Verizon Packet Network Surveillance (VPNS)

Administration Guide

VPNS Version 2007.1

Verizon Information Technology

Document Author - Shraddha Y. Mhatre

REVISION HISTORY

This section specifies changes made to the document as it is being developed.

Date	Name	Source	Description of Change	Version Number
1/19/2007	Shraddha Mhatre		created and released document	2007.1

Table of Contents

1 OVERVIEW 8

1.1 VPNS FEATURES 8
1.2 VPNS ARCHITECTURE 8
1.2.1 External Interfaces 9
1.2.2 VPNS Components 9
1.2.3 VPNS Databases 9
1.3 VPNS SERVICES 11
1.3.1 Mediation Service (MS) 11
1.3.1.1 Mediation Service Configuration files 14
1.3.2 Correlation Service (CS) 15
1.3.3 Alarm/Package Flow in Correlation Services 16
1.3.4 Posting Service and the Entity Thread 17
1.3.5 Pattern Service 17
1.3.6 Trending Service 18
1.3.7 Thresholding Service 18
1.3.8 Package Service 18
1.3.9 Topology Loader 18
1.3.9.1 Major Functionalities 18
1.3.9.2 Class/Package description 19
1.3.9.3 Starting/Stopping Topology Loader 19
1.3.9.4 Inventory Data 20
1.3.9.5 Stylesheet Transformation (xslt) 21
1.3.10 Admin Service 21
1.3.11 Weblogic Server 22
1.3.12 Raw Message Browser 22
1.4 SYSTEM REQUIREMENTS 23
1.4.1 VPNS Web Client Requirements 23

2 EXTERNAL SYSTEM INTERFACES 24

2.1 BAAIS 24
2.1.1 Overview of Inventory data and BAAIS/iView teams 24
2.1.2 Interface with Baais/iView (getting inventory data) 24
2.1.2.1 Initial batch file dump 24
2.1.3 Live Topology updates 24
2.1.4 Back up of Inventory Data files 25
2.2 INMS 25
2.2.1 Alarm Feed 25
2.2.1.1 Live Alarm feed 25
2.2.2 Resync Handler 25
2.2.3 Activate/De-activate INMS Adapter 26
2.2.3.1 Resume the INMS Interface Adapter 26

2.2.3.2 Suspend the INMS Interface Adapter 26
2.2.4 Send Keep Alive Message 26
2.2.4.1 Config parameters for Keep Alive message 27
2.2.5 Verify Keep Alive message 27
2.2.6 Store & forward Message 27
2.2.6.1 Config parameters for stored & forward 28
2.2.6.2 How to handle buffer overflow 28
2.2.7 Verify the start and end of stored & forward process 28
2.2.8 Topology Loader interface with INMS 28
2.2.8.1 Cronjob details 28
2.2.8.2 inventoryFileTransferToINMS.sh 29
2.3 DELPHI 29
2.3.1 Delphi Interface Data Flow 30
2.3.2 Configuring the Delphi Interface 30
2.3.3 Verifying Delphi Data Queries 30
2.3.4 Delphi Query Parameters 31

3 VPNS GUI 33

4 TOOLS 34

4.1 SCRIPTS 34
4.1.1 inmsInventoryTransfer.sh 34
4.1.2 updateDBStats.sh 34
4.2 SERVER COMMANDS 34
4.3 MPE DEBUGGER 35

5 CORRELATION SERVICE RULES 36

5.1 OVERVIEW 36
5.2 CORRELATION SERVICES LIFE CYCLE 37
5.3 TOPOLOGY CACHING 38
5.4 POSTING RULES 38
5.4.1 Posting Rule Instruction 38
5.4.2 Posting Rule Example 39
5.4.3 Posting-Rules-DTD Syntax File 39
5.5 FILTERING SERVICE RULES 40
5.5.1 Filter Service Rule Instruction 40
5.5.2 Filter Service Rule Example 40
5.5.3 Filtering Rules - DTD Syntax File 41
5.6 MAPPING RULES 41
5.6.1 Mapping Service Scenarios 41
5.6.2 Mapping Rule Instruction 41
5.6.3 Mapping Rule Example 42
5.6.4 Mapping Rules - DTD Syntax File 43
5.7 PATTERN RULES 43
5.7.1 Pattern Rule Instruction 43
5.7.2 Pattern Rule Example 44
5.7.3 Pattern Rules - DTD Syntax File 45

5.8 TRENDING RULES 46
5.8.1 Trending Service Scenarios 46
5.8.2 Trending Service Rule Instruction 47
5.8.3 Trending Rule Example 48
5.8.4 Trending Rules - DTD Syntax File 48
5.9 THRESHOLDING 49
5.9.1 Thresholding Rule Instruction 49
5.9.2 Thresholding Rule Example 50
5.9.3 Threshold Rules - DTD Syntax File 50
5.10 PACKAGING RULES 51
5.10.1 Packaging Service Rule Instruction 52
5.10.2 Packaging Service Rule Example 52
5.10.3 Package Rules - DTD Syntax File 53
5.11 FAULT PACKAGE DETAIL INFORMATION ALARM PACKAGES TRIGGERED BY MULTIPLE ALARMS 55

6 VPNS TOPOLOGY 57

6.1 VPNS TOPOLOGY MODEL (V3) 57
6.2 EXPLANATION FOR THE TOPOLOGY MODEL 58

7 CONFIGURATION AND LOG FILES 59

7.1 OVERVIEW 59
7.2 LOCAL CONFIGURATION FILES 60
7.3 LOG FILES 60
7.3.1 Log Levels 61
7.4 PARAMETERS / VALUES 62

8 DATABASE SCHEMA 63

8.1 TOPOLOGY DATABASE 63
8.1.1 ENTITY 66
8.1.2 ENTITY_ALIAS 67
8.1.3 ENT_ATTR 67
8.1.4 ENTITY_RELATIONS 68
8.1.5 ENTITY_TYPE_DEF 68
8.1.6 ATTR_TYPE_DEF 69
8.1.7 RELATION_DEF 69
8.1.8 SERIAL_ID_VAULT 70
8.1.9 CONN_TEST 70
8.2 ALARM DATABASE 71
8.2.1 ALARM 72
8.2.2 ALARM_SOURCE 74
8.2.3 ALARM_TYPE 74
8.2.4 ALARM_STATE 75
8.2.5 CLR_ALARM 75
8.2.6 PACKAGE 77
8.2.7 PACKAGE STATUS 80
8.2.8 ENTITY_STATUS 80
8.2.9 CLR_PACKAGE 80

8.2.10 PACKAGE_OP *83*
8.2.11 CLR_PACKAGE_OP *83*
8.2.12 TT_INFO *84*
8.2.13 CLR_TT_INFO *85*
8.2.14 QUEUE *86*
8.2.15 QUEUE_CRITERIA *86*
8.2.16 NE_NOTES *87*
8.2.17 DB_PURGE_CONTROL *87*
8.2.18 DB_PURGE_LOG *88*
8.3 RAW MESSAGE DATABASE 89
8.4 DATABASE OPTIMIZATION SCHEDULE 89

9 APPENDIX A - INVENTORY DATA DESCRIPTION 90

10 APPENDIX B – MESSAGE PROCESSING ENGINE (MPE) 93

10.1 OVERVIEW 93
10.2 AND-OR STRUCTURE 94
10.3 LOOPING CONTROL STRUCTURE 95
10.4 CURSOR-MOVING OPERATIONS 95
10.5 MISCELLANEOUS MPE OPERATIONS 98
10.6 WORK AREA OPERATIONS 99
10.7 TEXT CAPTURING OPERATIONS 99
10.8 VARIABLE MANIPULATION OPERATIONS 100
10.9 OUTPUT OPERATION 102
10.10 FUNCTION DEFINITION AND CALLING OPERATIONS 102
10.11 VARIABLE SCOPE 103
10.12 BUILT-IN STRING OPERATIONS 103
10.13 JAVA INTEGRATION 105
10.13.1 Using PIEIntf *105*
10.14 ALARM SEVERITIES 106
10.15 USING THE MPE DEBUGGER 106
10.15.1 Syntax *107*
10.15.2 Commands *107*
10.16 EXAMPLE MPE DEBUGGER SESSION 108
10.16.1 Raw Message from Gateway Controller *108*
10.16.2 Using MPE Debugger to find the end of the message and filter it *108*
10.17 EXAMPLE VPNS MPE PATTERN FILE 110

11 APPENDIX C - CONFIGURATION FILES 112

11.1 ALARMSKULKER.XML 112
11.2 DBCONNPOOLKEY.PROP 113
11.3 MEDIATIONSVCE.XML 115
11.4 TOPOLOGYLOADER.XML 119
11.5 VPNSINMSADAPTER.XML 124
11.6 VPNSINMSRESYNC.XML 127
11.7 CONFIG.XML 128
11.8 VNMS.XML 137

12 APPENDIX D - DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 142

1 Overview

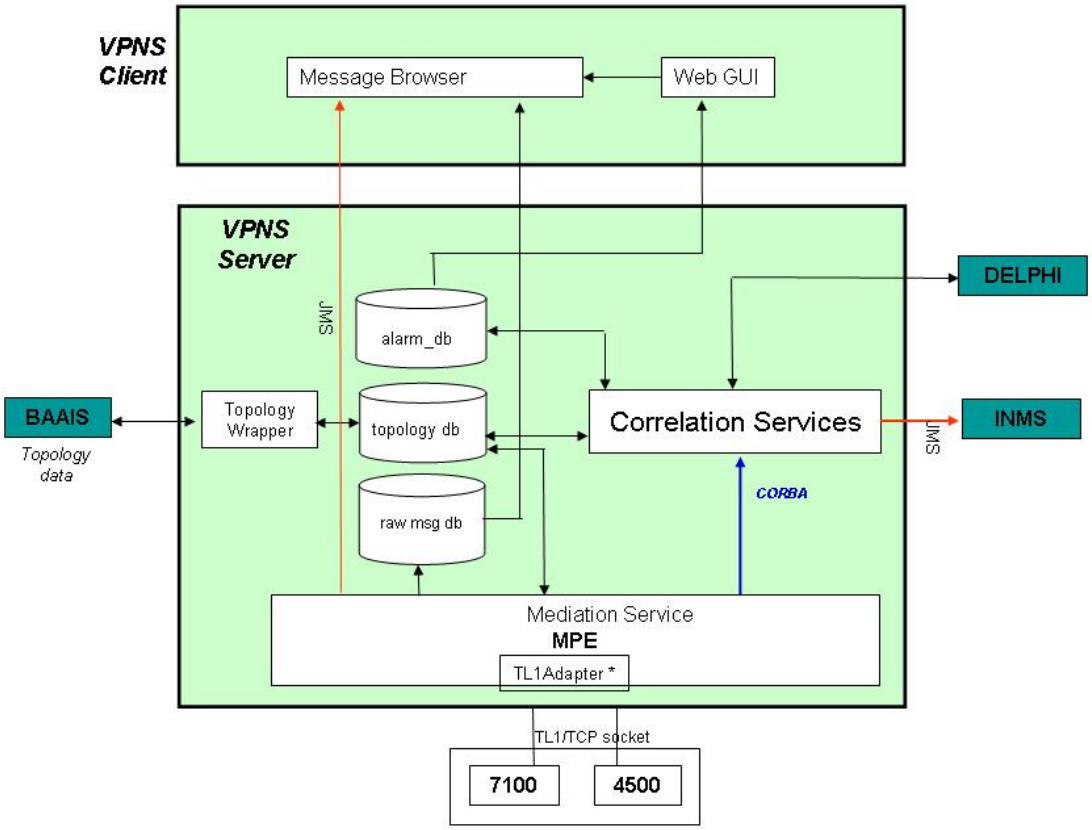
1.1 VPNS Features

VPNS Features are as follows:

- Network faults presented as Packages
- Alarm collection and processing. This includes filtering, trending (count-based thresholding), thresholding (time-based thresholding) and posting.
- Inventory feed from BAAIS which populates network topology.
- Web interface where System Administrators define user roles and access; users define their own network views.
- Message Browser for browsing raw messages by date/time/alarm string.
- Access Guardian Single Sign-On (SSOAG) used for user authentication.

