates that the process was set by an operator such that the subagent would immediately attempt to restore it to a connected state.
ZESA-VAL-STATE-DISABLED	"disabled" indicates that the process had been taken out of service explicitly by an operator setting the zesaBackupState MIB object to "disabled."

ZESA-VAL-STATE-CONNECTED

indicates the current state of the backup Ethernet Subagent process. The DDL AS clause of this value ("connected," defined above) appears in the message text following "current state."

ZEMS-VAL-NULL

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The Ethernet Subagent successfully started a backup process identified by zesaBackupCPU:zesaBackupPIN.

Effect. Ethernet Subagent configuration changes are now saved to the backup. In case of a primary failure, the backup will resume subagent processing using the last successfully checkpointed configuration.

Recovery. Informational message only; no corrective action is needed.

Sample Messages

```
98-07-08 11:31:36 \NOVA.$ZESA      TANDEM.ETHSA.G06    003005

                                   Object available  backup-process -
                                   \NOVA.$ZESA:384566222,
                                   event number: backup-process-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined
98-07-08 11:30:55 \NOVA.$ZESA      TANDEM.ETHSA.G06    003005

                                   Object available  backup-process -
                                   \NOVA.$ZESA:384566222,
                                   event number: backup-process-available,
                                   reason: operator-initiated,
                                   previous state: enabled,
                                   current state: connected,
                                   user content: undefined
```

3006: ZESA-EVT-BACKUP-OBJ-UNAVAIL

Unconditional Tokens		**Value**
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-BACKUP
ZESA-TKN-BACKUP	<2>	zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-BACKUP-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZESA-VAL-STATE-*state*
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	zesaProcessName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content:  <8>
```

`ZESA-TKN-BACKUP`

identifies the subject of the event (the backup Ethernet Subagent process). The DDL heading of this token (“backup-process”) is inserted in the message text following “Object unavailable.”

`zesaProcessName`

is the Ethernet Subagent process name specified for the subagent’s private zesaProcessName MIB object.

`ZESA-EVT-SUBAGENT-OBJ-UNAVAIL`

is the event number (3006). The DDL AS clause of this value (“backup-process-unavailable”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the backup Ethernet Subagent process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following “cause.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	“underlying-serv-failed” indicates that an underlying service on which the backup Ethernet Subagent process depends has failed.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has taken the backup Ethernet Subagent process out of service.

`ZESA-VAL-STATE-`*`state`*

indicates the previous and current states of the backup Ethernet Subagent process. The DDL AS clauses associated with this value are inserted in the message text following “previous state” and “current state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.

Value of ZEMS-TKN-STATE-CURRENT	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent function is not available.
ZESA-VAL-STATE-DISABLED	“disabled” indicates that the process has been taken out of service explicitly by an operator setting the zesaBackupState MIB object to “disabled.”

`zesaProcessName`

is the name of the underlying process whose failure caused the backup Ethernet Subagent process to go out of service. The name of the Ethernet Subagent process specified for the subagent’s private zesaProcessName MIB object is inserted in the message text following “underlying object.”

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The backup Ethernet Subagent process stopped or failed to start.

Effect. The Ethernet Subagent process continues to run without a backup. Configuration changes are not saved if a primary process failure occurs.

Recovery. Check the status of the configured backup processor (zesaBackupCPU). If it is not available or there are insufficient resources to run a backup process, disable the backup (set zesaBackupState "Disabled (3)"), set the zesaBackupCPU to an available processor, and reenable the backup (set zesaBackupState "Enabled(4)").

Sample Message

```
98-07-08 30:54:34 \NOVA.$ZESA     TANDEM.ETHSA.G06    003006

                                  Object unavailable  backup-process -
                                  \NOVA.$ZESA:384566222,
                                  event number: backup-process-unavailable,
                                  cause: underlying-serv-failed,
                                  previous state: connected,
                                  current state: disconnected,
                                  underlying object: \NOVA.$ZESA:384566222,
                                  symptom string: T0326G06_25JUN98_ETHSA,
                                  user content: undefined


98-07-08 11:12:19 \NOVA.$ZESA     TANDEM.ETHSA.G06    003006

                                  Object unavailable  backup-process -
                                  \NOVA.$ZESA:384566222,
                                  event number: backup-process-unavailable,
                                  cause: operator-initiated,
                                  previous state: connected,
                                  current state: disconnected,
                                  user content: undefined
```

3007: ZESA-EVT-EMSCOLL-AVAIL

Unconditional Tokens **Value**

```
ZEMS-TKN-SUBJECT-MARK         <1> ZESA-TKN-EMSCOLLECTOR
ZESA-TKN-EMSCOLLECTOR         <2> zesaEmsCollectorName
ZEMS-TKN-EVENTNUMBER          <3> ZESA-EVT-EMSCOLL-AVAIL
ZEMS-TKN-CHANGE-REASON        <4> ZEMS-VAL-reason
ZEMS-TKN-STATE-PREVIOUS       <5> ZESA-VAL-STATE-state
ZEMS-TKN-STATE-CURRENT        <6> ZESA-VAL-STATE-CONNECTED
ZEMS-TKN-USER-CONTENT         <7> ZEMS-VAL-NULL
```

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

ZESA-TKN-EMSCOLLECTOR

identifies the subject of the event (the EMS collector process). The DDL heading of this token (“emscollector-interface”) is inserted in the message text following “Object available.”

zesaEmsCollectorName

is the EMS collector process name specified for the subagent’s private zesaEmsCollectorName MIB object.

ZESA-EVT-EMSCOLL-AVAIL

is the event number (3007). The DDL AS clause of this value (“emscollector-interface-available”) appears in the message text following “event number.”

ZEMS-VAL-*reason*

indicates the reason the interface with the EMS collector process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	Associated Text (DDL AS Clause) and Meaning
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the interface with the EMS collector process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the EMS collector process up.

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the interface to the EMS collector process. The DDL AS clause associated with this value is inserted in the message text following "previous state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	"connected" indicates a normal operational interface between the EMS collector and the subagent.
ZESA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state such that subagent function dependent on the interface is not available.
ZESA-VAL-STATE-ENABLED	"enabled" indicates that the interface had been set by an operator such that the subagent will immediately attempt to restore it to a connected state.
ZESA-VAL-STATE-DISABLED	"disabled" indicates that the interface had been taken out of service explicitly by an operator setting the zesaEmsCollectorState MIB object to "disabled."

`ZESA-VAL-STATE-CONNECTED`

Indicates the current state of the interface between the EMS collector process and the Ethernet Subagent. The DDL AS clause of this value ("connected," defined above) appears in the message text following "current state."

`ZEMS-VAL-NULL`

Indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The subagent acquired or reacquired its connection to the configured EMS collector process (zesaEmsCollectorName).

Effect. Subsequent events generated by the subagent will be sent to the configured EMS collector.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
98-07-08 12:21:51 \NOVA.$ZESA   TANDEM.ETHSA.G06    003007
                                Object available  emscollector-interface -
                                \NOVA.$p0,
                                event number: emscollector-interface-
                                available,
                                reason: operator-initiated,
                                previous state: enabled,
                                current state: connected,
                                user content: undefined
```

3008: ZESA-EVT-EMSCOLL-UNAVAIL

Unconditional Tokens

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-EMSCOLLECTOR
ZESA-TKN-EMSCOLLECTOR	<2>	zesaEmsCollectorName
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-EMSCOLL-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	zesaEmsCollectorName

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content: <8>
```

`ZESA-TKN-EMSCOLLECTOR`

identifies the subject of the event (the EMS collector process). The DDL heading of this token (“emscollector-interface”) is inserted in the message text following “Object unavailable.”

`zesaEmsCollectorName`

is the EMS collector process name specified for the subagent’s private zesaEmsCollectorName MIB object.

`ZESA-EVT-EMSCOLL-UNAVAIL`

is the event number (3008). The DDL AS clause of this value (“emscollector-interface-unavailable”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the interface with the EMS collector process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following “cause.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	“underlying-serv-failed” indicates that an underlying service on which the interface with the EMS collector process depends has failed.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has taken the EMS collector process out of service.

`ZESA-VAL-STATE-`*`state`*

Indicates the previous state of the interface with the EMS collector process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a normal operational state.
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state where subagent function dependent on the interface is not available.
ZESA-VAL-STATE-ENABLED	“enabled” indicates that the interface has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZESA-VAL-STATE-DISABLED	“disabled” indicates that the interface has been taken out of service explicitly by an operator setting the zesaEmsCollectorState MIB object to “disabled.”

`ZESA-VAL-STATE-DISCONNECTED`

indicates the current state of the interface with the EMS collector process. The DDL AS clause of this value (“disconnected,” described above) appears in the message text following “current state.”

`zesaEmsCollectorName`

is the name of the underlying process whose failure caused the interface between the Ethernet Subagent and the EMS collector to go out of service. The name of the EMS collector process specified for the subagent’s private zesaEmsCollectorName MIB object is inserted in the message text following “underlying object.”

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

Indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

Indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The subagent lost its connection to the configured EMS collector process (zesaEmsCollectorName).

Effect. Events generated by the subagent are discarded until the EMS collector interface is returned to a connected state. The subagent attempts to reacquire the EMS collector connection whenever the subagent attempts to generate an event message and based at the rate indicated by its zesaKeepAliveTimer object.

Recovery. Check the operating state of the EMS collector process (zesaEmsCollectorName).

Sample Messages

```
98-07-08 11:31:35 \NOVA.$ZESA     TANDEM.ETHSA.G06    003008

                                  Object unavailable emscollector-interface -
                                  \NOVA.$p0,
                                  event number: emscollector-interface-
                                  unavailable,
                                  cause: underlying-serv-failed,
                                  previous state: connected,
                                  current state: disconnected,
                                  underlying object: \NOVA.$p0,
                                  symptom string: T0326G06_25JUN98_ETHSA,
                                  user content: undefined


98-07-08 12:03:00 \NOVA.$ZESA     TANDEM.ETHSA.G06    003008

                                  Object unavailable emscollector-interface -
                                  \NOVA.$p0,
                                  event number: emscollector-interface-
                                  unavailable,
                                  cause: operator-initiated,
                                  previous state: connected,
                                  current state: disconnected,
                                  user content: undefined
```

3009: ZESA-EVT-TCPIP-OBJ-AVAIL

Unconditional Tokens	**Value**
`ZEMS-TKN-SUBJECT-MARK`	`<1> ZESA-TKN-TCPIP`
`ZESA-TKN-TCPIP`	`<2> zesaTcpIpProcess`
`ZEMS-TKN-EVENTNUMBER`	`<3> ZESA-EVT-TCPIP-OBJ-AVAIL`
`ZEMS-TKN-CHANGE-REASON`	`<4> ZEMS-VAL-`*`reason`*
`ZEMS-TKN-STATE-PREVIOUS`	`<5> ZESA-VAL-STATE-`*`state`*
`ZEMS-TKN-STATE-CURRENT`	`<6> ZESA-VAL-STATE-CONNECTED`
`ZEMS-TKN-USER-CONTENT`	`<7> ZEMS-VAL-NULL`

Conditional Tokens

None

Message Text