1.2 VPNS Architecture

The below Architecture gives an overall view of VPNS system.

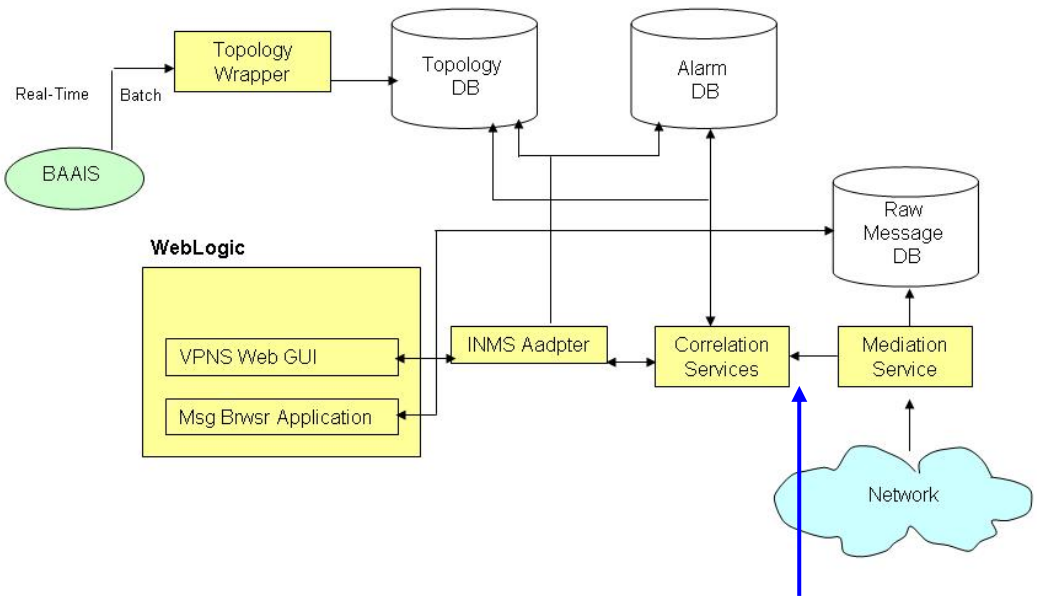


1.2.1 External Interfaces

VPNS receives topology data from inventory and provisioning systems and collects alarms from a variety of operation and support systems. In addition to collecting alarms from network entities, VPNS feeds the Packages and Topology updates to INMS.

Note: - Refer Chapter 2 for more details.

1.2.2 VPNS Components



Note: - In the below sections the VPNS components are explained in detail.

1.2.3 VPNS Databases

The VPNS server also houses three databases.
Topology Database - Receives nightly updates from BAAIS and also receives real-time updates for GE circuit activations. Both the Application Server and Correlation Services utilize the Topology data. The topology data includes inventories of network entities, logical networks, physical cards, and virtual cards. The data identifies which NEs are in service and have customer traffic, and is used

during correlation and packaging alarms.
This is the only VPNS database that requires metadata creation. This is accomplished by running the script loadDbs.sh, which loads empty skeleton tables to hold the inventory data. The script is run at initial installation and whenever the database schema needs to be updated.
The Topology Database is created with unbuffered logging, which means that when a commit is executed, the data is written to the table space immediately and replication occurs immediately on a redundant standby server using Informix CDR.

Alarm Database - Contains up-to-date information about all alarms and alarm packages received by and being processed in the VPNS system. Data includes package type, status, severity, and generation time. This database is continually updated by the Correlation and Package Monitor. Data is retained for a rolling 30 days (configurable).
The housekeeping tools for the Alarm Database are the AlarmJanitor script, which runs daily, and the PackageJanitor script, which runs monthly.
The Alarm Database is created with buffered logging, which means that when a commit is executed the data is written to the table space immediately but replication to the VPNS Standby Server does not occur until the buffer specified by Informix is full.
Package skulking runs every two hours and removes untouched message files that are older than 14 days from the VPNS server memory

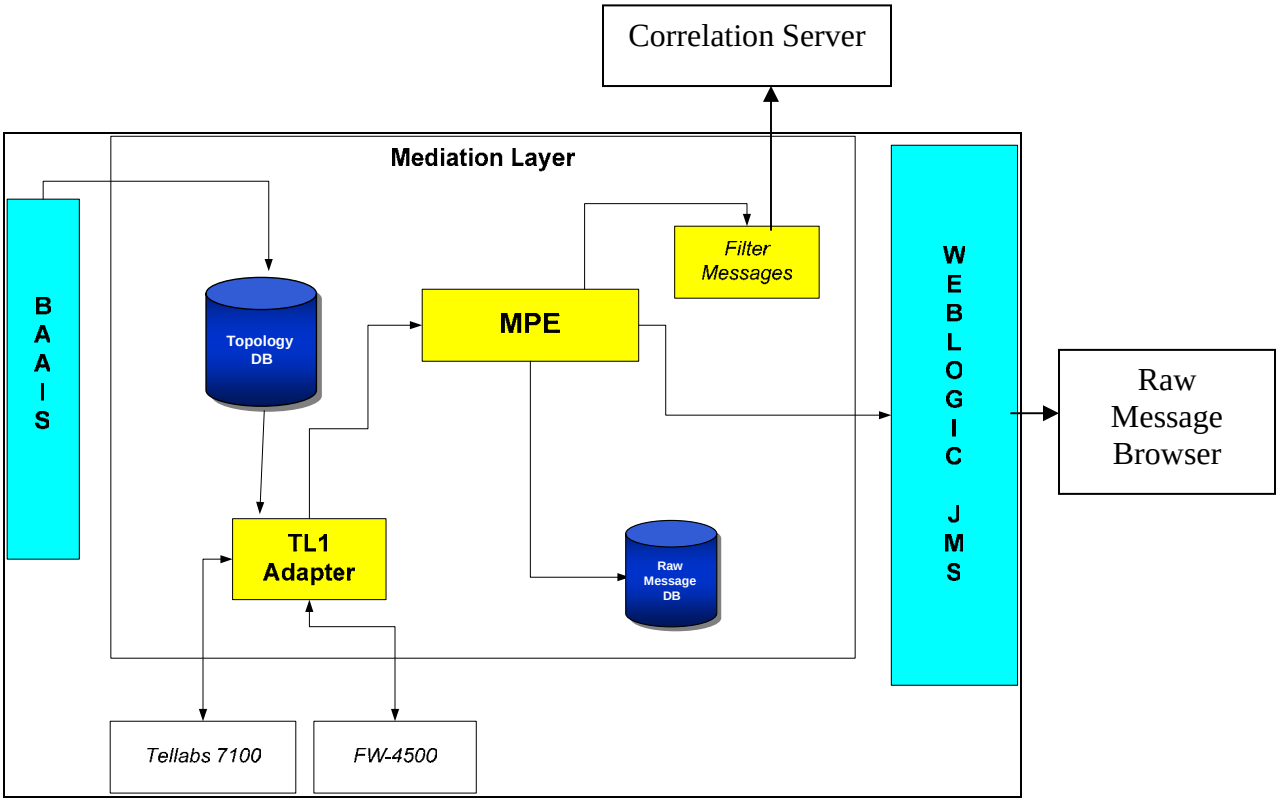
Raw Message Database - Contains all the raw messages received or retrieved by the VPNS Mediation Service. By default, messages are stored for 30 days. Due to the high volume of raw messages, the Raw Message Databases are not synchronized.
The Message Browser in the VPNS Web GUI can search and view raw messages in this database.

Note: - Refer **Chapter 8** for more details on VPNS Database.

1.3 VPNS Services

1.3.1 Mediation Service (MS)

Architecture of Mediation Service



The above figure shows the architecture of Mediation Service in VPNS. Inventory information comes from BAAIS and is stored in Topology Database. The Mediation Service collects raw messages from NEs (Tellabs-7100 and Fujitsu FW-4500), parses raw messages with the help of MPE, and sends them to both the Raw Message Database and Raw Message Browser. Those unfiltered, parsed messages (alarms) are forwarded to the correlation service.

The mediation Nodes are specified in the topology DB. Each node includes following information:

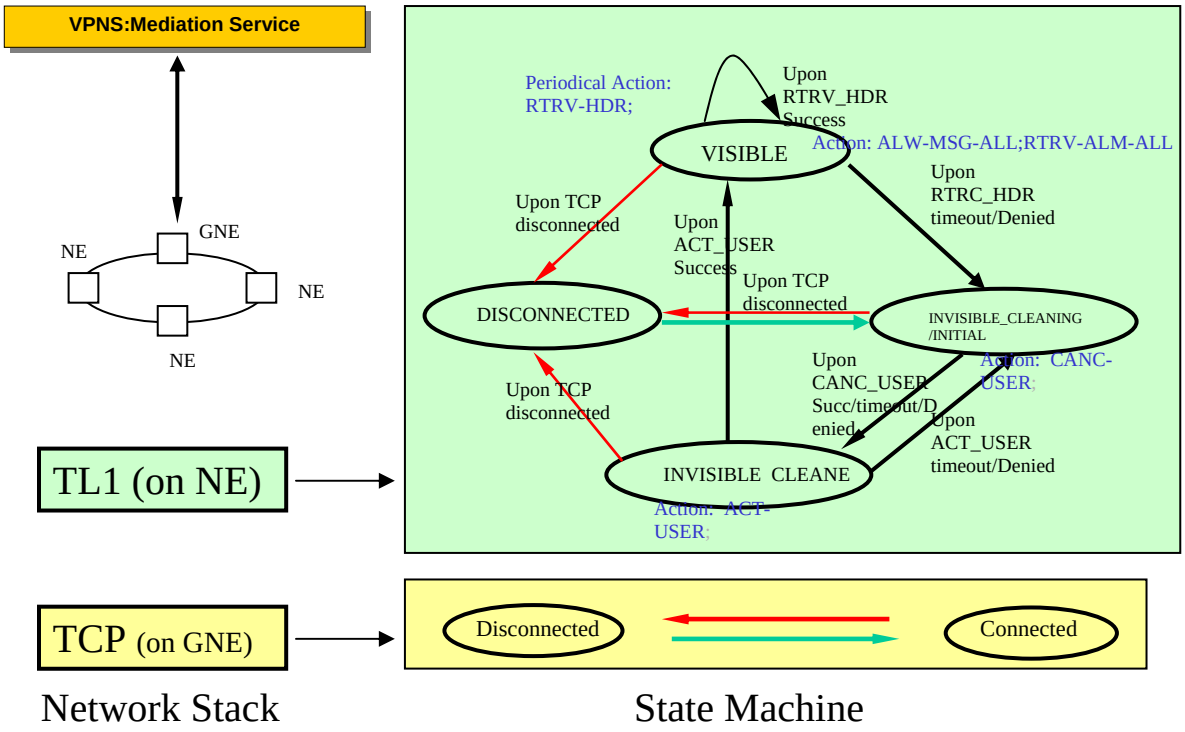
- Name: TID (Clli of the device),
- Primary gateway IP,
- NE Type: the vendor-device type for this particular NE.
- a port number
- Username/Password: Used in the login to the device monitored by VPNS.

As shown in above figure Architecture of Mediation Service, Mediation Service creates non-blocked socket connection to the mediation nodes (GNE) which has been presented in the topology DB. Mediation Service transport layer is implemented on the basis of java network IO (NIO) library. It is a two step process.

Step 1: Transport Connect (TCP connection): Open Socket Channel, one for each node, and establish connection to the IPAddress:port# specified for the TCPNode belonging to the channel.

Step 2: TL1 Session Connect: Creates the session specific variables for the each channel and then send the TL1 commands over the socket channel to each TL1 Device (Mediation Node). Once session is established with mediation nodes, Mediation Service starts collecting autonomous messages from the TL1 devices (GNE and NEs).

State Machine: TL1 Over IP



In mediation service, TCPMediationChannel and TL1NE are the two classes to implement the “State Machine: TL1 Over IP” shown in the Figure State Machine: TL1 Over IP. TCPMediationChannel takes charge of the Mediation Service Transport Layer. Once MS starts it initiates Tcp. This class creates instances for each TL1 devices, establishes the non-blocked socket channel with all the TL1 nodes and if connected successfully listens for the incoming asynchronous IO events. Based on the type of event it performs read write operations with each channel. TL1NE class maintains the state of a TL1 nodes. As shown in above Figure State Machine: TL1 Over IP, the starting state of each TL1NE is DISCONNECTED. The first trigger event MUST come from a successful Transport layer connection in TCPMediationChannel. Then the state of TL1 node becomes INVISIBLE_CLEANING, and a CANC_USER command is sent out to the device

immediately. The response of the CANC_USER could be COMPLETED/TIMEOUT/DENIED. TL1 node's state becomes INVISIBLE_CLEANED after receiving response of the CANC_USER. An ACT_USER command will be sent to the TL1 device regardless of the response of the CANC_USER command. TL1 node's state will be VISIBLE if ACT_USER is successful, and autonomous TL1 messages can be sent to VPNS from this device. If ACT_USER fails (TIMEOUT/DENIED), The state of the TL1 node goes back to INVISIBLE_CLEANING, and it will repeat the CANC_USER/ACT_USER commands until ACT_USER is successful.

In VISIBLE state, a TL1 NE will send out RTRV_HDR command to check the device's status every 5 minutes, which is configurable in Mediation.xml. If RTRV_HDR command fails, TL1 NE's state becomes INVISIBLE_CLEANING, and CANC_USER/ACT_USER commands will resume.

At any moment, TL1 NE's state becomes DISCONNECTED if the transport layer connection, here the TCP connection, is lost. Be noted that all NEs in the same ring will be affected by the loss of the transport layer connection loss.

Message Parsing Engine (MPE) parses these autonomous messages & creates a list of name/value pairs. These Name-value pairs together is an Autonomous TL1 Message. It is forwarded to the VPNS raw message browser through JMS Topic directly, while Autonomous TL1 Message with some fields of name-value pairs is stored in the Raw Message database. The Autonomous TL1 Message is also submitted to correlation server after they go through the filtering process, which filters the alarms based on the filtering rules. Only Alarms, Event & PM messages data will go through CORBA for correlation and packaging in Correlation Service of VPNS. When VPNS Mediation service receives an alarm in a format it cannot handle, it will drop the alarm and an exception message will be logged in mediationSvce.log.

All these autonomous messages are assigned a unique identifier for their storage in Raw Message db. One table per NE is maintained in the Raw Message Database. The table is named as <TID>_yyyymmdd. The table Life is 7 days. A new table for each NE is created at the midnight of Sunday while the old table is purged from the database by the DB Manager in the mediation service.

A Connection to the JMS Topic, which forwards the Automnomous TL1 Message to the raw message browser, is created at the time when the service starts. If the JMS Topic is unavailable, Service again tries to establish a connection after a reconfigurable Time Period. If JMS is not available at the time of publishing the message, the message is dropped.

Mediation Service does two levels of health-check as follows:-

- First Level is Sanity Check where mediation service checks for the validity and status of all the threads running in mediation server.

- Second level of Health-Check is accomplished with TL1 Devices using RTRV-HDR TL1 Command. This command is sent to each node after a reconfigurable interval. In case of health-check failure the service reports a LOV to Correlation Server.

Loss of Visibility (LOV) reporting is done at two levels as follows:-

- **Transport layer** – Transport Error occurs when channel is unable to establish a TCP connection to the GNE.

- **Session layer** – Session Error occurs when channel receives a deny/timeout response for TL1 Commands sent to the NE.

Table below describes the format of the LOV generated in VPNS mediation service.

Condition type	"comm_link failure"
AIDTYPE	"comm_link"
Alarm description (LOV header)	"comm_link failure".
Alarm Severity	Critical

1.3.1.1 Mediation Service Configuration files

Configurations for mediation services, such as default values for raw message table's life span, TL1NE's sanity check interval, etc., are specified in the mediation configuration file. This file is located at location "${VNMS_HOME}/etc/config/medsvc.xml".

Note: - For more details on Mediation service please refer **chapter 6**.

The following are VPNS - Mediation Service Configuration files:-

- **MediationSvce.xml / Config.xml** – Contains all the configurable parameters that are required to run Mediation Service. It configures various subsystems of Mediation Service via JMS, JDBC, logging etc; they are different standard services which Mediation Service requires for communicating with various sub-components of Mediation Service.

- **PieConfig.xml** – Contains the location of various Pie Grammar indexed by the NE Type of Device for which it is applicable.

```
<?xml version='1.0' encoding='us-ascii'?>

<!-- DTD for connection information -->

<!-- The document element -->
<!DOCTYPE pieConfigMatrix [

     <!ELEMENT pieConfigMatrix (pieConfig*)>

     <!ELEMENT pieConfig EMPTY>
     <!ATTLIST pieConfig
       Type    CDATA          #REQUIRED
       PieFileName    CDATA          #REQUIRED
     >
```

```
]>


<pieConfigMatrix>

    <pieConfig Type="TELLABS-7100E" PieFileName="tl7100.f" />
    <pieConfig Type="FUJITSU-FW4500E" PieFileName="fw4500.f" />
    <pieConfig Type="TELLABS-7100" PieFileName="tl7100.f" />
    <pieConfig Type="FUJITSU-FW4500W" PieFileName="fw4500.f" />

   </pieConfigMatrix>
```

The following are other Mediation configuration files:-

- **tl7100.f** – MPE Pie Grammar for Tellabs 7100 devices. It also contains filter rules to be used by mediation service to drop the unwanted alarms.
- **fw4500.f** – MPE Pie Grammar for FW-4500 devices. It also contains filter rules to be used by mediation service to drop the unwanted alarms.
- **Dbuser.par** – Contains Base-64 encoded database username and password.
- **Vnmsenv** – Contains environment variables.

1.3.2 Correlation Service (CS)

Raw messages and traps that have been determined to be alarms by the VPNS Mediation Service are forwarded to the VPNS Correlation Services for processing. As alarms pass from the Mediation Service to the Correlation Services, the state of the alarm can be:

- **1** = raw (unprocessed)
- **2** = processed and presented to user (processed)
- **3** = cleared after user has seen it (Cleared)
- **4** = Alarm filtered by VPNS without being presented to user (Filtered)
- **5** = Alarm cleared while waiting in Thresholding service (configurable delay, default – 30 seconds). User did not see. (Cleared in Threshold)

? Should this section belongs here

For the alarm get cleared, the clr_operator field will be used to track it

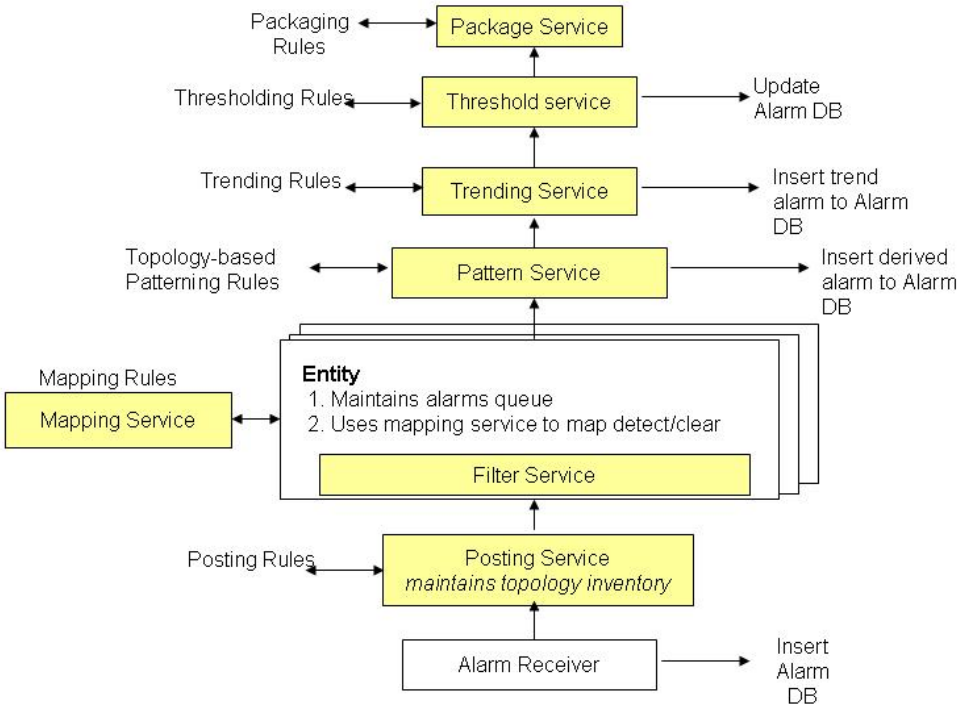
```
NETWORK_CLEAR = 1; For the alarm get cleared from the network.

PATTERN_CLEAR = 2; For the alarm get cleared in the pattering services.
```

```
SYSTEM_CLEAR = 3; For the alarm get cleared due to topology change.


RESOLVE_CLEAR = 4; For the alarm get cleared when use resolve the package.

TREND_CLEAR = 5; For the alarm get cleared in the trending services.


EXPIRED_CLEAR = 6;  For the alarm which gets cleared after alarm expiration time reached.

POSTING_CLEAR = 7; For the alarm which can not be posted on the GE topology entity and
thus get cleared

Filter Clear = 8; For the alarm get cleared in the filtering services.
```

The Correlation Services handle alarm posting, filtering, mapping, trending, thresholding, and packaging according to customizable correlation rules and use topology data from BAAIS for topology-based correlation.
Site-defined rules are stored in XML files. The locations of the XML files are listed in VNMS.par located in $VNMS_HOME/etc/.

1.3.3 Alarm/Package Flow in Correlation Services

Figure below shows the general flow of alarms and packages as they are processed by the Correlation Services.