```
Object available <1>-<2>, event number: <3>, reason: <4>,
previous state: <5>, current state <6>, user content: <7>
```

`ZESA-TKN-TCPIP`

identifies the subject of the event (the TCP/IP process). The DDL heading of this token (“tcpip-resource”) is inserted in the message text following “Object available.”

`zesaTcpIpProcess`

is the TCP/IP process name specified for the subagent’s private zesaTcpIpProcess MIB object.

`ZESA-EVT-TCPIP-OBJ-AVAIL`

is the event number (3009). The DDL AS clause of this value (“tcpip-resource-available”) appears in the message text following “event number.”

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP process has become available. The DDL AS clause associated with this value is inserted in the message text following “reason.”

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-UP	“underlying-serv-up” indicates that the interface with the TCP/IP process is available because an underlying resource on which it depends has become available.
ZEMS-VAL-OPERATOR-INITIATED	“operator-initiated” indicates that an operator has brought the TCP/IP process up.

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the TCP/IP process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a normal operational interface between the Ethernet Subagent and the TCP/IP process it is monitoring.
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a nonoperational state such that subagent function dependent on the interface is not available.
ZESA-VAL-STATE-ENABLED	“enabled” indicates that the interface to the TCP/IP resource has been set by an operator such that the subagent immediately attempts to restore it to a connected state.
ZESA-VAL-STATE-DISABLED	“disabled” indicates that the interface to the TCP/IP resource has been taken out of service explicitly by an operator setting the zesaTcpIpState MIB object to “disabled.”

`ZESA-VAL-STATE-CONNECTED`

indicates the current state of the TCP/IP process being monitored. The DDL AS clause of this value (“connected,” defined above) appears in the message text following “current state.”

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The subagent acquired or reacquired its connection to the configured TCP/IP process (zesaTcpIpProcess).

Effect. The MIB object values are rebuilt using SPI responses from the TCP/IP process (zesaTcpIpProcess). When the update sequence is complete (as indicated by zesaRefreshNow value = "autoDynamicRefresh (0)"), SNMP requests made against the MIB objects will return current values.

Recovery. Informational message only; no corrective action is needed.

Sample Messages

```
98-07-08 12:08:44 \NOVA.$ZESA      TANDEM.ETHSA.G06    003009

                                   Object available  tcpip-resource -
                                   \NOVA.$ZTC0
                                   event number: tcpip-resource-available,
                                   reason: underlying-serv-up,
                                   previous state: disconnected,
                                   current state: connected,
                                   user content: undefined


98-07-08 12:21:51 \NOVA.$ZESA      TANDEM.ETHSA.G06    003009

                                   Object available  tcpip-resource -
                                   \NOVA.$ZTC1,
                                   event number: tcpip-resource-available,
                                   reason: operator-initiated,
                                   previous state: enabled,
                                   current state: connected,
                                   user content: undefined
```

3010: ZESA-EVT-TCPIP-OBJ-UNAVAIL

Unconditional Tokens		**Value**
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-TCPIP
ZESA-TKN-TCPIP	<2>	zesaTcpIpProcess
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-TCPIP-OBJ-UNAVAIL
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-*reason*
ZEMS-TKN-STATE-PREVIOUS	<5>	ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZESA-VAL-STATE-*state*
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

ZEMS-TKN-UNDERLYING-OBJ-NAME	<9>	*underlying-process*

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>,
[ underlying object: <9>,] symptom string: <7>,
user content: <8>
```

`ZESA-TKN-TCPIP`

identifies the subject of the event (the TCP/IP process). The DDL heading of this token (“tcpip-resource”) is inserted in the message text following “Object unavailable.”

`zesaTcpIpProcess`

is the TCP/IP process name specified for the subagent's private zesaTcpIpProcess MIB object.

`ZESA-EVT-TCPIP-OBJ-UNAVAIL`

is the event number (3010). The DDL AS clause of this value ("tcpip-resource-unavailable") appears in the message text following "event number."

`ZEMS-VAL-`*`reason`*

indicates the reason the TCP/IP process has become unavailable. The DDL AS clause associated with this value is inserted in the message text following "cause."

Value of ZEMS-TKN-CHANGE-REASON	**Associated Text (DDL AS Clause) and Meaning**
ZEMS-VAL-UNDERLYING-FAILED	"underlying-serv-failed" indicates that an underlying service on which the TCP/IP process depends has failed.
ZEMS-VAL-OPERATOR-INITIATED	"operator-initiated" indicates that an operator has taken the TCP/IP process out of service.

`ZESA-VAL-STATE-`*`state`*

indicates the previous and current states of the TCP/IP process being monitored. The DDL AS clauses associated with this value are inserted in the message text following "previous state" and "current state."

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-CONNECTED	"connected" indicates a normal operational state.

Value of ZEMS-TKN-STATE-CURRENT	**Associated Text (DDL AS Clause)**
ZESA-VAL-STATE-DISCONNECTED	"disconnected" indicates a nonoperational state where subagent function is not available.
ZESA-VAL-STATE-DISABLED	"disabled" indicates that the process has been taken out of service explicitly by an operator setting the zesaTcpIpState MIB object to "disabled."

`underlying-process`

is the name of the underlying process whose failure caused the interface between the Ethernet Subagent and the TCP/IP process to go out of service. The name of one of the following processes is inserted in the message text following "underlying object":

- The name of the TCP/IP process specified for the subagent's private zesaTcpIpProcess MIB object
- The name of the SCP process specified for the subagent's private zesaScpProcess MIB object
- The name of the SNMP agent process specified for the subagent's private zesaAgentName MIB object

This information is displayed only if the value of the ZEMS-TKN-CHANGE-REASON is ZEMS-VAL-UNDERLYING-FAILED.

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. The Ethernet Subagent process lost its connection to the configured TCP/IP process (zesaTcpIpProcess), the SCP SPI interface process (zesaScpProcess), or the SNMP agent process (zesaAgentName).

If the underlying object is the SNMP agent process (zesaAgentName), the subagent has lost its connection to the agent process and has temporarily disconnected the TCP/IP interface to conserve resources.

Effect. Values for the MIB objects supported by the Ethernet Subagent are not computed. SNMP requests for these MIB objects receive an SNMP-ERR-GEN-ERROR response. Private MIB objects supported by the Ethernet Subagent can still be accessed.

Recovery. If the underlying object is the TCP/IP process or the SCP process, use SCF or the STATUS command to check the operating state of the TCP/IP or SCP process. The subagent attempts to reacquire the TCP/IP connection according to the value of its zesaKeepAliveTimer object.

If no error conditions are found in the TCP/IP and SCP processes, or if the subagent fails to reacquire the SCP interface, stop and restart the subagent with a different SCP process.

If the underlying object is the SNMP agent process, check the operating state of the SNMP agent process. If necessary, restart the SNMP agent. The Ethernet Subagent attempts to reconnect to the SNMP agent at the rate indicated by its zesaKeepAliveTimer object.

Sample Messages

```
98-07-08 12:21:51 \NOVA.$ZESA      TANDEM.ETHSA.G06    003010

                                   Object unavailable tcpip-resource -
                                   \NOVA.$ZTC1,
                                   event number: tcpip-resource-unavailable,
                                   cause: underlying-serv-failed,
                                   previous state: connected,
                                   current state: disconnected,
                                   underlying object: \NOVA.$zscp,
                                   symptom string: T0326G06_25JUN98_ETHSA:,
                                   updateSCP.6 retcode (-10),
                                   user content: undefined


98-07-08 12:18:39 \NOVA.$ZESA      TANDEM.ETHSA.G06    003010

                                   Object unavailable tcpip-resource -
                                   \NOVA.$ZTC0,
                                   event number: tcpip-resource-
                                   unavailable,
                                   cause: underlying-serv-failed,
                                   previous state: connected,
                                   current state: disconnected,
                                   underlying object: \NOVA.$ZSNMP,
                                   symptom string: T0326G06_25JUN98_ETHSA:,
                                   user content: undefined
```

3011: ZESA-EVT-SRL-CALL-FAILED

Unconditional Tokens		Value
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-SUBAGENT
ZESA-TKN-SUBAGENT	<2>	zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-SRL-CALL-FAILED
ZEMS-TKN-TXFAULT-TYPE	<4>	ZESA-VAL-TF-SRL
ZEMS-TKN-SYMPTOM-STRING	<5>	*internal-context-text*
ZEMS-TKN-USER-CONTENT	<5>	ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Transient Fault <1>-<2>, event number <3>, fault type: <4>
symptom string: <5> user content: <6>
```

ZESA-TKN-SUBAGENT

identifies the subject of the event (the Ethernet Subagent process). The DDL heading of this token (“Ethernet-SNMP-subagent”) is inserted in the message text following “Transient Fault.”

`zesaProcessName`

is the Ethernet Subagent process name specified for the subagent's private zesaProcessName MIB object.

`ZESA-EVT-SRL-CALL-FAILED`

is the event number (3011). The DDL AS clause of this value ("srl-call-failed") appears in the message text following "event number."

`ZESA-VAL-TF-SRL`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value ("srl-call-failure") is inserted in the message text following "fault type."

`internal-context-text`

indicates where the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause "undefined" always appears in the message text following "user content."

Cause. A shared resource library (SRL) call failed.

Effect. Old values in the MIB are retained and the MIB does not get refreshed.

Recovery. Informational message only; no corrective action is needed.

Sample Message

```
98-07-08 10:51:02 \NOVA.$ZESA      TANDEM.ETHSA.G06    003011
                                   Transient Fault Ethernet-SNMP-subagent -
                                   \NOVA.$ZESA:384566222,
                                   event number: srl-call-failed,
                                   fault type: srl-call-failure,
                                   symptom string: LM_Get_Statistics_()
                                   failed with Severity 3, Origin 2 and
                                   Code 1
                                   user content: undefined
```

3012: ZESA-EVT-INTERNAL-FAULT

Unconditional Tokens		**Value**
ZEMS-TKN-SUBJECT-MARK	<1>	ZESA-TKN-SUBAGENT
ZESA-TKN-SUBAGENT	<2>	*zesaProcessName*
ZEMS-TKN-EVENTNUMBER	<3>	ZESA-EVT-INTERNAL-FAULT
ZEMS-TKN-CHANGE-REASON	<4>	ZEMS-VAL-INTERNAL-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5>	ZESA-VAL-STATE-*state*
ZEMS-TKN-STATE-CURRENT	<6>	ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7>	*code-location/ internal-context-text*
ZEMS-TKN-USER-CONTENT	<8>	ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>, symptom string: <7>
user content: <8>
```

`ZESA-TKN-SUBAGENT`

identifies the subject of the event (the Ethernet Subagent process). The DDL heading of this token (“Ethernet-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

`zesaProcessName`

is the Ethernet Subagent process name specified for the subagent’s private zesaProcessName MIB object.

`ZESA-EVT-INTERNAL-FAULT`

is the event number (3012). The DDL AS clause of this value (“internal-fault”) appears in the message text following “event number.”

`ZEMS-VAL-INTERNAL-FAILED`

indicates that an internal error was encountered. The DDL AS clause associated with this value (“internal-failed”) is inserted in the message text following “cause.”

`ZESA-VAL-STATE-`*`state`*

indicates the previous state of the Ethernet Subagent process. The DDL AS clause associated with this value is inserted in the message text following “previous state.”

Value of ZEMS-TKN-STATE-PREVIOUS	**Associated Text (DDL AS Clause) and Meaning**
ZESA-VAL-STATE-CONNECTED	“connected” indicates a subagent in a running state, having completed its startup initialization.
ZESA-VAL-STATE-DISCONNECTED	“disconnected” indicates a subagent performing startup initialization.

`ZESA-VAL-STATE-DISCONNECTED`

indicates the current state of the Ethernet Subagent process. The DDL AS clause of this value (“disconnected,” defined above) appears in the message text following “current state.”

`code-location/internal-context-text`

indicates where in the subsystem or application code the fault occurred.

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The Ethernet Subagent internal consistency checking logic has detected a program logic error.

Effect. The Ethernet Subagent terminates.

Recovery. Contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- The saveabend file (if generated)
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products.

Sample Message

```
98-07-08 11:49:38 \NOVA.$ZESA      TANDEM.ETHSA.G06    003012

                                   Object unavailable Ethernet-SNMP-
                                   subagent - \NOVA.$ZESA:384566222,
                                   event number: internal-fault,
                                   cause: internal-failed,
                                   previous state: connected,
                                   current state: disconnected,
                                   symptom string:T0326G06_25JUN98_ETHSA:,
                                   AWAITIOX error -1,
                                   user content: undefined
```

3013: ZESA-EVT-CONFIGURATION-INVALID

Unconditional Tokens	Value
ZEMS-TKN-SUBJECT-MARK	<1> ZESA-TKN-SUBAGENT
ZESA-TKN-SUBAGENT	<2> zesaProcessName
ZEMS-TKN-EVENTNUMBER	<3> ZESA-EVT-CONFIGURATION-INVALID
ZEMS-TKN-CHANGE-REASON	<4> ZEMS-VAL-INTERNAL-FAILED
ZEMS-TKN-STATE-PREVIOUS	<5> ZESA-VAL-STATE-ENABLED
ZEMS-TKN-STATE-CURRENT	<6> ZESA-VAL-STATE-DISCONNECTED
ZEMS-TKN-SYMPTOM-STRING	<7> *configuration-fault-detail*
ZEMS-TKN-USER-CONTENT	<8> ZEMS-VAL-NULL

Conditional Tokens

None

Message Text

```
Object unavailable <1>-<2>, event number: <3>, cause: <4>,
previous state: <5>, current state <6>, symptom string: <7>
user content: <8>
```

ZESA-TKN-SUBAGENT

identifies the subject of the event (the Ethernet Subagent process). The DDL heading of this token (“Ethernet-SNMP-subagent”) is inserted in the message text following “Object unavailable.”

zesaProcessName

is the Ethernet Subagent process name specified for the subagent’s private zesaProcessName MIB object.

ZESA-EVT-CONFIGURATION-INVALID

is the event number (3013). The DDL AS clause of this value (“startup-configuration-invalid”) appears in the message text following “event number.”

`ZEMS-VAL-INTERNAL-FAILED`

indicates that an internal error was encountered. The DDL AS clause associated with this value (“internal-failed”) is inserted in the message text following “cause.”

`ZESA-VAL-STATE-ENABLED`

indicates the previous state of the Ethernet Subagent process. The DDL AS clause of this value (“enabled”) appears in the message text following “previous state.”

`ZESA-VAL-STATE-DISCONNECTED`

indicates the current state of the Ethernet Subagent process. The DDL AS clause of this value (“disconnected”) appears in the message text following “current state.”

`configuration-fault-detail`

provides additional information about the configuration (“invalid startup params”).

`ZEMS-VAL-NULL`

indicates the type of subsystem defined event. Because the Ethernet Subagent process has not defined this as a private event, the DDL AS clause “undefined” always appears in the message text following “user content.”

Cause. The startup parameters passed to the subagent process contain invalid definitions or the specified agent process was not found.

Effect. The Ethernet Subagent terminates.

Recovery. Check the startup parameter string passed to the Ethernet Subagent. Check the operating state of the specified SNMP agent process to ensure that it is running and ready to accept subagent connections.

Sample Message

```
98-07-08 11:24:51 \NOVA.$ZESA    TANDEM.ETHSA.G06    003013

                                 Object unavailable Ethernet-SNMP-subagent -
                                 \NOVA.$ZESA:384566222,
                                 event number: startup-configuration-invalid,
                                 cause: internal-failed,
                                 previous state: enabled,
                                 current state: disconnected,
                                 symptom string: T0326G06_25JUN98_ETHSA:
                                 main(), invalid startup params
                                 user content: undefined
```

3014: ZESA-EVT-TAKEOVER-BY-BACKUP

Unconditional Tokens	**Value**
ZEMS-TKN-EVENTNUMBER	<1> ZESA-EVT-TAKEOVER-BY-BACKUP
ZEMS-TKN-TXFAULT-TYPE	<2> ZESA-VAL-TF-TAKEOVER

Conditional Tokens

None

Message Text

<1>, <2>

`ZESA-EVT-TAKEOVER-BY-BACKUP`

is the event number (3014). The DDL AS clause of this value (“Takeover-by-backup”) appears in the message text.

`ZESA-VAL-TF-TAKEOVER`

identifies the type of transient fault that occurred. For this event, the DDL heading of the value (“Primary-stopped”) appears in the message text.

Cause. The subagent primary process terminated. The recognized causes are either process termination or primary processor failure.

Effect. The backup process takes over the subagent primary role using the last known configuration parameters. Any transactions that were in progress at the time of the primary failure are lost.

The new primary attempts to restart a backup process in the former primary processor according to the zesaKeepAliveTimer interval.

Recovery. If the cause was a processor failure, change the backup processor (zesaBackupCPU) to a currently active processor.

If the cause was a primary process failure, examine the event log to determine the underlying cause for the process abend.

Sample Message

```
98-07-08 11:31:35 \NOVA.$ZESA     TANDEM.ETHSA.G06    003014
                                  Takeover by backup process, Primary-stopped
```

Converting Events to Traps

Any process that generates events, including the Ethernet Subagent, can have its events translated into traps by the EMS Trap Subagent. The EMS Subagent uses an event filter known as a “routing distributor” to determine where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert into traps. Refer to Section 9, EMS Trap Subagent, for more information.

Part V. Appendices

Part V consists of the following appendices, which provide reference information:

A

SCF Command Syntax Summary for the NonStop Agent

This appendix lists, in alphabetical order, the syntax for the SCF commands supported by the NonStop agent. It is included as a quick reference for those already familiar with how the NonStop agent SCF commands function.

In the following syntax diagrams, if you set a default PROCESS with the ASSUME PROCESS command, you can omit the NonStop agent process name and period, and just specify *#object-name*. For example:

```
-> ASSUME PROCESS $ZSNMP
-> INFO ENDPOINT #END1
-> STATUS ENDPOINT #END1
```

ABORT Command

The ABORT command causes the specified object configuration to become inactive. The object is not available for use by the NonStop agent until it is restarted. This is a sensitive command.

ENDPOINT Object

```
ABORT ENDPOINT [$agent-process.]#endpoint-name
```

PROFILE Object

```
ABORT PROFILE [$agent-process.]#profile-name
```

TRAPDEST Object

```
ABORT TRAPDEST [$agent-process.]#trapdest-name
```

ADD Command

The ADD command defines the configuration for an object. This is a sensitive command.

ENDPOINT Object

```
ADD ENDPOINT [$agent-process.]#endpoint-name
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address" ]
```

PROFILE Object

```
ADD PROFILE [$agent-process.]#profile-name
   [ , COMMUNITY "community-name" ]
   [ , HOSTADDR "ip-address"]
   [ , ACCESS { READONLY | READWRITE } ]
```

TRAPDEST Object

```
ADD TRAPDEST [$agent-process.]#trapdest-name
   [ , COMMUNITY "community-name" ]
   [ , NETWORK [\node.]$tcpip-process ]
   , HOSTADDR "ip-address"
```

ALTER Command

The ALTER command changes the configuration of an object. This is a sensitive command.

ENDPOINT Object

```
ALTER ENDPOINT [$agent-process.]#endpoint-name
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address" ]
```

PROCESS Object

```
ALTER PROCESS $agent-process
   [ , EMSCOLL [\node.]$ems-collector ]
```

PROFILE Object

```
ALTER PROFILE [$agent-process.]#profile-name
   [ , COMMUNITY "community-name" ]
   [ , HOSTADDR "ip-address"]
   [ , ACCESS { READONLY | READWRITE } ]
```

TRAPDEST Object

```
ALTER TRAPDEST [$agent-process.]#trapdest-name
   [ , COMMUNITY "community-name" ]
   [ , NETWORK [\node.]$tcpip-process ]
   [ , HOSTADDR "ip-address"]
```

DELETE Command

The DELETE command removes the definition of an object from the NonStop agent configuration. This is a sensitive command.

ENDPOINT Object

```
DELETE ENDPOINT [$agent-process.]#endpoint-name
```

PROFILE Object

```
DELETE PROFILE [$agent-process.]#profile-name
```

TRAPDEST Object

```
DELETE TRAPDEST [$agent-process.]#trapdest-name
```

INFO Command

The INFO command displays the current attribute values for an object. This is a nonsensitive command.

ENDPOINT Object

```
INFO ENDPOINT [$agent-process.]#endpoint-name [ , DETAIL ]
```

PROCESS Object

```
INFO PROCESS $agent-process [, SUB [ NONE | ALL | ONLY ] ]
   [, DETAIL ]
```

PROFILE Object

```
INFO PROFILE [$agent-process.]#profile-name [ , DETAIL ]
```

TRAPDEST Object

```
INFO TRAPDEST [$agent-process.]#trapdest-name  [ , DETAIL ]
```

NAMES Command

The NAMES command lists the names of all objects defined for a NonStop agent configuration.

PROCESS Object

```
NAMES [ PROCESS ] $agent-process [, SUB [ NONE | ALL | ONLY
]]
```

START Command

The START command activates an object for use in a NonStop agent operations environment. This is a sensitive command.

ENDPOINT Object

```
START ENDPOINT [$agent-process.]#endpoint-name
```

PROFILE Object

```
START PROFILE [$agent-process.]#profile-name
```

TRAPDEST Object

```
START TRAPDEST [$agent-process.]#trapdest-name
```

STATUS Command

The STATUS command displays the operational states of objects in a NonStop agent configuration. This is a nonsensitive command.

ENDPOINT Object

```
STATUS ENDPOINT [$agent-process.]#endpoint-name
```

PROCESS Object

```
STATUS PROCESS $agent-process [ , SUB [ NONE | ALL | ONLY ] ]
```

PROFILE Object

```
STATUS PROFILE [$agent-process.]#profile-name
```

TRAPDEST Object

```
STATUS TRAPDEST [$agent-process.]#trapdest-name
```

STOP Command

The STOP command causes an object to become inactive. The object is not available for use by the NonStop agent until it is restarted. This is a sensitive command.

ENDPOINT Object

```
STOP ENDPOINT [$agent-process.]#endpoint-name
```

PROFILE Object

```
STOP PROFILE [$agent-process.]#profile-name
```

TRAPDEST Object

```
STOP TRAPDEST [$agent-process.]#trapdest-name
```

TRACE Command

The TRACE command starts a trace operation on a NonStop agent process, alters trace parameters set by a previous TRACE PROCESS command, or stops a previously requested trace operation. This is a sensitive command.