1.3.4 Posting Service and the Entity Thread

The Correlation Services Alarm Receiver uses the Correlation Posting Service to assign a new alarm to an Entity. The task of moving the alarms through the VPNS Correlation services is performed by the Entity thread. Examples of Entities include: port, card, shelf, or NE. Each references the source or location of the alarm. Each Entity has its own user-defined mapping rules. The Entity maintains the alarm in the Correlation Services Alarm Queue and uses the Correlation Mapping Service to determine which alarms should be cleared.

The Topology cache every ten minutes to remove any topology entities that are not attached at a current alarm. When certain alarms are cleared, the Mapping Service may use this information to delete other alarms in the queue.
The remaining alarms are passed to the Pattern Service for correlation.

1.3.5 Pattern Service

The Pattern Service uses user-defined correlation rules to create derived alarms. The Pattern Service then sends a brief summary of each alarm to the VPNS Alarms database and forwards the derived alarms to the Trending Service.

1.3.6 Trending Service

The Trending Service identifies alarm trending patterns such as:

- **Bouncing Condition** - Alarms from the same source and same condition type are generated and cleared a number of times within a set timeframe.
- **Multiple Detects** - Multiple alarms from the same source and same condition type are generated within a given time frame.

The Trending Service can use **autorestore** (autoclear) to reduce alarms by matching problem messages with messages that pair with the problem message and clearing them without passing them to the VPNS monitoring technician. It can use **thresholding** to count certain alarms that contain specified strings.
Alarms that are not cleared are passed to the Thresholding Service and also sent to the Alarm Database.

1.3.7 Thresholding Service

The Thresholding Service holds alarms for a configurable time period (5-20 seconds), allowing the opportunity for alarms to be cleared by the Correlation Services. After the time period has expired, remaining alarms are sent to the Correlation Package Service.

1.3.8 Package Service

When an alarm enters the Package Service, the Package Service determines whether the alarm should be associated with a pre-existing Affected Entity (AE) package or a new package should be created. In either case, the alarm is dispatched via INMS Adapter to the INMS.

Note: - For detail information on correlation Service refer **chapter 5.**

1.3.9 Topology Loader

1.3.9.1 Major Functionalities

Topology Loader also known as Topology Wrapper does the following major tasks:

- loads topology metadata
- creates and maintains connection to JMS server, listen to Topic to get Topology live updates (from Baais/iView)
- continuously monitors designated location in the file system for initial batch dump of topology data files
- saves topology data for backup as well as for INMS transfer
- transforms topology xml coming from Baais/iView into standard format using stylesheet
- loads transformed xml into Topology DB to create entities and their relationship which would

represent the Topology existing in the field (as per Topology Model)
- publish topology update messages (when NE is updated) to JMS Topic for Mediation service

It runs as a separate service inside its own JVM.

Location of the log file:
`${VNMS_HOME}/logs/topologyLoader.error`
This file will contain only the errors which occurred while Topology Loader was running.

Location of the config file:
`${VNMS_HOME}/etc/config/TopologyLoader.xml`

Name of the java class file having main method for Topology Loader:
`com.vz.vpns.topology.loader.VPNSTopologyLoader`

1.3.9.2 Class/Package description

Some of the important packages/classes used by the Topology Loader:

- `com.vz.vpns.topology.loader`

  `VPNSTopologyLoader.java:`
  `Class having main method, initializes xslt, receives JMS message about Topology updates and sends it for processing.`

  `VPNSTopologyDataLoader.java:`
  `Loads metadata; does the actual processing of topology data like transformation and loading into DB.`

  `FileLoader.java:`
  `Takes care of loading of data from file system (initial batch dump), saving it for backup and inms.`

- `com.vz.vpns.topology`

  `TopologyLoaderXSLHelperBean.java:`
  `Helper class for stylesheet transformation. It is used for implementing some complex logic and storing some variables which can be used at different places in xslt.`

- `com.vz.vpns.baais`
  `Java classes required for establishing and maintaining JMS Topic connections as subscriber or publisher.`

- `com.vz.vpns.db`
  `Topology Database related classes, to update or retrieve information from DB.`

1.3.9.3 Starting/Stopping Topology Loader

Topology Loader can be started/stopped independently from the Unix console

- Unix Console:
  The command to start Topology loader is: `vnms –start topoloader`

The command to stop Topology loader is: `vnms –shutdown topoloader`

1.3.9.4 Inventory Data

Broadcast Video Inventory data as coming from BAAIS, constitutes the following major entities:
circuit name (CircuitId), names of nodes marked as VHO (Gige add point) or VSO (Gige drop point), cross-connect details of Cisco-6509 (or similar device) to VHO, cross-connect details of VSO to BigBand (or similar device), all the NEs (OLA or VSO) being used in the circuit to send signal from a VHO to VSO, for each NE all the shelf, card and port details which are being used, cross-connect details from one port to another port inside a VHO, VSO, OLA etc.
All this inventory data comes in the form of xml as per the schema which has been designed and distributed by Baais team.
As per current network plan, there would be two kinds of devices being used for Broadcast Video network:

- Tellabs-7100
- Flashwave-4500

Provisioning of TL-7100 devices:
In case of TL-7100 devices, provisioning of gige circuit typically goes through the following steps:

- First OC192 circuit is provisioned on Sonet network to create a virtual OC192 circuit that might span over more than one DWDM rings (VHO and VSO has been identified at this stage only, though more VSOs can be added later on as a separate supplementary gige orders)
- VHOs are provisioned for all the six Gige circuits
- All the six gige circuits are provisioned on top of the virtual OC192 circuit thus created

At each stage of provisioning, there are multiple cross connections done between various ports within a node and between different nodes. For each order, there will be one inventory xml generated which will flow to Surveillance systems through iView. Once all the orders are completed from Baais side, the next step is to activate the circuits thus created. This activation is done by eCO team. The xml which is generated before activation has circuit status as Pending and the ActionType of some the cross connections as A (to be added). The xml generated after activation has circuit status as InEffect and the ActionType of all the cross connections as W or R (working or repeat). So for each circuit created by Baais, Surveillance system would be getting two xmls, one for Pending state and another for InEffect state. (significance of Pending and InEffect status: when an alarm comes on an entity which is in Pending state, then it won't be passed to INMS and would be dropped in VPNS system itself).

Since there are in total six gige signals traversing on the virtual OC192 ring created (3 acting as primary and the remaining 3 acting as protect signals), so there would be at least six gige circuits created (for each VSO drop). For the OC192 circuit and each gige circuit provisioned, Surveillance system would get one xml with cross-connect details.

Provisioning of FW-4500 devices:
FW-4500 devices are pretty much different from TL-7100 devices in terms of provisioning OC192 circuit. They can handle only one wavelength so there is no concept of provisioning OC192 circuit first. On these devices, all the 6 giges are provisioned straight away.
Except the first step of provisioning 7100 device (as mentioned above), rest two steps are more or less the same.

1.3.9.5 Stylesheet Transformation (xslt)

Stylesheet serves the following purposes:

- extract necessary information from the topology xml to create entities and their relationships
- apply certain business logic to derive the required data
- transform the incoming Topology data xml into standard form so that it can be loaded in Topo DB
- putting some default values in some cases where proper data is not coming

The transformation logic is based on the topology data which we are getting from Baais/iView and the Topology model which we are using for VPNS project. All the entities and their relationships with other entities are created as per the model, during transformation.
A helper java class is used for implementing some complex logic of transformation and storing some variables which can be used at different places in xslt.
Location of helper java bean:
`com.vz.vpns.topology.TopologyLoaderXSLHelperBean.java`

"vpns_topo.xsl" is the main stylesheet, it uses one more xslt file which has got some utility functions named: utilities.xsl

Location of the stylesheet (xslt):
`${VNMS_HOME}/etc/xslt/vpns/vpns_topo.xsl`
`${VNMS_HOME}/etc/xslt/vpns/utilities.xsl`

Location of schema for target xml:
`${VNMS_HOME}/etc/xsd/Topology.xsd`
The xml obtained after xslt transformation should conform to this schema.

Note: - For more topology information please refer **chapter 6**

1.3.10 Admin Service

Functionality of Admin Service is as follows:-

- The main functionality of Admin Service of VPNS Admin GUI is to start and stop various Services viz. Correlation, Mediation, Topology Wrapper and accomplish various corresponding customized supported Command options.
- Admin Server needs to be started before starting any Service through command line.
- Select List of Management Command start or stop options to start/stop the corresponding Service.
- Each Service support many commands that are customized based on particular service pickup viz. for Correlation Service corresponding other commands are ReloadFilters, ReloadPosting, PurgeCashe, ReloadTrend/Threshold, InmsTicketsTransfer, ReconnectWeblogic etc.
- Status can be updated by pressing the Show/Refresh Service Status button.

1.3.11 Weblogic Server

As the JMS provider, Weblogic JMS setting will be configured as follows:

NetworkAccessPoint :

```
<NetworkAccessPoint ListenAddress="132.197.137.18"
   ListenPort="9020" Name="T3Channel"/>
```

JMS topics

```
<JMSTopic JNDIName="vnms.administration" Name="vnms.administration"/>
<JMSTopic JNDIName="vnms.webmessages" Name="vnms.webmessages"/>
<JMSTopic JNDIName="vnms.packageupdates" Name="vnms.packageupdates"/>
<JMSTopic JNDIName="vnms.topologyupdates" Name="vnms.topologyupdates"/>
<JMSTopic JNDIName="vnms.baaisUpdates"    Name="vnms.baaisUpdates" StoreEnabled="true"/>
<JMSTopic JNDIName="vnms.rawmsg"   Name="vnms.rawmsg"/>
```

JMSConnectionFactory

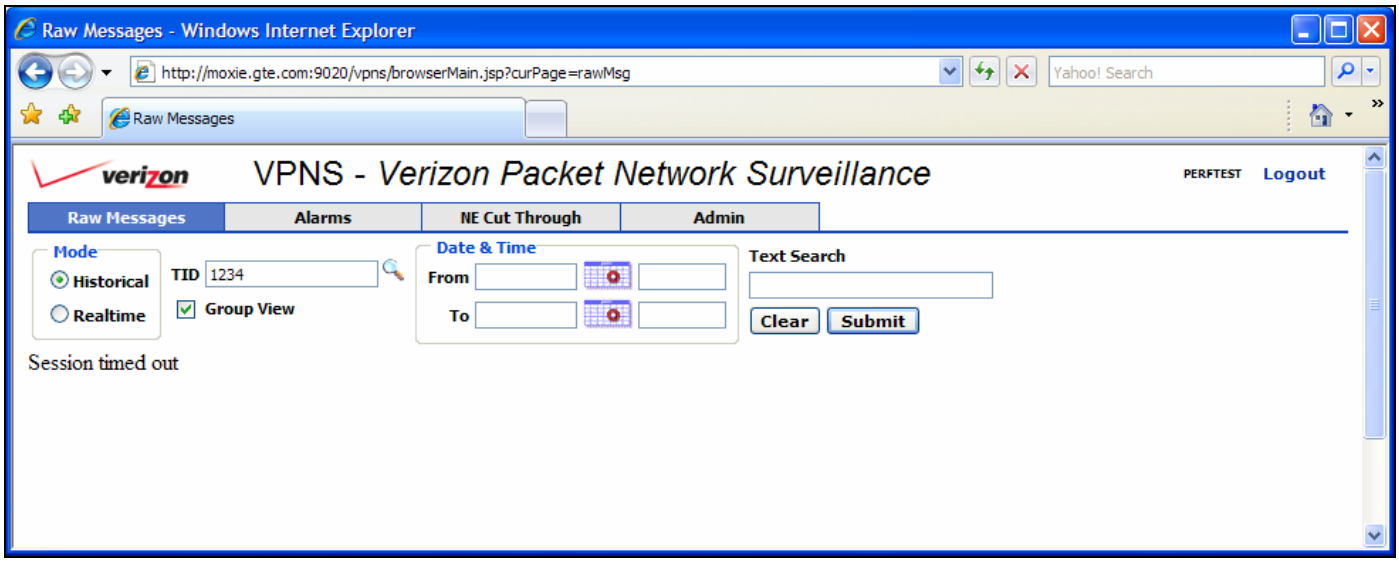
```
JNDIName="com.vz.vnms.jms.TopicConnectionFactory_1"
 Name="TopicConnectionFactory_1" />
```

To support durable JMS connection on topic “vnms.baaisUpdates" , a filestore is also needed.

In addition , Weblogic will host the VPNS web application: “Raw Message Browser”.

1.3.12 Raw Message Browser

Raw Messages - The Raw Message Menu item is be used to list the Alarms by Network Element.



Note: - For more details on VPNS Web GUI please refer *VPNS User guide*.

1.4 System Requirements

1.4.1 VPNS Web Client Requirements

In order for the application to run correctly make sure the computer meets a few basic requirements.

Hardware Requirements
Minimum recommended configuration:

- **Processor** - Intel Pentium II 333 MHz or greater
- **System RAM** - 128 MB or greater
- **Hard Disk Space** - 1 GB free disk space or greater

Operating System Requirements
The required operating system for VPNS is Windows XP professional SP1 or Windows 2000 professional SP4.

Java Program Requirements
The required Java Program for the VPNS application is the *Java 2 Runtime Environment, SE v1.4.2_X*.

Internet Browser Requirements
The required Internet browser for VPNS is *Microsoft Internet Explorer 6.0*.

2 External System Interfaces

2.1 BAAIS

2.1.1 Overview of Inventory data and BAAIS/iView teams

BAAIS is the team responsible for doing the provisioning of Broadcast Video circuits (VPNS). They get VPNS Circuit Orders to create/destroy new VPNS circuits or extend the existing circuits. Extension of circuits means adding new VSOs to the existing one. Implementing each order is a complex procedure which involves several tasks (related to Baais/iView/eCO) like: putting new NetworkElements (nodes) in the existing network or change the configuration of the existing NEs or both; changing or creating new cross-connections from one port to another port, within or outside the node. All these information constitute the Inventory which is required in VPNS system so that correlation can be done.

When the circuit is in the process of being provisioned, it is assumed to be in Pending state, as the circuit/network exists but none of the clients (customers) are being serviced. Surveillance system gets an xml file with all the Inventory details (as being provisioned for this order) but with circuit status as Pending. Once the provisioning is complete and the circuit has been activated (by eCO), it's the time when the clients start getting service from this circuit. At this time, the Surveillance team gets another xml which is similar to the previous one, but with Circuit status as InEffect. Most of the cross-connect information is being repeated in this xml.

2.1.2 Interface with Baais/iView (getting inventory data)

The inventory data flows from Baais system to iView. iView can send the data to VPNS system in two different ways:

- Initial batch dump
- Live Topology updates

2.1.2.1 Initial batch file dump

Initially when the VPNS system would go live, it is expected that some VPNS circuits would already exist. Inventory topology data for those existing circuits would be sent as batch file dump (by iView) through secure copy (SCP). Relevant xml files would be dumped at a designated location in VPNS file system. The topology loader would be continuously listening to the file system (at regular intervals) and would process any Inventory xml file as soon as it comes.
Location where batch files are dumped: `${BAAIS_HOME}/topodata/initialDump/`

2.1.3 Live Topology updates

Whenever a new circuit is provisioned or an existing circuit is modified, the live topology updates would be

sent to Surveillance system (by iView) in real time, through JMS messaging interface. A message would be published on the designated topic. The topology loader would be continuously listening to that topic and would process any xml message as soon as it is published.
Topic where JMS messages are published/received: `vpns.baaisUpdates`

2.1.4 Back up of Inventory Data files

All the inventory data files which are received by VPNS system (by secure copy or JMS message) are stored as backup for future reference.
The file naming convention being followed is: `BAAIS_GigECkt_<mmddyyyy>_<hhmmss>.xml`
Location where the backup xml files would be kept:
`${TOPO_HOME}/topodata/backup/<mmddyyyy>/`
where, `TOPO_HOME` is the home directory for "baais" account
`<mmddyyyy>` is the folder where every day's inventory file would be saved, one folder created for each day
`<hhmmss>` is date timestamp when the message was received by Topology loader

2.2 INMS

INMS adapter is a sub component of VPNS system that forwards alarm package to the remote INMS servers.

2.2.1 Alarm Feed

2.2.1.1 Live Alarm feed

VPNS server will forward alarm packages and their update information to INMS Adapter, which convert the alarm packages into XML messages and then send XML messages to INMS through JMS.

2.2.2 Resync Handler

VPNS updates alarm data to INMS using VPNS-INMS Adapter Live Alarm feed. If for some reason connection between VPNS and INMS goes down, Alarms will not be sent to INMS. In this situation of connection going down between VPNS and INMS, VPNS provides mechanism to update Alarms generated during downtime.

The following command is for creating XML dump of all ACTIVE alarms and to transfer the data to INMS using SCP:-

UnixPrompt> vnms -resync

To check if the process is running properly, the following steps need to be verified.
a. Goto fault directory VNMS_FAULT_DIR.
Default fault dump directory is /usr/lpp/vnms/inmsData/faultData

b. If the fault dump directory value is changed in install time, then go to the changed fault directory.

c. In order to find the location of the changed fault dump directory perform the following steps:-

Go to VNMS_HOME/version/etc/config/
Enter the following command:-
UnixPrompt> grep VPNS_FAULT_DIR= vnmsenv.

d. Following line will show the location of the fault dump directory:-
UnixPrompt> VPNS_FAULT_DIR=<dir_value_entered_during_install>

e. To check if the dump files are created
Go to dump directory location and execute the following command:-
UnixPrompt> ls -lt temp

If the timestamp is close to current timestamp +/- 2 min, dump is being processed.

2.2.3 Activate/De-activate INMS Adapter

Users can always suspend or resume the INMS Adapter to forward messages to INMS server without restarting VPNS server.

2.2.3.1 Resume the INMS Interface Adapter

The following command is used to resume the message forwarding process in INMS Adapter:-

UnixPrompt> vnms –resumeINMSFeed

If the logger level is set to “debug”, you should be able to see the message “livefeed is resumed manually” in ${VNMS_HOME}/logs/vnms.log, which verifies that the INMS interface adapter has been resumed properly.

2.2.3.2 Suspend the INMS Interface Adapter

The following is used to suspend the message forwarding process in INMS Adapter:-

UnixPrompt> vnms -suspendINMSFeed

If the logger level is set to “debug”, you should be able to see the message “livefeed is stopped manually” in ${VNMS_HOME}/logs/vnms.log, which verifies that the INMS interface adapter has been suspended properly.

2.2.4 Send Keep Alive Message

A keep-alive message distinguishes between “quiet” communication paths and communication paths that are not functioning properly. The VPNS application will send a keep-alive message to INMS if

VPNS has no data to forward to INMS for a period of time. The keep-alive message indicates that VPNS is active and online.

System Administrators can configure the time interval that VPNS will send the keep-alive message. The configuration interval is between 1 minute and 120 minutes. The default time will be 5 minutes.

When VPNS sends its keep-alive message, the message text will identify the VPNS host sending the keep-alive message. The keep-alive message should also carry a severity attribute of “Clear”

2.2.4.1 Config parameters for Keep Alive message

The following parameters are related to Keep Alive message:-

Parameters	Description	Default Value
keepalive.enable	Whether to keep alive messages or not	True
keepalive.interval	Time interval for a keep alive message to be sent (in minutes)	5
keepAlive.enhancedformat	Whether to build keep alive message in an enhanced format	False

2.2.5 Verify Keep Alive message

If the logger level is set to “debug”, you should be able to find keep-alive message sent to INMS in ${VNMS_HOME}/logs/vnms.log, for example:

msg [<?xml version="1.0" encoding="utf-8"?>

<AlarmPackage>

<Source>camilo</Source>

<EventTime>1168902611399</EventTime>

<ConditionType>KeepAliveNotification</ConditionType>

<Status>RESOLVED</Status>

</AlarmPackage>]

2.2.6 Store & forward Message

The INMS Adapter implements Store & Forward alarm message queues in-memory.

The hosting Web Logic JMS server on the INMS application servers will provide the ability to achieve reliable message delivery from Verizon-owned subtending systems to INMS.

VPNS employs in-memory Store & Forward queues and resend logic.

In the event connectivity is lost for hours, it is recognized INMS will experience a delay associated

with correlating the alarm packages as it will be processing the receipt of the alarm packages which are in a great number.

2.2.6.1 Config parameters for stored & forward

The following parameters are related to stored & forward:

Parameters	Description	Default Value
eventBufferSize	The maximum number of xml messages to store	1000
<JMS instance>. eventBufferSize	The maximum number of xml messages to store for a particular JMS instance	1000

2.2.6.2 How to handle buffer overflow

It is possible that message arrival rate is faster than message leaving rate. As a result, the event buffer will overflow and some messages will get lost. In this case, the INMS adapter will (1) generate a special "Event Buffer Overflow" message to send to INMS if the connection between VPNS and INMS is alive. (2) Otherwise, the following error message will be logged into vnms.log if the VPNS-INMS connection is down:

buffer overflow while JMS connection is unavailable, will not buffer events anymore. Upon reconnection resync may be triggered

In either case, a resync should be done to capture lost messages.

2.2.7 Verify the start and end of stored & forward process

If the logger level is set to "debug", the following messages which show the start and end of stored&forward process should be found in ${VNMS_HOME}/logs/vnms.log file:

Lose JMS connection, start to store message!
Regain JMS connection, start to send stored message!

2.2.8 Topology Loader interface with INMS

Whatever Inventory files which VPNS system receive from Baais/iView, are sent to INMS, as they also need to update their topology Database. Topology Loader sends the Inventory files coming through initial dump as well as JMS messages, using the same interface, in real time. Loader saves a copy of the inventory data (file or msg) in a designated location in file system with same naming convention as followed for backup copies. A Cronjob running at regular interval transfers all the xml file present in the designated directory to a particular location on INMS server through secure copy (SSH and SCP).

2.2.8.1 Cronjob details

Cronjob invokes a Unix shell script which does all the transferring work of inventory files. The shell script logs into INMS box using SSH and do the transfer using SCP. Transfer is a two step process; it first copies the xml file at a temporary location on INMS box and then moves that file to the final target location.
Typically cronjob entry looks like:

```
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58
*  *  *  *
/home/vnms/vnms_version/bin/inventoryFileTransferToINMS.sh
```

first line tells that the cronjob would be running every 2 minutes. In order to make it run every 10minutes, change it to something like this: `0,10,20,30,40,50`
second line (* * * *) tells that job would be running every hour, every day, every month and every year.
last line (/home/vnms/vnms_version/bin/inventoryFileTransferToINMS.sh) gives the absolute location of the Unix shell script which will be executed.