PROCESS Object

```
TRACE PROCESS $agent-process { , STOP | { ,trace-option }...
}}

trace-option is
    COUNT count
    PAGES pages
    RECSIZE size
    SELECT { METHOD | 3  | ALL | -1}
    TO file-ID
    WRAP
```

VERSION Command

The VERSION command displays the version number, product name, product number, and release date of the NonStop agent product. This is a nonsensitive command.

PROCESS Object

```
VERSION [ PROCESS ] $agent-process [ , DETAIL ]
```

B

SCF Error Messages for the NonStop Agent

This appendix describes the types of error messages generated by SCF and provides cause, effect, and recovery information for the SCF error messages specific to the NonStop agent.

Types of SCF Error Messages

Command Parsing Error Messages

Command parsing error messages are generated when a command is being broken down into its component parts. These error messages have no associated error numbers and are generally self-explanatory. For example:

```
Expecting an existing SCF supported object name
Expecting an SCF command or a program file
```

SCF-Generated Numbered Error Messages

SCF generates numbered error messages that begin at 20000. For example:

```
SCF E20211 Invalid object type
```

Common Error Messages

SCF provides a pool of error messages, called common errors, that can be used by all subsystems. These errors always have negative error numbers. Each error message is preceded by the name of the subsystem in which the error is encountered and a single-character type code (E for critical or W for noncritical). For example:

```
SNMP E-00005 Command is not supported by this subsystem
SNMP E-00042 Resource Not Available
```

Subsystem-Specific Error Messages

Subsystem-specific error messages are generated by the subsystem and pertain solely to that subsystem. These errors always have positive error numbers. Like common errors, subsystem-specific error messages are divided into two classes—critical and noncritical:

- Critical messages can be serious, such as the notification of software errors from which there is no automatic recovery. Critical messages are preceded by the letter E.

An example of a critical NonStop agent error message:

```
SNMP E00009 Internet address invalid
```

- Noncritical messages are generally informational. Noncritical messages are preceded by the letter W.

  An example of a noncritical NonStop agent error message is:

```
SNMP W00004 Duplicate TRAPDEST definition
```

Common Versus Subsystem-Specific Errors

When you receive an SCF error, check the subsystem identifier to see which subsystem returned the error and look at the error number to determine whether it is a common error or a subsystem-specific error.

For more information on error messages:

Type of Error Message	Where To Find Additional Information
Common errors (negative-numbered errors)	The *SCF Reference Manual for G-Series Releases* contains cause, effect, and recovery information for common errors.
Subsystem-specific errors (positive-numbered errors) generated by the NonStop agent	This appendix contains cause, effect, and recovery information for errors returned by the NonStop agent.
Other subsystem-specific errors (positive-numbered errors) from underlying subsystems	Part IV, SNMP Subagents contains descriptions of the subsystem-specific error messages.

Example of a Subsystem-Specific Error

The following error is specific to the NonStop agent. No other subsystem generates this same error number and text. Cause, effect, and recovery information for this error is provided later in this appendix. The error number is a positive number.

```
SNMP E00005 Invalid wildcard specification in HOSTADDR
```

Example of a Common Error

In contrast, the following error can also be returned by the NonStop agent, but the error number and text is common to all subsystems using SCF. Many other subsystems might use this same -00005 error number and text. The error number is a negative number.

```
SNMP E-00005 Command is not supported by this subsystem
```

SCF Error Messages Help

To request help for any SCF error message:

```
-> HELP subsystem error-number
```

For example, suppose the following messages appears on your terminal:

```
SNMP E00001 Attribute conflict: HOSTADDR
SNMP E-00015 Object $ZSMP.#END1 Already Defined
SCF E20211 Invalid object type
```

To display additional information:

```
-> HELP SNMP 1
-> HELP SNMP -15
-> HELP SCF 20211
```

If You Have to Call HP

If the recovery for an error message indicates you should contact your service provider, be prepared to supply the following information. (If the error caused SCF to terminate, reenter SCF.)

1. Enter a LOG command to collect the following displays into a single file, for example:

   ```
   -> LOG PROBLEM !
   ```

2. Enter a LISTPM command to collect information about the product versions of the SCF components, a list of the product modules on your system, and information about any product modules running when the error occurred:

   ```
   -> LISTPM
   ```

3. Enter an ENV command to collect information about the SCF environment that was present when the error occurred:

   ```
   -> ENV
   ```

   If the error caused SCF to terminate, respecify any environmental characteristics that were present when the error occurred.

4. Enter a DETAIL CMDBUFFER command to capture the contents of the SPI buffer:

   ```
   -> DETAIL CMDBUFFER, ON
   ```

5. Reproduce the sequence of commands that produced the SCF error.

NonStop Agent Error Messages

The NonStop agent SCF error messages are listed in numeric order.

SNMP Error 00001

```
SNMP E00001 Attribute conflict:  HOSTADDR
```

Cause. You tried to define an ENDPOINT object with a HOSTADDR address that is either:

- Identical to the HOSTADDR of an existing ENDPOINT object
- A member of a full wildcard address specification in an existing ENDPOINT object

Two ENDPOINT objects cannot reference the same TCP/IP Internet address.

Effect. The ADD or ALTER command is not executed.

Recovery. Either:

- Redefine the ENDPOINT object, specifying an available Internet address, and retry the ADD or ALTER command.
- First STOP and DELETE or STOP and ALTER the existing ENDPOINT object whose configuration you want to override.

SNMP Error 00002

```
SNMP E00002 Attribute conflict:  COMMUNITY
```

Cause. You tried to define a PROFILE object with the same COMMUNITY attribute value as an existing PROFILE object definition. Two PROFILE objects cannot define the same community.

Effect. The ADD or ALTER command is not executed.

Recovery. Either:

- Redefine the PROFILE object, defining a new community name, and retry the ADD or ALTER command.
- First STOP and DELETE or STOP and ALTER the existing PROFILE object whose configuration you want to override.

SNMP Error 00004

```
SNMP W00004 Duplicate TRAPDEST definition
```

Cause. You defined a TRAPDEST object with the same NETWORK and HOSTADDR attribute values as an existing TRAPDEST object definition.

Effect. This is an informational message only. Configuring more than one TRAPDEST object that points to the same destination results in duplicate traps being sent to the same address.

Recovery. No action is necessary.

SNMP Error 00005

```
SNMP E00005 Invalid wildcard specification in HOSTADDR
```

Cause. You provided an invalid wildcard specification as a HOSTADDR attribute value. You defined either:

- A TRAPDEST object with one or more wildcard values in the HOSTADDR specification. Each TRAPDEST must point to a single Internet address.
- An ENDPOINT or PROFILE object with a partial wildcard specification as the HOSTADDR attribute value. Partial wildcard specifications are not supported for the ENDPOINT and PROFILE objects.

Effect. The ADD or ALTER command is not executed.

Recovery. Define a separate TRAPDEST object for each Internet address to which traps are to be sent. For ENDPOINT and PROFILE objects, define the object with either a full wildcard specification (“0.0.0.0”) or a specific Internet address.

SNMP Error 00006

```
SNMP W00006 Resource not available for obj-type: will retry
            connection later
```

Cause. You attempted to start an object, but a resource on which the object relies is not available.

Effect. If you were trying to start a TRAPDEST object, the object enters the STARTING state. The NonStop agent cannot send traps to the trap destination until the underlying resource becomes available. Each time the NonStop agent receives or generates a trap, it attempts to access the resources whose services it requires. As soon as the resource becomes available, the object enters the STARTED state.

If you were trying to start an ENDPOINT object, the object enters the STARTING state. The NonStop agent cannot receive SNMP messages through that request/response connection until the underlying resource becomes available. Using an internal timer, the NonStop agent periodically tries to access the resources whose services it requires. As soon as the resource becomes available, the ENDPOINT object enters the STARTED state.

Recovery. No action is required. However, you might want to investigate the unavailability of the resource if the object remains in the STARTING state.

SNMP Error 00007

```
SNMP W00007 Unable to update SNMPCTL file.
```

Cause. The NonStop agent could not update the SNMPCTL file with the configuration change you submitted.

Effect. The command is executed, but the SNMPCTL file no longer reflects the current environment.

Recovery. Check the event log to find out the corresponding file system error number. Then refer to the *Guardian Procedure Errors and Messages Manual* for a description of file system errors and possible recovery actions.

SNMP Error 00009

```
SNMP E00009 Internet address invalid
```

Cause. You tried to define an invalid dotted-decimal format Internet address for the HOSTADDR attribute.

Effect. The ADD or ALTER command is not executed.

Recovery. Redefine the object and specify a valid Internet address. For information on specifying Internet addresses, see the *TCP/IP Configuration and Management Manual*.

C Unsolicited SNMP Agent Messages

The SNMP agent generates traps and EMS event messages when noteworthy conditions, such as a change in the state of an object, occur. This appendix describes these traps and events.

Traps

Table C-1 describes traps the SNMP agent generates. The traps are a subset of the standard SNMP traps prescribed in RFC 1157.

Table C-1. Traps Generated by SNMP Agent

Generic Trap Type	Reason for Generating Trap
authenticationFailure	The SNMP agent received a request that could not be authenticated for one or more of these reasons: ● The request is associated with a community not configured into the authentication table. ● The request originates from an address not defined for a configured community. ● The request is a SetRequest PDU from a community configured for READONLY access. Refer to Configuring Security on page 2-26 for more information on the authentication scheme used by the SNMP agent.
coldStart	The SNMP agent is being initialized. A coldStart trap is generated when: ● The SNMP agent is started for the first time. The message goes to all configured trap destinations. ● The backup process takes over. The message goes to all configured trap destinations. ● A new trap destination is defined and started. The message goes to the new trap destination.

The SNMP agent forwards traps to SNMP managers configured to receive them. If started using the WARM startup parameter and no trap destination definitions exist, the SNMP agent creates a trap destination definition for the Internet address associated with any manager from which a request is received through TCP/IP. For more information, see Configuring Trap Destinations on page 2-38.

Events

Event messages are sent to the EMS collector specified when starting the SNMP agent or to the EMS collector assigned through SCF by altering the PROCESS object. By default, event messages are sent to $0 on the local node. You can suppress sending event messages with the COLLECTOR startup parameter. Configuring the EMS collector is described in Section 2, Installing and Configuring the SNMP Agent..

Table C-2 summarizes and the following pages describe event messages the SNMP agent generates.

Table C-2. SNMP Agent Event Messages

Event Number	**Symbolic Name (ZSMP-EVT-*event*)**	**Description**
001	INVALID-CAID	The SNMP agent process has a creator accessor ID (CAID) that does not belong to the super user group.
002	INTERNAL-ERROR	An internal program fault was detected.
003	STATE-CHANGE	The state of a SNMP agent object changed.
004	OSS-ERROR	The SNMP agent process was unable to either decode an SNMP request or encode an SNMP response.
005	SOCKET-ERROR	An error occurred when the SNMP agent was communicating with HP TCP/IP software through the sockets library.
006	EVT-CONFIG-ERROR	The SNMP agent was started with an incorrect configuration parameter.
009	BAD-IPC-PDU-RCVD	A message received from a subagent contained an error. One cause for an error is the incorrect installation of the ZSMPTMPL template file.
010	BAD-NMS-PDU-RCVD	A message received from an SNMP manager contained an error, such as an invalid object identifier.
012	NO-SUCH-TRAPDEST	A subagent process tried to send a trap to a trap destination that was not configured within the SNMP agent.
013	INVALID-PDU-ON-SVC-PNT	A subagent process has incorrectly opened the agent for the service requested.
014	GUARDIAN-ERROR	An error was returned in response to a Guardian procedure call issued by the SNMP agent.

The SNMP agent can generate events with event numbers having a symbolic name with the prefix ZCMK. These events have negative numbers in the range -32600 to -32767 and are described in the *Operator Messages Manua*.

001: ZSMP-EVT-INVALID-CAID

```
Process must run under a SUPER group creator accessor id
```

Cause. The SNMP agent process was started by a user who was not a super-group user (255,*n*), and the SNMP agent is receiving request messages on a port numbered less than 1023.

Effect. The agent process does not start.

Recovery. Use one of these approaches to enable a user who is not a super-group user to start the SNMP agent:

- Reset the program file as follows:
  1. Log on as a super-group user (255,*n*) and then start the SNMP agent.
  2. Use the File Utility Program (FUP) to give ownership of the SNMP agent program file (SNMPAGT) to a super-group user. Then secure the program file so that a user who is not a super-group user can execute it, and set the PROGID so that the owner ID of SNMPAGT is used as the creator accessor ID when the program is run.
- Alternatively, a user who is not a super-group user can restart the SNMP agent if the PORT startup parameter is assigned a value greater than 1023 when the subagent is started.

002: ZSMP-EVT-INTERNAL-ERROR

```
Internal error in method method-name at location location-
number.  Detail:  explanatory-text
```

method-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

explanatory-text

is information describing the problem.

Cause. The SNMP agent encountered an unrecoverable internal error, such as unsuccessful completion of initialization procedures or corruption of memory.

Effect. Depending on the severity of the situation, the SNMP agent responds as follows:

- If a nonfatal error occurs (such as when an unknown value is encountered in a request), the SNMP agent stops processing the request.
- When the SNMP agent cannot run at all and a takeover by the backup would only cause a recurrence of the error (such as when the SNMP agent cannot complete its initialization procedures), the agent process stops.
- When a takeover by the backup will clear the error, such as when corruption of memory occurs, the SNMP agent primary process abends and lets the backup take over.

Recovery. Contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs and the Saveabend file of the affected process

If your local operating procedures require contacting the HP Global Customer Center (GCSC), supply your system number and the numbers and versions of all related products as well.

003: ZSMP-EVT-STATE-CHANGE

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }
```

obj-type

is one of the following configurable objects: PROCESS, ENDPOINT, PROFILE, or TRAPDEST.

obj-name

is the name of the configurable object, in the form $*agent-process*.#*object-name*.

state

is one of the following: DEFINED, STARTED, STARTING, STOPPING, or STOPPED.

Cause. A SNMP agent object changed state for one of the following reasons:

`Object`	A component of the SNMP agent's configuration defined in the SNMP agent's SNMPCTL file entered a different state.
`Operator`	An SCF command or SNMP Set request was issued against a component of the SNMP agent's configuration, causing it to move to a different state.
`Service`	An underlying service that the component of the SNMP agent's configuration depends on became either available or unavailable.

Effect. Object and operator messages are generally informational only.

`Service` messages reflect the SNMP agent's ability to use a service. An unavailable service reduces the SNMP agent's ability to function as configured. An unwanted service that becomes available can cause the SNMP agent to function in an undesirable way.

Recovery. `Object` and `Operator` messages are informational only; no corrective action is needed. `Service` state changes can require investigation of the circumstances regarding the availability of the underlying resource.

004: ZSMP-EVT-OSS-ERROR

```
OSS error in method method-name at location location-number.
OSS Function:  OSS-function-name  Error:  OSS-err-num
```

`method-name`

is the name of a function used internally by the SNMP agent.

`location-number`

is a numeric identifier for a specific location within the function.

`OSS-function-name`

is the name of an Open Systems Solutions, Inc., (OSS) ASN.1 function used to encode and decode SNMP messages. This function, called by *`method-name`*, generated the error.

`OSS-err-num`

is an encoding or decoding error returned by the specified function. For values, see Table C-3.

Cause. One of the errors listed in Table 6-3 was returned to the SNMP agent by an Open Systems Solutions, Inc., (OSS) ASN.1 function.

Effect. If the error is fatal, the agent stops running. Messages associated with nonfatal errors are discarded.

Recovery. Contact your local Simple Network Management Protocol (SNMP) expert and provide all relevant information as follows:

- Descriptions of the problems and accompanying symptoms
- Details from the message or messages generated
- The OSS error number
- Supporting documentation such as Event Management Service (EMS) logs and the saveabend file of the affected process

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Table C-3. Encoding and Decoding Errors Returned by an Open Systems Solutions, Inc., ASN.1 Function (page 1 of 2)

OSS-err-num	**Fatal Error?**	**Description**
4	No	The end of the input buffer was reached before the data was fully encoded.
0	No	The PDU was encoded or decoded successfully. This value should never be seen in an event message for encoding.
-1	No	The user-provided output buffer is too small. This error can be an encoding or decoding error.
-2	Yes	An invalid encoding value was passed. This error can be an encoding or decoding error.
-3	Yes	A protocol data unit (PDU) number passed to the encoder has no corresponding PDU number generated by the ASN.1 compiler. This error can be an encoding or decoding error.
-4	No	The input data contains an encoded value that exceeds the implementation limits of the decoder. This is a decoding error.
-5	No	The data being decoded is erroneous. This is a decoding error.
-6	Yes	An unexpected NULL pointer was passed as an argument to the encoder. This error can be an encoding or decoding error.
-7	Yes	An incompatible version of the ASN.1 compiler was used to produce the control table. This error can be an encoding or decoding error.
-8	No	A memory allocation error occurred. This error can be an encoding or decoding error.
-9	No	Two possible causes for this error: • Encoding a varbind, the syntax selector (CHOICE) was out of range. • The tag associated with the protocol data unit (PDU) number specified with pdunum does not match the first PDU in the input buffer.

Table C-3. Encoding and Decoding Errors Returned by an Open Systems Solutions, Inc., ASN.1 Function (page 2 of 2)

OSS-err-num	Fatal Error?	Description
-10	No	A bad OBJECT IDENTIFIER was encountered. The first element must be 0 (ccit), 1 (iso), or 2 (joint-iso-ccit). The second element must be 0 (standard), 1 (registration authority), 2 (member body), or 3 (identified organization).
-11	Yes	An unexpected NULL pointer was passed to the encoder.
-12	Yes	A value having more than the allowed precision was found in a time type.
-13	Yes	A type with a length or count field before it had a negative length.
-15	Yes	The control table was corrupt.
-16	Yes	An attempt was made to violate the constraint imposed by SizeConstraint. For example, the number of values in an array of integers is greater than the number specified in the SizeConstraint of the corresponding SET OF INTEGER.
-18	Yes	An error occurred during error recovery.

Effect. If the error is fatal, the agent stops running. Messages associated with nonfatal errors are discarded.

Recovery. Contact your local SNMP expert and provide all relevant information as follows:

- Descriptions of the problems and accompanying symptoms
- Details from the message or messages generated
- The mnemonic associated with the OSS error number
- Supporting documentation such as Event Management Service (EMS) logs and the saveabend file of the affected process

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

005: ZSMP-EVT-SOCKET-ERROR

```
Socket error in method method-name at location location-
number.  Socket Function:  socket-function-name, Error:
socket-err-num, File Name:  socket-file-name
```

method-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

socket-function-name

is the name of a socket function. This function, called by *method-name*, generated the error.

socket-err-num

is the socket error number.

socket-file-name

is the name of the protocol stack with which the socket is communicating.

Cause. A socket library call returned an error when the SNMP agent was communicating with HP TCP/IP software.

Effect. Depends on the error condition.

Recovery. Network management personnel should consult the file $SYSTEM.SYSTEM.ERRNOH to translate the *socket-err-num* value to a socket error mnemonic. Then consult the *TCP/IP and IPX/SPX Programming Manual* to interpret the socket error.

If necessary, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- The socket error mnemonic to which *socket-err-num* translated
- Supporting documentation such as Event Management Service (EMS) logs and the Saveabend file of the affected process

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

006: ZSMP-EVT-CONFIG-ERROR

```
Configuration error in method method-name at location
location-number.  Detail: explanatory-text
```

method-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

explanatory-text

is information describing the problem.

Cause. The SNMPCTL file specified at agent startup is already in use by another agent process.

Effect. The agent does not start.

Recovery. Network management personnel should perform one of these two recovery procedures:

- Stop the agent that already has the SNMPCTL file open and then restart the agent.
- Start the agent from a subvolume that contains an SNMPCTL file not in use or in which a new SNMPCTL file can be created.

009: ZSMP-EVT-BAD-IPC-PDU-RCVD

```
Bad PDU received from process-name.  Method function-name at
location location-number.  Detail:  explanatory-text
```

process-name

identifies a subagent process.

function-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

explanatory-text

is information describing the problem.

Cause. A subagent constructed and sent the SNMP agent an interprocess communication (IPC) PDU that the agent cannot decode. This problem should occur only during the debugging of a subagent.

Effect. The SNMP agent discards and does not process the message. If you are using the Event Management Service (EMS) Trap Subagent, no events are translated into traps.

Recovery. The subagent developer must check the construction of the IPC PDU.

010: ZSMP-EVT-BAD-NMS-PDU-RCVD

```
Bad NMS PDU received from IP-address.  Method function-name
at location location-number.  Detail: explanatory-text
```

IP-address

is the internet protocol (IP) address of the SNMP manager that sent the erroneous message.

function-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

explanatory-text

is information describing the problem.

Cause. An SNMP manager sent the SNMP agent a message that contained decodable but erroneous data.

Effect. The SNMP agent discards the message.

Recovery. The manager developer must check the construction of the PDU.

012: ZSMP-EVT-NO-SUCH-TRAPDEST

```
Cannot send trap to non-existent trapDest, trapdest-name, for
process-name
```

trapdest-name

is the name of the trap destination.

process-name

is the name of the subagent process.

Cause. A subagent process tried to send a trap to a trap destination that was not configured within the SNMP agent.

Effect. The SNMP agent discards the message.

Recovery. Network management personnel should configure the trap destination for the agent process.

013: ZSMP-EVT-INVALID-PDU-ON-SVC-PT

```
Invalid PDU, pdu-name, received on service point, protocol-
name, from process-name
```

pdu-name

is the name of the protocol data unit (PDU).

protocol-name

is the name of the protocol.

process-name

is the name of the subagent process.

Cause. A subagent process has incorrectly opened the agent for the service requested.

Effect. The SNMP agent sends a close request to the subagent and waits for the close response from the subagent.

Recovery. The subagent developer needs to correct the problem and then restart the agent.

014: ZSMP-EVT-GUARDIAN-ERROR