2.2.8.2 inventoryFileTransferToINMS.sh

This shell script has got all the details to do the SCP file transfer from VPNS file system to INMS box. This script assumes that the "`VNMS_HOME`" parameter is being setup in the `.profile` file of "`vnms`" account and the config file "`vnmsenv`" is present at the designated location. It executes "`vnmsenv`" file to set up some of the parameters.
Location of this script: `/home/vnms/vnms_version/bin/inventoryFileTransferToINMS.sh`
Location of the log file created by this script:
`/home/vnms/vnms_version/logs/inventoryFileTransferToINMS.log`

Couple of other important parameters which are being set in this script are:

```
SCP_SSH_BINARY_HOME:
Gives the location of binary files: scp and ssh, set up during installation and stored
in vnmsenv file.
SRC_DIR_FILE_DUMP:
Source location for Inventory data files on VPNS file system (set in this script)
INMS_HOST: IP address of INMS box (set up in vnmsenv)
INMS_USER: User name for INMS box (set in this script)
INMS_TEMP_DIR: Temporary directory where files would be copied on INMS box (set in this
script)
INMS_DIR: Final target location for file transfer in INMS box (set in this script)
```

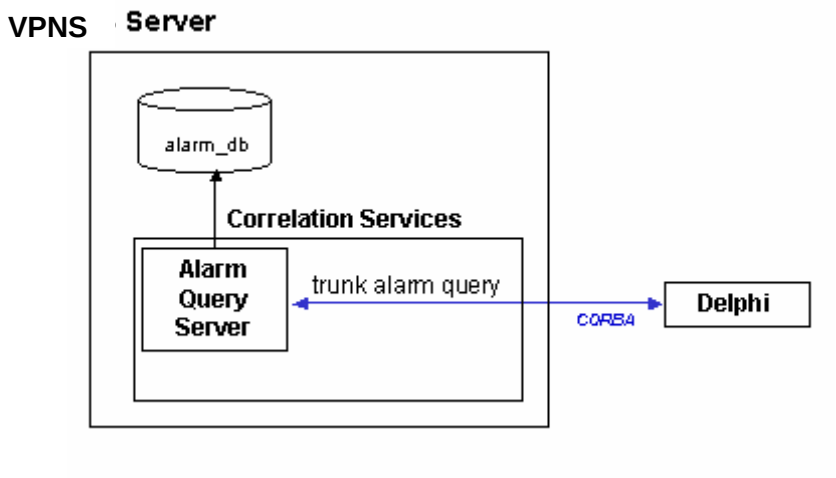
These parameters can be changed if required, at the place where they have been defined.

2.3 Delphi

VPNS provides an interface for DELPHI, Verizon's Circuit Testing System. Delphi provides reactive testing for VoIP services through verification of real and virtual circuits, connectivity to routers and switches, and review of SoftSwitch alarms. Delphi queries the VPNS Alarm Database for trunk alarms. A Delphi query provides a passage id IOR and network equipment information, such as TID/CLLI, shelf, slot, or port etc. VPNS either returns a list of alarms or an error message indicating a processing error, input error, or missing fields. VPNS allows up to five simultaneous queries of up to six query groups per request.

2.3.1 Delphi Interface Data Flow

Figure below shows the flow of data for the Delphi interface. Requests from Delphi are received by the Alarm Query server in the VPNS Correlation Services and forwarded to the VPNS Alarm Database. The query response is then passed back to Delphi via the VPNS CORBA connection. Response time is not configured in the VPNS environment.



2.3.2 Configuring the Delphi Interface

To set up this interface, send alrmQuery.ior from both the Primary and Standby Servers to the Delphi Production Support team. This can be done by either e-mail, ftp, or Sametime. The file to send to the Delphi Production Support team is located in:
/usr/lpp/vnms/vnms_version/logs/alarmQuery.ior

2.3.3 Verifying Delphi Data Queries

VPNS supplies data on demand. The requesting Delphi service verifies the receipt of data. The AlarmQueryServer logs Delphi transfer errors in the vnms.log located in `$VNMS_HOME/logs/`. No logging occurs for successful queries.
The following list is a preliminary summary of Delphi error messages:

- -201- Invalid Input XML. Could be non-conforming to DTD or not well formed.
- -204 - Internal: Error while transforming Document object to XML string.
- -304 - Internal: Error accessing database and/or some other SQL error.
- -701 - Required field is missing from QueryGroup.
- -702 - More than maximum supported QueryGroups sent in the request.
- -703 - Too many simultaneous queries in progress.
- -704 - Less than 1 valid QueryGroup sent in request.
- -705 - More than 1 FilterGroups in the request.
- -706 - More than 1 SortGroups in the request.
- -707 - QueryGroup Prop value not conforming to the contract.
- -708 - FilterGroup Prop value not conforming to the contract.
- -709 - SortGroup Prop value not conforming to the contract.

2.3.4 Delphi Query Parameters

The following properties can be used in querying alarms:
• **PackageId** - Required for querying alarms based on a package ID.

• **Clli** - The soft switch CLLI which the component belongs to. Required if for querying alarms based on equipment information.

• **Tid** - Required for querying alarms based on equipment information.

• **ShelfId** - Shelf II. Required for querying alarms on equipment at CARD or lower level. Can be ignored if it is not applicable.

• **SlotId** - Slot ID. Required for querying alarms on equipment at CARD or lower level. Can be ignored if it is not applicable

• **PortId** - Port ID. Required for querying alarms on PORT or lower level. Can be ignored if it is n- Trunk group ID, required for querying trunk group alarms. Alarms can be filtered by start time, end time, and severity. If a start time is not specified, then alarms are provided for the most recent 72 hours. They can be sorted by maximum rows up to 250, primary sort, and secondary sort.

The following properties are used in the query response:
• **PackageId** - Package Id to which this alarm contributed

• **clli** - Required. The soft switch clli which the component belong to.

• **componentID** - Original component identifier from which the alarms were raised

• **AlarmSource** - One of the following: CS2000, PP8600, PP15000, PVG, SAM21, GWC, WAS, RTP_Portal, JUNIPER_M40, MG9000, MCS5200, USP, CICM, NWKSVR, APPSVR, VMAILSVR, and SBC.

• **Tid** - Target identifier of the Network Element that reported this alarm.

• **ShelfI** - Shelf Id of the equipment that reported this alarm. Ignored if it is not applicable.

• **SlotId** - Slot Id of the equipment that reported this alarm. Ignored if it is not applicable.

• **PortId** - Port Id of the equipment that reported this alarm. Ignored if it is not applicable.

• **TrunkGroupId** - Trunk Group Id.

• **AidType** - Access Identifier type of the alarm.

• **Aid** - Aid of the alarm.

• **GenerationTime** - Required. The time this alarm is generated. String representation of

Unix Time (GMT) in milli seconds.

- **Severity** - Required. Severity of the alarm.
- **Status** - Required. Status of the alarm.
- **Service Affect** - Required. Service Impact of the alarm.
- **ConditionType** - Required. Condition Type of the alarm.
- **Description** - Required. Description of the alarm.
- **Direction** - The direction of Alarm

3 VPNS GUI

Refer to the latest version of *VPNS Client User Guide* for complete details on how to work with the VPNS GUI.

4 Tools

4.1 Scripts

4.1.1 inmsInventoryTransfer.sh

This script reads BAAIS data feeds and FTPs them to INMS. It is set up during installation to run automatically.

4.1.2 updateDBStats.sh

This script can be run manually or as a cron job. It optimizes any database and has three levels:

- **High** - Provides high optimization but can only be run when the database is offline.
- **Medium** - Provides good optimization and can be run while the database is online.
- **Low** - Provides quick optimization and can be run while the database is online.

Verizon IT recommends that the script be run at the low level daily and medium level weekly.
Run it at the command line as follows:

```
${VNMS_HOME}/bin/updateDBStats.sh -d <database name> <level; high, medium, or low>
```

4.2 Server Commands

Login as vnms and use these commands at /home/vnms/vnms3.0 or VNMS_HOME:
Start Server: vnms -start [all|vnms|medsvc|topowrapper|namesvc|notifsvc]
By Default vnms server starts in dual mode and recovers alarm and package data.

NOTE: vnms -start all will launch any vnms process that is not running. Since vnms -shutdown all does not shut down name service, you will get a message that they are already running.

Shutdown Server: vnms -shutdown [all|vnms|medsvc|topowrapper|namesvc|notifsvc]

NOTE: vnms -shutdown all will shut down all services except name service.

List Services: vnms –list

Run Alarm Cleaner: vnms -alarmJanitor

Load Topology Meta Data: vnms -loadTopologyMetaData

Run Package Cleaner: vnms -packageJanitor

DB uname/passwd change: vnms -dbuser

Start AdminClient: vnms -adminClient [(r)eload [filter|mapping|package|posting|threshold|trending|medsvc]

Release Information vnms -version

Help vnms -help

4.3 MPE Debugger

The MPE debugger is used to debug pattern files used by the Mediation Service. It performs the following functions:

- Verifies pattern syntax
- Steps through the operations of any given pattern, viewing the current string with the cursor position indicated.
- Sets and clears breakpoints to stop the pattern execution before any pattern operation by specifying the line number, and the number of operations on the line.

  Note: - Refer MPE **chapter 10** for more information on using the MPE debugger.

5 Correlation Service Rules

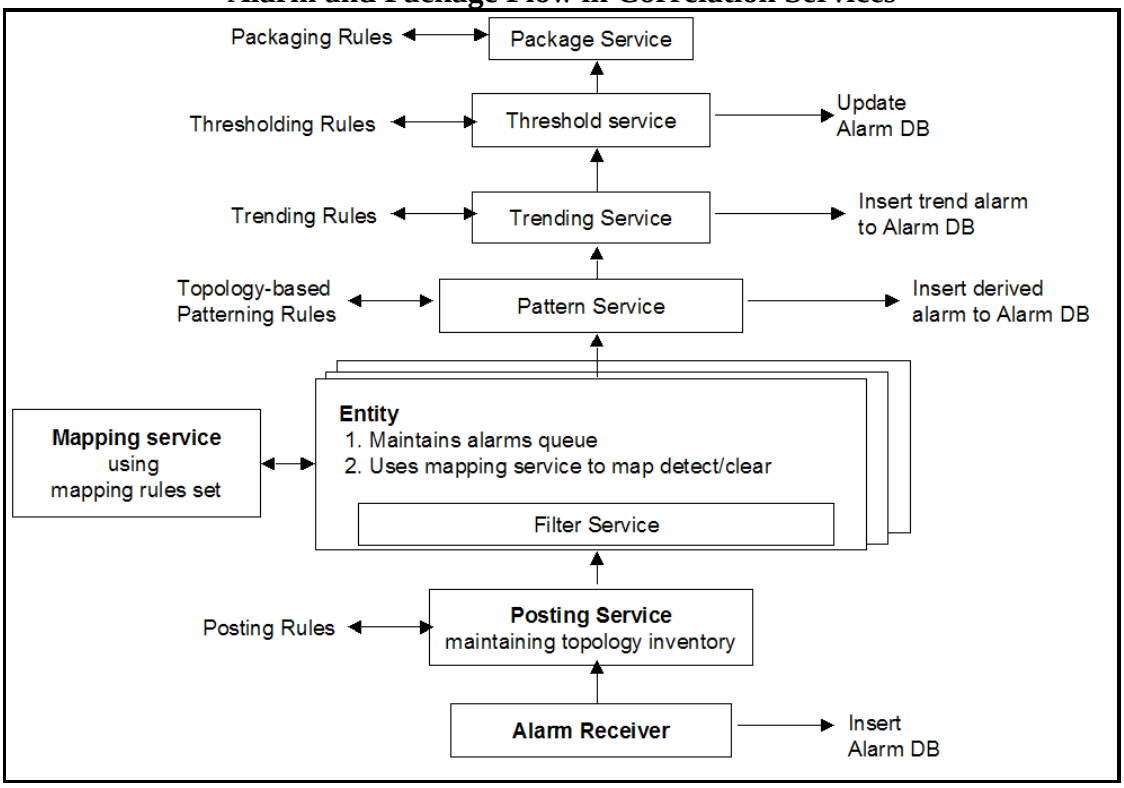
5.1 Overview

Correlation rules are user-specified rules written in XML to handle correlation services such as: alarm filtering, posting, mapping, trending, thresholding, and packaging.
A rule is a set of variables and an action that triggers when an alarm or a package meets these conditions. Each rule may have a set of variables, such as:

- rule-name, rule-cond-type, rule-entity-type, rule-desc
- Regex (the alarm regular expression)
- Entity types
- Condition types

The Correlation Services handle alarm posting, filtering, mapping, patterning, trending, Thresholding, and packaging according to the user-specified correlation rules. This chapter provides instructions and syntax examples to create and edit correlation rules The locations of the XML rule files are listed in VNMS.par located in $VNMS_HOME/etc/xml.
Following figure shows the general flow of alarms and packages as they are processed by the Correlation Services using correlation rules.