```
Guardian error in method method-name at location location-
number.  Detail:  explanatory-text
```

method-name

is the name of a function used internally by the SNMP agent.

location-number

is a numeric identifier for a specific location within the function.

explanatory-text

is information including the Guardian procedure, the returned error, and in some cases a brief descriptive text.

Cause. A Guardian procedure called by the SNMP agent returned an error.

Effect. The SNMP agent continues execution but cannot return valid information for some of its private MIB objects.

Recovery. Collect any relevant information, including the details from the error message. Refer to the *Guardian Procedure Calls Reference Manual* for information about the specific procedure and returned error number.

If necessary, contact your service provider and provide all relevant information as follows:

- Descriptions of the problem and accompanying symptoms
- Details from the message or messages generated
- Supporting documentation such as Event Management Service (EMS) logs and the Saveabend file of the affected process

If your local operating procedures require contacting the GCSC, supply your system number and the numbers and versions of all related products as well.

Glossary

Abstract Syntax Notation One. A formal language used to define PDUs and MIBs for SNMP. The language was developed and standardized by CCITT (International Consultative Committee on Telegraphy and Telephony) and ISO (International Organization for Standardization).

access. The attribute of a MIB object that specifies how it can be accessed. Valid values are read-only, read-write, and non-accessible.

access mode. The ability of members of a community to access MIB objects. There are two access modes: READONLY and READWRITE.

address resolution protocol. The protocol in the Internet suite of protocols used to map Internet Protocol (IP) addresses onto Ethernet (and other media) addresses.

agent. In the SNMP framework, a process that performs the network management functions requested by SNMP managers. The SNMP agent accepts requests from, and sends responses and traps to, one or more SNMP managers.

agent process. The process that represents an instance of the running SNMP agent.

ALRFMT. The object file that the $ZNFMT process uses to format Syshealth problem-report alarm data into readable text.

application layer. The layer in the Internet suite of protocols that describes end-user service technologies.

ASN.1. *See* Abstract Syntax Notation One.

authentication. The process of determining whether an entity is allowed to perform an operation.

authenticationFailure trap. A trap signaling that the SNMP agent has received an SNMP request for which one or more of the following apply:

- The request is associated with a community that has not been configured into the authentication table.
- The request is from an address that has not been defined for a configured community.
- The requests attempts to perform a Set operation but is from a community that has been configured for READONLY access.

authentication scheme. The set of rules by which an SNMP message is accepted or rejected by the SNMP agent.

authentication table. An internal table that the SNMP agent consults to determine whether to accept an incoming SNMP request.

Basic Encoding Rules. A language for describing transfer syntax.

BER. *See* Basic Encoding Rules.

broadcast trap destination. In a SNMP agent configuration, a trap destination to which the SNMP agent directs traps when not specifically told to send a directed trap by the subagent. *See also* trap destination *and* directed trap destination.

coldStart trap. A trap signaling that the SNMP agent has been initialized and that some object values may change as a result. This trap is generated by the SNMP agent when it is first started or when its backup process takes over. In addition, whenever a TRAPDEST object is added and started, the SNMP agent sends a coldStart trap to it.

collector. A process, defined by EMS, to which subsystems report events.

columnar objects. The table cells that make up a table row.

community. The relationship between an SNMP agent and one or more SNMP managers that defines access to MIB objects. An agent establishes one community for each combination of access mode and SNMP manager station(s).

community name. A string of 1 to 50 characters identifying a community. SNMP messages always contain a community name. If the community name corresponds to a community known to the receiving SNMP entity, the receiving entity accepts the message.

connection. A logical binding between two or more users of a service. Also, the path between two protocol modules that provides reliable stream delivery service.

connectionless. Characterized by the absence of logical connections between communicating entities. Connectionless operation requires that the data itself contain all the context information, such as addresses, required to get it to its destination and identify it to the recipient. Higher levels of protocol, or the application itself, are responsible for message sequencing and error handling.

connection-oriented. Characterized by logical connections between communicating entities. Connection-oriented protocols ensure that data arrives at its destination in sequence and error free.

DARPA. *See* Defense Advanced Research Projects Agency.

datagram. The basic unit of information passed across the Internet. It is a self-contained unit of data transmitted independently of other datagrams. It contains the source and the destination address, along with the data to be transferred.

Defense Advanced Research Projects Agency. The agency of the U.S. Department of Defense under which the Internet suite of protocols was developed.

directed trap destination. In a SNMP agent configuration, a trap destination to which the SNMP agent directs traps only if explicitly told to do so by the subagent. *See also* trap destination and broadcast trap destination.

DEFINE name. A component of the routing destination statement in the routing distributor filter source code that identifies the SNMP agent to which the EMS Trap Subagent routes trap messages.

Distributed Systems Management/Software Configuration Manager. A tool for the centralized planning, management, and installation of software on distributed (target) HP systems. DSM/SCM runs on a customer’s central system and performs the tasks of receiving, archiving, configuring, and packaging software for the customer target sites. It also runs on each target system, where its primary function is to activate the software received from the central site.

distributor. A process, defined by EMS, responsible for sending event messages to applications, files, printers, or remote nodes.

DSM/SCM. *See* Distributed Systems Management/Software Configuration Manager.

dotted-decimal notation. The standard external representation of an Internet address comprising the ASCII value of each of the four octets, separated by periods (for instance, 133.50.85.43). Each octet value is a number in the range 0 to 255 inclusive, expressed either in decimal with no prefix or in hexadecimal preceded by the characters 0X (with X in either uppercase or lowercase letters). For example, two other valid ways to express this example address are all hexadecimal (0x85.0x32.0x55.0x2b), and a combination of decimal (ASCII) and hexadecimal (133.0X32.85.0X2B). *See also* Internet address.

empconfigure group. A collection of scalar objects in the ONS MIB that is used by HP problem management products.

EMS. *See* Event Management Service.

EMSMIBA. The name of the file containing the ASN.1 source code for the EMS Trap MIB.

EMSPROC. A system-level NonStop NET/MASTER MS NCL procedure that intercepts and processes messages from EMS.

EMS Trap MIB. The MIB used by the EMS Trap Subagent and the NonStop NET/MASTER Trap Subagent. It contains objects that characterize individual events.

EMS Trap Subagent. The SNMP subagent that translates EMS events into SNMP traps whose objects are defined in the EMS Trap MIB.

ENDPOINT object. The SNMP agent configuration object that describes an interface for request/response messages. The object has two attributes: HOSTADDR (the Internet address of the TCP/IP subnet address through which the SNMP agent listens for requests from SNMP manager stations) and NETWORK (the TCP/IP process that handles request/response messages).

end-to-end services. Services that move octets from one system to another.

entry. A row in a MIB table.

event. A significant change in some condition in the system or network. Events can be operational errors, notifications of limits exceeded, requests for action needed, and so on.

Event Management Service. A collection of processes, tools, and interfaces that provide event-message collection and distribution on NonStop systems.

Exterior Gateway Protocol. A reachability protocol used by gateways in a two-level internet.

filter. An EMS object that distributors can be configured to use to select events to route.

filterManagement group. A table of entries in the ONS MIB used to inhibit traps for an entire subsystem or for a particular event. The filterManagement group table contains one entry for each subsystem with which Open Notification Service is working (for example, NNM, OMS, and TMF).

flow control. A mechanism of the Open Notification Service subsystem that is implemented to prevent the sending of unwanted subsystem data to the network management platform (NMP).

Gigabit Ethernet 4-Port ServerNet adapter (G4SA). A multiport ServerNet adapter that provides 1000 megabits/second (Mbps) data transfer rates between HP NonStop™ S-series systems and Ethernet LANs. The G4SA is the only LAN adapter supported for the I/O Adapter Module (IOAM) enclosure, and it is installed in slots 1, 2, 3, 4, and 5 of an IOAM. Although the G4SA supersedes the Ethernet 4 ServerNet adapter (E4SA), Fast Ethernet ServerNet adapter (FESA), and the Gigabit Ethernet ServerNet adapter (GESA), it cannot be installed in an HP NonStop™ S-series enclosure.

GENTRAP. A Network Control Language (NCL) procedure of the NonStop NET/MASTER Trap Subagent that assembles an SNMP trap from an EMS event and calls NMMTRAP to handle communications with the SNMP agent. To define a trap that differs from the one defined by the EMS Trap MIB, you must create a MIB definition in ASN.1 and modify the GENTRAP's NCL code to define the new trap MIB.

Get operation. An SNMP operation that retrieves the value of a MIB object.

GetNext operation. An SNMP operation that retrieves the value of the next instance of a MIB object. It could access the next object in the MIB, or it could access the next entry in a MIB table.

GetNextRequest PDU. An SNMP PDU used to issue a GetNext request.

GetRequest PDU. An SNMP PDU used to issue a Get request.

GetResponse PDU. An SNMP PDU used to respond to Get, GetNext, and Set requests.

heterogeneous. Consisting of components following different protocols and often manufactured by different vendors.

HMSAMIB. The name of the file containing the ASN.1 source code for the Host Resources Subagent's MIB.

Host Resources Subagent. The SNMP subagent that provides information for system, storage, and device resources on the NonStop system where it is installed.

Host Resources Subagent's MIB. The MIB used by the Host Resources Subagent. It describes the mandatory groups of the Host Resources MIB defined in RFC 1514: hrSystem, hrStorage, and hrDevice.

hrDevice group. The MIB-II group in the Host Resources Subagent's MIB that describes devices ranging from printers to tape drives.

hrStorage group. The MIB-II group in the Host Resources Subagent's MIB that describes storage media attached to the host. Storage media include physical memory, virtual memory, fixed or removable disks, and so forth.

hrSWRun group. The optional MIB-II group in the Host Resources Subagent's MIB that describes the processes running on the system.

hrSWRunPerf group. The optional MIB-II group in the Host Resources Subagent’s MIB that describes the resource consumption of the processes contained in the hrSWRun group.

hrSystem group. The MIB-II group in the Host Resources Subagent’s MIB that describes general host characteristics, such as the number of current users.

ICMP. *See* Internet Control Message Protocol.

ICMP group. The group in MIB-II supported by the TCP/IP Subagent that provides information about the ICMP layer of a TCP/IP subsystem being managed.

IFTBL. *See* network interface file.

instance. A single occurrence of a MIB object. Some MIB objects have only a single instance: for example, sysDescr. Other MIB objects have several instances: for example, ifType.

interface layer. The layer in the Internet suite of protocols responsible for transmission at the hardware (media) level.

Interfaces group. The MIB-II group that provides information about the network interfaces of a TCP/IP subsystem being managed.

internet. When spelled with a lowercase “i,” internet refers to any collection of packet-switching networks that use the TCP/IP protocol suite. The Internet, spelled with an uppercase “I,” is the largest internet in existence.

Internet. A large collection of connected networks that use the Internet suite of protocols and function as a single, cooperative, virtual network. The Internet provides universal connectivity and three levels of network service: unreliable, connectionless packet delivery; reliable, full-duplex stream delivery; and application-level services (like electronic mail) that build on the first two. The Internet reaches many universities, government research labs, commercial organizations, and military installations.

Internet Activities Board. The technical body overseeing the development of the Internet suite of protocols.

Internet address. The 32-bit address assigned to hosts that want to participate in the Internet using TCP/IP. Internet addresses are an abstraction of physical hardware addresses, just as the Internet is an abstraction of physical networks. Actually assigned to the interconnection of a host to a physical network, an Internet address consists of a network portion and a host portion.

Internet Control Message Protocol. A simple reporting protocol for the Internet Protocol (IP) that provides low-level feedback about how the internet layer is operating. It supports a modest number of basic control messages for error reporting. Specifically, gateways and hosts use ICMP to send reports of problems about datagrams back to the original source of the datagram. ICMP also includes an echo request-reply used to test whether a destination is reachable and responding. This protocol is used by the network layer to communicate the reachability of particular network nodes and routing control information. Strictly speaking, ICMP is part of the IP protocol because it shares the same Ethernet type field.

internet layer. The layer in the Internet suite of protocols responsible for providing transparency over both the topology of the Internet and the transmission media used in each physical network.

Internet Protocol. The network protocol in the Internet suite of protocols. It defines the internet datagram as the basic unit of information exchanged on an internet network, and it provides the basis for a connectionless, best-effort packet delivery service.

Internet suite of protocols. A collection of communications protocols originally developed under DARPA sponsorship; currently, the de facto solution of open networking. The Internet suite of protocols can be viewed as having four layers: interface, internet, transport, and application.

IP. *See* Internet Protocol.

IP group. The group in MIB-II supported by the TCP/IP Subagent that provides information about the IP layer of a TCP/IP subsystem being managed.

LAN. *See* local area network.

leaf. In an object tree, a node without subordinates.

lexicographic ordering. The ordering exhibited by object identifiers, which are expressed as integers. Lexicographic ordering extends to object-instance identifiers, since an object-instance identifier is also a sequence of integers. This ordering allows managers to search for and access objects without specifying them by name. By using lexicographic ordering, a manager can traverse the structure of a MIB, At any point in the tree, the manager can supply an object or object-instance identifier and ask for the object instance that occurs next in the ordering.

LIF. *See* logical interface (LIF).

llocal area network. Any one of a number of technologies providing high-speed, low-latency data transfer among electronic entities at a single site.

LOCPROC. The NonStop NET/MASTER MS process that manages EMS event messages logged to a file.

log file. A file to which an EMS collector writes events it receives from subsystems and applications.

logical interface (LIF). In a G-series ServerNet environment, the process that allows an application or another process to communicate with data communications hardware.

managed node. A network device containing an SNMP agent.

managed objects. The resources supervised and controlled by SNMP managers. The agent stands between the managed resources and the SNMP manger. Managed resources are accessed via a virtual information store, the MIB.

Management Information Base. A conceptual collection of objects that can be defined and accessed via a network management protocol, such as SNMP. An SNMP agent contains the intelligence required to access MIB values, which usually describe the configuration and status of the agent itself as well as devices the agent is instrumented to monitor.

manager. In the SNMP framework, an application that automates the management of the elements of a network. The manager directs the operations of one or more agents. It sends requests to and receives responses and traps from agent processes. Multiple managers can coexist on a network, but there are generally fewer managers than managed nodes.

manager station. A device on which a manager runs.

management process. A process through which an application issues commands to a subsystem. A management process can be part of a subsystem, or it can be associated with more than one subsystem; in the latter case, the management process is logically part of each subsystem.

MIB. *See* Management Information Base.

MIB-II. The current standard for the core network management information expected to be implemented by managed nodes in an SNMP environment.

MIB object. An item of network management in a MIB.

MIB table. A collection of entries (rows) describing MIB objects that can have more than one instance.

MSGPROC. The NonStop NET/MASTER MS process that manages the EMS event messages that are intended for viewing at an Operator Control Services (OCS) window.

NCL. *See* Network Control Language.

network. A collection of subnetworks connected by intermediate systems (such as routers) and populated by end systems (hosts, terminal servers, printers).

Network Control Language. A structured, high-level, compiled language supplied with NonStop NET/MASTER MS and integrated with other NonStop NET/MASTER MS services. It is used to develop NCL procedures that automate NonStop NET/MASTER operations.

network element. A resource that is monitored and controlled by network management applications. A network element consists of the managed entity (devices such as hosts, gateways, and terminal servers) and the managed entity's agent. The managed entity's agent is responsible for performing the network management functions requested by the network management stations.

network interface file. The file used by the Host Resources Subagent to maintain values for MIB objects in the hrNetworkTable.

network management application. A software application consisting of a single program or a collection of programs that monitors and controls network elements. Network management applications generally display the state of the network graphically.

network management protocol. The protocol used to convey management information. SNMP is a network management protocol used to communicate management information between managers and agents.

network management variable. A MIB object.

Network Operation Console. The system responsible for managing one or more nodes on an internet network.

nnm group. A group of scalar objects in the NonStop NET/MASTER Trap MIB that is used as variable bindings in the traps created from NonStop NET/MASTER events.

NMMTRAP. An NCL procedure of the NonStop NET/MASTER Trap Subagent which contains procedures for opening and closing sessions with the SNMP agent and for forwarding traps to the SNMP agent. Unlike GENTRAP, NMMTRAP's NCL code should not be modified.

NOC. *See* Network Operation Console.

NonStop NET/MASTER MS. A network management environment that lets you monitor and control local and remote NonStop systems interactively or programmatically (using NCL procedures). It also lets you tailor your individual operations environment.

not-accessible. A MIB object attribute that excludes access by any community to object values.

object. An entity in an SNMP environment.

object descriptor. A textual name for an object.

object group. A logical collection of MIB objects with similar characteristics.

object identifier. A sequence of integers, derived by traversing an object tree, that identifies a particular MIB object.

object tree. The model used to identify SNMP objects. It comprises an unlabeled root connected to labeled nodes which in turn may have subordinate nodes. A node and its subordinate nodes make up a subtree. A node with no subordinates is a leaf object. Each object on the tree is assigned an object descriptor and an object identifier.

OCS. *See* Operator Control Services.

octet. An 8-bit value.

OID. *See* object identifier.

omf group. A collection of scalar objects in the Object Monitoring Facility Trap MIB that is used as variable bindings in traps created from OMF events.

onscommon group. A collection of scalar objects in the Transaction Monitoring Facility Trap MIB that is used as variable bindings in traps created from TMF events.

onsconfigure group. A collection of scalar objects in the ONS MIB that specifies the objects needed to invoke ONS (onsState) and to monitor ONS activity.

onsconfirmTrap group. A table of objects in the ONS MIB used for HP problem-management products.

ONSDB. The Enscribe file that contains the ONS statistics database.

ONSFLTOA. The event filter file for the $ZNSDES distributor process.

ONSFLTOG. The event filter file for the $ZNSDG distributor process.

ONSFLTOS. The event filter file for the $ZONS remote operations.

ONS Trap Subagent. The SNMP subagent that collects EMS events from the HP system event log ($0) and translates them into SNMP traps. These SNMP traps are sent to an SNMP manager by the SNMP agent.

open systems. The concept of multiple vendors' hardware and software working together without proprietary boundaries.

Operator Control Services. The NonStop NET/MASTER MS service that provides the central point of operational control of the local NonStop system, the local NonStop NET/MASTER MS system, and remote NonStop systems.

Parallel Library TCP/IP. An HP product that provides increased performance and scalability over NonStop Transmission Control Protocol/Internet Protocol (TCP/IP). Parallel Library TCP/IP coexists with NonStopTCP/IP on HP NonStop™ S-series systems and supports Ethernet 4 ServerNet adapters (E4SAs), Fast Ethernet ServerNet adapters (FESAs), and ServerNet wide area network (SWAN) concentrators.

PDU. *See* protocol data unit.

port number. A means for identifying an application entity to a transport service in the Internet suite of protocols. On a NonStop host, each server has its own port number.

port 161. An SNMP protocol entity receives request and response messages at UDP port 161 on the host with which it is associated.

port 162. An SNMP protocol entity receives trap messages on UDP port 162 on the host with which it is associated.

PROCESS object. The SNMP agent configuration object that describes attributes of the agent process, such as the EMS collector to which the SNMP agent should route events it generates.

PROFILE object. The SNMP agent configuration object that represents an entry in the authentication table. The object has three attributes: COMMUNITY (the name of a community), HOSTADDR (the Internet address of an SNMP manager station), and ACCESS (READONLY or READWRITE).

proprietary. A protocol suite controlled by a vendor, and developed so that the vendor's computers can communicate with each other.

protocol. A set of rules used by computers to communicate with each other.

protocol data unit. A unit of information encoded according to specific protocol rules, usually containing both protocol control and user data. The five PDUs supported by the SNMP agent are:

GetRequest PDU
GetNextRequest PDU
GetResponse PDU
SetRequest PDU
Trap PDU

protocol suite. A group of protocols, all related to a common framework.

proxy agent. An SNMP agent that has access to information not held locally. This agent must perform a nonlocal interaction to satisfy management requests that reference that information.

read-only. A MIB object attribute that lets authorized communities retrieve, but not modify, object values.

READONLY. An access mode that lets SNMP managers associated with a particular community retrieve, but not modify, the values of MIB objects. MIB objects that have read-only or read-write access are readable.

read-write. A MIB object attribute that lets authorized communities retrieve and alter object values.

READWRITE. An access mode that lets SNMP managers associated with a particular community both retrieve and modify the values of MIB objects. MIB objects that have read-write access can be modified. MIB objects that have read-only or read-write access can be read.

Request for Comments. A document in a series containing surveys, measurements, ideas, techniques, and observations, as well as proposed and accepted standards for the Internet suite of protocols, such as TCP/IP and SNMP.

request message. An SNMP message that initiates data flow between the SNMP manager and the SNMP agent. A request message may contain a SetRequest PDU, a GetRequest PDU, or a GetNextRequest PDU.

response message. An SNMP message that contains a GetResponse PDU with the reply to a request message.

Response operation. An SNMP operation that returns a response to the originator of a Get, GetNext, or Set request. It contains identifiers that associate it with the previous PDU and provide information about the status of the response (error codes, error status, and a list of additional information).

RFC. *See* Request for Comments.

RMS. *See* Rule Management Services.

root. The unlabeled top of an object tree.

routing distributor filter. An event filter used by the EMS Trap Subagent to specify where to route trap messages. The filter contains a destination statement that identifies the SNMP agent as the routing destination. The filter can also contain specifications for selecting events to convert traps.

RowStatus object. A MIB table variable that is used to manage the creation and deletion of conceptual rows, and is used as the value of the SYNTAX cause for the status column of a conceptual row.

RowStatus values. The values of the status column of a conceptual row. These values are:

active	Indicates that the conceptual row is available for use by the managed device.
notInService	Indicates that the conceptual row exists in the agent, but is unavailable for use by the managed device.
notReady	Indicates that the conceptual row exists in the agent, but is missing information necessary in order to be available for use by the managed device.