Alarm and Package Flow in Correlation Services



As you can see by the diagram, there are sets of rules that guide the processing of alarms into packages. The following Correlation Services are associated with a set of rules:

- Posting
- Filtering
- Mapping
- Patterning
- Trending
- Thresholding
- Packaging

5.2 Correlation Services Life Cycle

The following alarm analysis services (in descending progression) make up the Correlation Services life cycle:

Table: Correlation Services Life cycle

Correlation Services/ File names	What is done at this step in lifecycle?
Mediation	Alarm from Mediation Service is assembled into a generic format to be used with Correlation Services.
Posting Service	Identifies the affected network entity (AE) from the alarming entity.
Filtering Services	The Alarm on the affected entity is evaluated to determine whether it needs to be filtered out. **Masking** - Specified alarms are deleted or cleared
Mapping Service	Alarm Mapping determines if one alarm is a clear alarm and maps this clear alarm to one or more detect alarms.
Trending Service	**Trending** - detects the trends in the alarms such as the ***bouncing condition*** (where the alarms form the same source and same condition type are generated and cleared a number of times within a set time frame) or ***multiple detects*** (where multiple alarms from the same source and same condition type are generated within a given time frame) and places all the related alarms in one trending package. All alarms in a trending package have the same source and same condition type. **Autorestore** - a rule-based expert system automatically handles normally repeated events that have passed masking test. **Thresholding** - Specifications of alarm frequency and severity determines whether molars are hidden or further processed.
Thresholding Service	Alarms are held for a configurable time period (5-20 seconds), allowing the opportunity for them to be cleared by the Correlation Services. After the time period has expired, remaining alarms are sent to the Correlation Package Service.

Package Service	Root cause alarm is identified and subsequent-related alarms are associated to this package. Aggregate related alarms to a single work item, (alarm 1…. alarm n).

5.3 Topology Caching

When the Topology database is queried for an alarm, the topology data is then cached in memory so that subsequent queries on the same device can re-use this data without wasting processing time querying the Topology database again.
VPNS automatically checks the Topology cache every ten minutes to remove any topology entities that are not attached to a current alarm.

5.4 Posting Rules

The Correlation Services Alarm Receiver uses the Correlation Posting Service to assign a new alarm to an Entity. The task of moving the alarms through the VPNS Correlation Services is performed by the Entity thread. Examples of Entities include: port, card, shelf, or NE. Each references the source or location of the alarm.

Given an alarm from a reporting entity, the Posting Service determines which entity the alarm came from and identifies the corresponding affected entity based on the alarm field or fields combination specified in the Posting Rule.

The posting rule can specify on which entity the alarm will be posted. The posted entity can be different with the actual entity the alarm came from, it can be the parent entity, supported entity… or the actual entity. If VPNS can not find an exact entity in the Topology database, it will try to post on its immediate parent.

If there are no posting rules that can be used to post the alarm, the posting service will try to post the alarm on NE (top level entity) using neClli field. If the top level entity still can not be found, the alarm will be posted on a UNKNOWN entity.

5.4.1 Posting Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Posting Rules directory:

```
cd ~vnms/vnms_version/etc/xml/posting
```

2. Open the file that contains the posting rule information:

```
vi PostingRules.xml
```

The Posting Service Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file. Edit or Change the necessary fields in the new rule segment Contact Verizon IT Support for more details on the syntax to be

followed.

4. Save the file and exit vi.

5. Apply the Posting Rules through the syntax below

```
vnms -adminClient reload posting
```

Notes:

- *The **cond_type** and **regex** are criteria to match an alarm to this rule.*
- *There can be 0 ~ any of **regex**.*
- *The **field** value is not case sensitive.*
- *The **pattern** value is not case sensitive*
- ***cond_type** is an optional criteria, it can be a list of several condition types separated by ",", this can be specified if the condition type is really known, this way can improve the speed of posting.*
- *The **rule_name** is an optional attribute.*
- *The **report_entity** can be "sid"(clli), "compid" (compID),"loc" (location), "aid" (future use), "subaddr" (subAddress), or any combination of these attributes, such as "$sid+$compid"*
- *The **poston_entity** value can be "entity"(iteself), "parent" (parent entity), "ne"(NE entity), "shelf" (shelf entity),"slot"(slot entity), "port" (port entity).*

5.4.2 Posting Rule Example

An example of a file segment listing is below:

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE rules SYSTEM "PostingRules.dtd">

<rules>

      <posting-rule rule_name="default"
      report_entity ="$sid+$compid"
      poston_entity ="this.entity">
      <regex field="type" pattern="1"/>
      </posting-rule>
</rules>
```

5.4.3 Posting-Rules-DTD Syntax File

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE rules SYSTEM "PostingRules.dtd">
<rules>
 <posting-rule rule_name="default"
      report_entity ="$aid"
      poston_entity ="this.entity">
         <regex field="type" pattern="1"/>
```

```
    </posting-rule>
</rules>
```

5.5 Filtering Service Rules

Alarm filtering excludes the processing of specific alarms based on their attributes. The Filtering Service filters alarms matching them to the filtering rules. If an alarm is filtered out, it will be stored in the Alarm Database and no longer processed by VPNS.

The user can search for alarms and packages stored in the Alarm Database from the Search Menu in the VPNS Web GUI.

5.5.1 Filter Service Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Filtering directory:

```
cd ~vnms/vnms_version/etc/xml/filter
```

2. Open the file that contains the Filtering Rule information:

```
vi filter.xml
```

The Filtering Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file. Edit or change the necessary fields in the new rule segment.

4. Save the file and exit vi.

5. Apply the Filtering Rules through the syntax below:

```
vnms -adminClient reload filter
```

5.5.2 Filter Service Rule Example

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE filterlist SYSTEM "filter.dtd">
      <rules>
      <alarmfilter rule_name="filter_1">
                      <regex field="severity"       pattern="4"/>
                      <regex field="condType"     pattern="^(?!T-[1-9a-zA-Z-]*$)"/>
        </alarmfilter>

               <alarmfilter rule_name="filter_2">
                             <regex field="condType"       pattern="^\d*$"/>
                          <regex field="desc"           pattern="^\d*$"/>
        </alarmfilter>

        <alarmfilter rule_name="filter_3">
                             <regex field="ent.status"     pattern="^1$"/>
        </alarmfilter>
      </rules>
```

5.5.3 Filtering Rules - DTD Syntax File

```
<?xml version="1.0" encoding="us-ascii"?>
<!ELEMENT rules (alarmfilter*)>
<!ELEMENT alarmfilter (regex*,fromTimeStamp?,toTimeStamp?) >
<!ATTLIST alarmfilter

  rule_name               CDATA #IMPLIED
>

<!ELEMENT regex EMPTY >
<!ATTLIST regex

 field (condType|conditionType|neClli|Clli|aid|aidType|direction|dir|location|loc| d
escription|desc|alarmSource|type|serviceAffect|svcAft|severity|compID|compo
nentID|subNeAddress) #REQ UIRED  pattern  CDATA #REQUIRED >
<!ELEMENT fromTimeStamp EMPTY >
<!ATTLIST fromTimeStamp
value                 CDATA #REQUIRED
> <!ELEMENT toTimeStamp EMPTY >
```

5.6 Mapping Rules

Alarm Mapping determines which alarm is a clear alarm and maps this clear alarm to one or more detect alarms.

5.6.1 Mapping Service Scenarios

- When an alarm is received, it will check all the “clear-alarm-regex” section fields in mapping rules. If there is a matched rule, it will be regarded as a clear alarm. Otherwise, it will be a detect alarm.
- A detect alarm will be saved to the “processedAlarm” queue on Entity to matching clear alarm later on. At the same time, the detect alarm will be sent out to the next service.
- A clear alarm will not be saved to the “processedAlarm” queue, and will not be sent out.
- If the detect alarms in the “processedAlarm” queue matched the incoming clear alarm, new alarms will be cloned from these detect alarms and the cloned alarm “state” will be set to “cleared” (Simulates a cleared event.)
- If there are no matched detect alarms, the clear alarms will be discarded at this point.

5.6.2 Mapping Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Mapping directory:

```
cd ~vnms/vnms_version/etc/xml/mapping
```

2. Open the file that contains the Mapping Rule information:

```
vi Matchingrules.xml
```

The Mapping Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file. Edit or change

the necessary fields in the new rule segment.

4. Save the file and exit vi.

5. Apply the Mapping Rules through the syntax below

```
vnms -adminClient reload mapping
```

Notes:

- *The **rule_name** is optional, it is used for user to easily position a rule.*
- *The **entityType** is optional, if this field is presented, the mapping performance can be improved.*
- *The **field** value is not case sensitive.*
- *The pattern value is not case sensitive*
- *The same field can be used for several patterns*
- *<**detect-alarm-regex**> section can exist or not.*
- *<**clear-alarm-regex**> section can exist not*
- *<**equal-fields**> section can exist or not*
- *Only the detect alarm and clear alarm come from the same entity type can be matched, otherwise, they are not regards as detect/clear pair.*

5.6.3 Mapping Rule Example

```
</detect-alarm-regex>
        <clear-alarm-regex>

            <regex field="severity" pattern="5"/>
        </clear-alarm-regex>
        <equal-fields>

            <field name="conditionType"/>
             <field name="compID"/>
             <field name="aid"/>

        </equal-fields>
    </matching-rule>

</rules>
```

```
MatchingRules.xml: END
```

5.6.4 Mapping Rules - DTD Syntax File

```
<!ELEMENT entity_type EMPTY> <!ELEMENT detect-alarm-regex (regex*)> <!ELEMENT
clear-alarm-regex (regex*)> <!ELEMENT equal-fields (field*)> <!ELEMENT regex
EMPTY> MatchingRules.dtd (62%)