createAndGo	Is supplied by an SNMP manager to create a new instance of a conceptual row and to have it available for use by the managed device.
createAndWait	Is supplied by an SNMP manager to create a new instance of a conceptual row but not to have it available for use by the managed device.
destroy	Is supplied by an SNMP manager to delete all of the instances associated with an existing conceptual row.

Rule Management Services. The NonStop NET/MASTER MS application that lets you define rules to specify EMS message filtering and task automation requirements. This facility helps automate routine network management operations.

SCF. *See* Subsystem Control Facility.

SCP. *See* Subsystem Control Point.

ServerNet LAN Systems Access (SLSA) subsystem. The software that supports parallel LAN I/O in a G-series ServerNet based system.

Set Request. An SNMP operation that alters the value of a MIB object.

SetRequest PDU. An SNMP PDU used to host a Set request.

Simple Network Management Protocol. A collection of specifications for network management that includes a message protocol, the definition of the structure and format of management information, and associated concepts. Designed to operate with TCP/IP-based internets, SNMP is the most widely used network management protocol in the industry.

SLSA. *See* ServerNet LAN Systems Access (SLSA) subsystem.

SNMP. *See* Simple Network Management Protocol.

SNMP agent. *See* Agent.

SNMPAGT. The name of the executable file for the SNMP agent.

SNMP control file. A file, named SNMPCTL, in which the SNMP agent maintains values for its current configuration and persistent MIB objects in the MIB-II System and SNMP groups it supports.

SNMPCTL. *See* SNMP Control File.

SNMP group. The MIB-II group supported by the SNMP agent that provides statistical information about SNMP message traffic.

SNMP manager. *See* Manager.

SNMP manager station. *See* manager station.

socket. A pairing of an IP address and a port number. Also, a logical connection between two applications across a TCP/IP network. On SNMP systems, a socket corresponds to a file system open. The same application or system can use multiple sockets. When you identify the process or device with which you want to communicate, you identify the socket by specifying the address and port.

SPI. *See* Subsystem Programmatic Interface.

startup parameter. A parameter you specify when you invoke the SNMP agent or a subagent to control attributes of the running agent or subagent processes.

subagent. A process that handles a particular collection of network resources on behalf of an SNMP agent. Examples of subagents offered on SNMP systems are the EMS Trap Subagent and the Host Resources Subagent.

Subagent Toolkit. The product offered by HP that helps programmers generate subagents that make SNMP resources manageable by SNMP managers.

subnet. A physical network within an IP network. The terms “subnet” and “subnetwork” are used interchangeably.

Subsystem Control Facility. The interactive product you use to configure, control, and collect information about many SNMP subsystems. SCF is the primary configuration management facility for the SNMP agent.

Subsystem Control Point. A process through which management applications, including SCF, control subsystems.

Subsystem Programmatic Interface. A set of procedures and associated definition files and a standard message protocol used to define common message-based interfaces for communication among applications and subsystems that run on NonStop systems.

subtree. On an object tree, a node that is not the root or a leaf. It acts as the logical root of a related set of objects descending from it.

syh group. A collection of scalar objects in the Syshealth Trap MIB that is used as variable bindings in traps created from Syshealth events.

System group. The MIB-II group supported by the SNMP agent that provides information about the agent and the node on which it resides.

table. *See* MIB table.

Tandem LAN Access Method (TLAM). The pre-G-series communication subsystem that provides a LAN interface to the NonSTop system. TLAM software implements the IEEE 802.2 Logical Link Control (LLC) local area network (LAN) standard, providing a uniform interface between upper-layer protocol products (such as TCP/IP) and the physical LAN medium. In addition to 802.2 type 1 connectionless service, TLAM supports Ethernet and IEEE 802.3 protocols. TLAM allows NonStop systems to communicate with other systems and with workstations over local area networks based on the IEEE 802.3 and Ethernet II standards. The pre-G-series HP TCP/IP communications product uses TLAM.

HP TCP/IP. HP's implementation of TCP/IP. HP TCP/IP provides TCP/IP connections for the NonStop system. HP TCP/IP connects the NonStop system to a variety of systems, including the Integrity S2 (HP's fault-tolerant UNIX system) and systems manufactured by other vendors. Cooperative applications can partition functions to use the strengths of the different operating systems. Applications running on a NonStop system or an Expand network can transparently exchange data with TCP/IP devices.

TCP. *See* Transmission Control Protocol.

TCP group. The group in MIB-II supported by the TCP/IP Subagent that provides information about the TCP layer of a TCP/IP subsystem being managed.

TCPIPSA. The name of the executable file for the TCP/IP Subagent.

TCP/IP Subagent. The NonStop subagent that implements MIB-II groups, enabling management of TCP/IP subsystems.

template. An EMS object used to specify how to format event messages that need to be read as text.

TESTSEND. An NCL file that sends a predefined trap to a specified SNMP manager. This trap has values for all eight objects defined in the EMS Trap MIB.

TLAM. *See* Tandem LAN Access Method.

token. In SPI, a distinguishable unit in an SPI message. The SNMP agent and subagents use values supplied in tokens to derive MIB object values.

Transmission Control Protocol. The Internet standard transport protocol that provides a reliable, full-duplex, connection-oriented, stream service used by many application protocols. One of its main jobs is to recover from traffic that may have been discarded by the connectionless-type protocol IP. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection. Software implementing TCP usually resides on the operating system and uses the IP protocol to transmit information across the Internet. It is possible to terminate (shut down) one direction of flow across a TCP connection, leaving a one-way connection. The Internet protocol suite is often referred to as TCP/IP because TCP is one of the two most fundamental protocols. This protocol is used by applications that require reliable end-to-end data transfer. It is a byte-stream-oriented protocol and includes no concept of packet boundaries. The only guarantee is that all of the data sent will be received in the same order in which it was sent.

transport layer. The layer of the Internet suite of protocols that describes end-to-end services.

trap. An unsolicited PDU sent asynchronously by an SNMP agent to an SNMP manager station to signal an important event. The agent is responsible for performing threshold checks and only reporting conditions that meet certain threshold criteria. In an SNMP environment, traps are usually generated as the result of few and critical events, and the interrupt message is simple and short.

TRAPDEST object. The SNMP agent configuration object that describes an SNMP manager station to which to route trap messages. The object has three attributes: COMMUNITY (the name to put in the community-name field of trap PDUs), HOSTADDR (the Internet address of an SNMP manager to which traps are to be sent), and NETWORK (the TCP/IP process that handles the sending of trap messages).

trap message. An SNMP message that contains a trap PDU.

trap PDU. An SNMP PDU used to issue traps. The trap PDU contains these fields, with descriptions that are applicable to the SNMP agent and other HP SNMP products:

enterprise	The object identifier for the SNMP agent, indicating the origin of the trap: 1.3.6.1.4.1.169.3.155.1.
agent-address	The Internet address of the system on which the SNMP agent forwarding trap is installed.
generic-trap	A 16-bit number set to 6 to signify that the trap is enterpriseSpecific.
specific-trap	A 16-bit number set to 0.
time-stamp	A 32-bit number indicating how much time has passed since the SNMP agent was last started. The subagent inserts 0, and the SNMP agent inserts the value of sysUpTime from the MIB-II System group.
variable-bindings	The objects defined in the EMS Trap MIB.

UDP. *See* User Datagram Protocol.

UDP group. The group in MIB-II supported by the TCP/IP Subagent that provides information about the UDP layer of a TCP/IP subsystem being managed.

User Datagram Protocol. The internet standard protocol that allows an application program on one machine to send a datagram to an application program on another machine. UDP uses the Internet protocol to deliver datagrams. The important difference between UDP and IP is that UDP messages include a protocol port number, allowing the sender to distinguish among multiple destinations (application programs) on the remote machine.

varbind. *See* variable binding.

variable binding. In SNMP request, response, and trap PDUs, a list of variable names and corresponding values.

WAN. *See* wide area network.

well-known port. Any of a set of protocol ports preassigned for specific uses by transport level protocols (that is, TCP and UDP). Servers follow the well-known port assignments so clients can locate them.

wide area network. Any of a number of technologies that data transfer among electronic entities that are geographically remote from each other.

X.25. The CCITT standard protocol for transport-level network service. Originally designed to connect terminals to computers, X.25 provides a reliable stream transmission service that can support remote logon.

X.25 Access Method. A HP product that implements, for wide area networks, the services of the network layer and layers below.

X25AM. *See* X.25 Access Method.

X.25 network. Any network or subnetwork linked using X.25 standards. X.25 standards are CCITT standards that define packet-switching carrier communication in the network layer over wide area networks.

zagInEndpoint group. The group in the SNMP agent’s private MIB that describes request/response connections between the SNMP agent and SNMP managers.

zagInProcess group. The group in the SNMP agent’s private MIB that provides information about the SNMP agent’s process objects.

zagInProfile group. The group in the SNMP agent’s private MIB that defines the criteria the SNMP agent uses to determine whether to accept or reject incoming requests from SNMP managers.

zagInternal group. The group for the SNMP agent’s private MIB under which the zagInProcess, zagInEndpoint, zagInProfile, and zagInTrapdest groups are organized.

zagInTrapdest group. The group in the SNMP agent’s private MIB that defines SNMP manager Internet addresses and TCP/IP processes used by the SNMP agent to route traps to specific SNMP managers.

zhrmDevUnavail group. A group defined by HP in the Host Resources Subagent’s MIB that describes hrDevice group devices that are in a state that might require operator intervention.

zhrmDiskThreshold trap. A trap generated by the Host Resources Subagent that identifies a disk device that is experiencing critically high usage.

zhrmRAMThreshold trap. A trap generated by the Host Resources Subagent that identifies a CPU whose RAM storage area is experiencing critically high usage.

zhrmRefresh group. A group defined by HP in the Host Resources Subagent's MIB that provides information about the MIB value updates by the Host Resources Subagent and lets you request that the subagent refresh MIB values on demand.

zhrmSaProcess group. A group defined by HP in the Host Resources Subagent's MIB that provides information about the Host Resources Subagent process. This group also lets you change the priority of the subagent process, cause the subagent's backup process to take over and a new backup process to be created, and control whether the subagent maintains hrSWRun and hrSWRunPerf values.

zhrmTableInfo group. A group defined by HP in the Host Resources Subagent's MIB that describes the status of and operations performed on objects in the hrStorage and hrDevice groups.

zhrmThreshold group. A group defined by HP in the Host Resources Subagent's MIB that describes RAM and disk access in the hrStorage group whose usage has reached critically high levels.

ZHRMTMPL. The name of the file containing the compiled EMS message template used by the Host Resources Subagent to format event messages it generates for display.

zhrmTrapDeviceStateChange trap. A trap generated by the Host Resources Subagent that identifies a CPU, printer, disk, network device, or tape drive that might require operator attention.

ZSMPMIB. The file that defines the objects contained in the SNMP agent's private MIB. ZSMPMIB is supplied with the SNMP agent's installation subvolume (ISV) and must be incorporated into the network management application's MIB.

ZSMPTMPL. The name of the file containing the compiled EMS message template used by the EMS Trap Subagent to format traps it generates from EMS events.

ZTSAMIB. The name of the file containing the ASN.1 source code for the TCP/IP Subagent's private MIB objects.

ZTSATMPL. The name of the file containing the compiled EMS message template used by the TCP/IP Subagent to format event messages it generates for display.

ZZSMPTRP. The default name of the primary trace file created when you use the TRACE startup parameter when starting the SNMP agent.

ZZSMPTRB. The default name of the backup trace file created when you use the TRACE startup parameter when starting the SNMP agent.

Index

Special Characters