<!ATTLIST matching-rule
          rule_name                CDATA  #IMPLIED
>
    <!ATTLIST entity_type
 Value CDATA  #REQUIRED
```

5.7 Pattern Rules

The Pattern Service creates derived pattern alarms according to contributing alarm and pattern rules, based on the alarms and topology information.

It can specify which entity the patterning will operate on, the patterning entity itself, parent entity, or a supported entity.

Pattern rules can be used to specify maximum alarms, minimum alarms, or percentages of alarms. This feature can be quite useful when handling many entities. For example, one pattern can specify the maximum number of alarms from several specified entities that would be correlated into a derived alarm. Another pattern can specify a percentage of alarm entities that would be correlated into a derived alarm, while a third pattern would specify the minimum alarm entities required to generate a derived alarm.

5.7.1 Pattern Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Trending directory:

```
cd ~vnms/vnms_version/etc/xml/pattern
```

2. Open the file that contains the Pattern Rule information:

```
vi pattern.xml
```

The Pattern Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file, before the tag "</tpattern_rule>". Edit or change the necessary fields in the new rule segment.

4. Save the file and exit vi.

5. Apply the Pattern Rules through the syntax below

```
vnms -adminClient reload pattern
```

Notes:

- *The **entity_type,cond_type**, and **regex** are criteria to match the alarms to this rule.*
- *The **entity_type** and **cond_type** are optional, if they are not present, the alarm will only try to match **regex**.*
- ***regex** can be 1~ any.*
- *The **pattern-alarm** section specify the new pattern alarm attributes.*
- *The **rule_name** attribute is optional*
- *The **rule_entity_type** attribute specifies on which entity the patterning operation will occur. This can be itself, parent, support entity of the entity where contributing alarms came from.*
- *The **correlation_relation** specifies the correlation type such as: self, parent, child_support, or parent_support.*
- *The **exclude_pending** attribute specifies whether the pending entities will be counted into the criteria for generate new pattern alarms*
- *The **min_num_contrib_entities** attribute specifies the minimum number of contributing entities for generating new pattern alarms*
- *The **pattern_gen_percentage** attribute specifies on which percentage of contributing entities are present, then the pattern alarms will be generated.*
- *The **pattern_clear_percentage** attribute specifies on which percentage of contributing entities are present, then the pattern alarms will be cleared.*
- *The **field** value is not case sensitive.*
- *The **pattern** value is not case sensitive*

5.7.2 Pattern Rule Example

```
rule_entity_type = "gwc"xml/pattern> ls
correlate_relation="child_support"
exclude_pending="y"/vnms5.0/etc/xml/pattern> more pattern.xml
min_num_contrib_entities="2"
pattern_gen_percentage="70"
pattern_clear_percentage="5" >
<pattern_alarm
```

```
    rule_cond_type="GWC_BROKEN"
 severity="1"
 svc_affect="y"
    rule_desc="GWC 22 primary and backup unit were all down"
     /> <regex field="type" pattern="1"/>
     <regex field="conditionType" pattern="GWC302"/>
    </pattern_rule> </rules>
```

5.7.3 Pattern Rules - DTD Syntax File

```
     <?xml version='1.0' encoding='us-ascii'?>

     <!--          DTD          for          pattern          templates          -->
     <!ELEMENT             rules             (             pattern_rule*             )>


<!ELEMENT pattern_rule ( entity_type?, cond_type?, pattern_alarm, regex* )>

<!ATTLIST pattern_rule
             rule_name              CDATA    #REQUIRED
          rule_entity_type          CDATA    #REQUIRED
               correlate_relation      (

                              self|parent|child_support|parent_support ) #REQUIRED
  exclude_pending     ( y|n ) "y"    min_num_contrib_entities  CDATA #REQUIRED
     pattern_gen_percentage      CDATA    #REQUIRED
    pattern_clear_percentage    CDATA    #REQUIRED
>
<!ELEMENT entity_type EMPTY>
<!ATTLIST entity_type

       value            CDATA #REQUIRED > <!ELEMENT cond_type EMPTY> <!ATTLIST cond_type
       value            CDATA #REQUIRED > <!ELEMENT pattern_alarm EMPTY> <!ATTLIST pattern_alarm
       rule_cond_type      CDATA #REQUIRED
       severity            ( 1|2|3|4 ) #REQUIRED
       svc_affect        ( y|n ) "y"
       rule_desc         CDATA #REQUIRED >

<!ELEMENT regex EMPTY> <!ATTLIST regex
       field            (condType|conditionType|neClli|Clli|aid|aidType|direction|dir|location|l
```

oc|description|desc|alarmSource|type|serviceAffect|svcAft|severity|compID|c

omponentID|subNeAddress) #REQUIRED pattern CDATA

#REQUIRED

5.8 Trending Rules

Trending detects the trends in the alarms such as bouncing condition (where the alarms form the same source and same condition type are generated and cleared a number of times within a set time frame) or multiple detects (where multiple alarms from the same source and same condition type are generated within a given time frame) and places all the related alarms in one trending package. All alarms in a trending package have the same source and same condition type.

The Trending Service can use **autorestore** (autoclear) to reduce alarms by matching problem messages with messages that pair with the problem message and clearing them without passing them to the VPNS monitoring technician. It can use **thresholding** to count certain alarms that contain specified strings.

Alarms that are not cleared are passed to the Thresholding Service and also sent to the Alarm Database.

When the Trending Service receives an alarm:

- If there are no rules matched to this alarm, the alarm (detect/clear) will be sent out to Threshold and return
- If there is a rule matched to this alarm, the trending service will check the “forwarding” flag in rule, if it is true, the alarms will be sent out to threshold as well.
- If there is a rule matched to this alarm, the trending service will roll back “timer” period to count the related alarms whose receive time is within this period. The alarm out of this period will be removed.
- If there are no alarms, every 1 minute, the Trending service will check the alarm count for the last 1 minute period, if the number is less than count1, a new clear trending alarm will be generated.
- If the “countPair” in rule is true, the trending service will count the detect/clear pair number; if the “countPair” is false, just simply count the detect alarm number
- If the count meets the rule count (gen_count/raise_severity_count), generate a new trending alarm
- If the count does not meet the rule count (clear_count), clear all the new trending alarms.

5.8.1 Trending Service Scenarios

- If there is no rule-matched alarm, then the alarm (detect/clear) will be sent out to Thresholding Service and returned.
- If there is a rule-matched alarm, then follow the procedure below: **1.**Find the proper queue and insert Detect Alarm into it.

2.Roll back "timer" period to check the alarm count during this period. An alarm not from this "timer" period will be removed. If "forwarding" flag in rule is false, the alarms within the queue of the entity will be removed also.

3. If the "trending_type" is bouncing, check the number of clear_ids in all alarms, that is the pair num; if the "trending_type" is multidetect, just simply check the alarm number.

4. If the count meets the rule count (gen_count/raise_severity_count), generate a new trending alarm if there is no trending alarm yet.

5. If the count does not meet the rule count (clear_count), clear the new trending alarm if it exists.

6. Reset the "time to check" timer (current time + sliding window).

- Idle check thread - The thread will check the "time to check "every some time (sliding window / 3), if the "time to check "is arrived, check all the alarm queues.

5.8.2 Trending Service Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Trending directory:

```
cd ~vnms/vnms_version/etc/xml/trending
```

2. Open the file that contains the Filtering Rule information:

```
vi trendingrule.xml
```

The Trending Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file. Edit or change the necessary fields in the new rule segment.

4. Save the file and exit vi.

5. Apply the Trending Rules through the syntax below

```
vnms -adminClient reload trending
```

Notes:

- *The Trending **rule_name** should be unique.*
- *There are no default trending rules.*
- ***gen_count** and **raise_severity_count** can be "-1" to set the feature invalid.*
- ***entity_type/cond_type** are optional.*
- ***cond_type** is optional. If user does not know the condition type, it can be ignored. If you do not know the condition type, you can put the condition pattern in the alarm-regex section. Using the exact condition type on **cond_type** can improve the processing performance.*

- ***cond_type*** *can be a list of condition types, separated by ",".*
- ***regex*** *gives the user more criteria flexibilities to define more attributes beyond entityType and conditionType, such as description, etc. These fields also can also be empty.*
- ***trending_alarm*** *gives the user capabilities to define the trending alarm attributes.*

5.8.3 Trending Rule Example

```
      <?xml version="1.0" encoding="us-ascii"?>
      <!DOCTYPE rules SYSTEM "trendingrule.dtd">


      <rules>

<trendingrule rule_name="7100-tca-1"
                  trending_type = "multidetect"
                  trend_clear_count="4"
                  trend_gen_count="6"
                  raise_severity_count="-1"
                  timer_window="120m">
         <cond_type value="T-BCPKT"/>
         <regex field="severity" pattern="1|2|3|4"/>
         <trending_alarm rule_cond_type="T-BCPKT-trend"
                         rule_desc="T-BCPKT Trending Alarm"/>
    </trendingrule>

   </rules>
   trendingrule.xml: END
```

5.8.4 Trending Rules - DTD Syntax File

```
      <!-- DTD for trending service templates -->



      <!ELEMENT rules (trendingrule*)>
      <!ELEMENT trendingrule (entity_type?,cond_type?,regex*,trending_alarm?)>
      <!ELEMENT entity_type EMPTY>
      <!ELEMENT cond_type EMPTY>
      <!ELEMENT regex EMPTY>
      <!ELEMENT trending_alarm EMPTY>



      <!ATTLIST entity_type
```

```
            valueCDATA#REQUIRED
      >
      <!ATTLIST cond_type

        valueCDATA#REQUIRED > <!ATTLIST trendingrule rule_name CDATA  #REQUIRED
trending_type (multidetect|bouncing)#REQUIRED trend_clear_countCDATA #REQUIRED
trend_gen_countCDATA #REQUIRED raise_severity_countCDATA #REQUIRED timer_windowCDATA
#REQUIRED >

<!ATTLIST regex
        field
(condType|conditionType|neClli|Clli|aid|aidType|direction|dir|locati
on|loc|description|desc|alarmSource|type|serviceAffect|svcAft|severity|comp
ID|componentID|subNeAddre ss) #REQUIRED    patternCDATA #REQUIRED >

<!ATTLIST trending_alarm rule_cond_type CDATA#REQUIRED rule_desc CDATA#REQUIRED >
```

5.9 Thresholding

The Thresholding Service holds alarms for a configurable time period (5-20 seconds), allowing the opportunity for alarms to be cleared by the Correlation Services. After the time period has expired, remaining alarms are sent to the Correlation Package Service. The Thresholding rule is associated with a default rule. It assigns this rule if it cannot find a rule for a given entity, type, subtype, and condition.

The Threshold Service holds the detect alarms for a certain time period which is varied according to matched rule. During the hold period, if the clear alarm comes, the detect alarm will be cleared and will not be displayed in client. After the hold period, if the clear alarm still has not come, the detect alarm will be send to the Package Service for packaging. If the clear alarm comes after the hold period, the clear alarm will also be sent to package service for possible auto resolving.

5.9.1 Thresholding Rule Instruction

1. Go to the VPNS Thresholding directory:

```
cd ~vnms/vnms_version/etc/xml/threshold
```

2. Open the file that contains the threshold rule information:

```
vi threshold.xml
```

The Threshold Rule file appears. Each file segment represents one rule.

3. Copy an existing rule segment and paste it at the end of the file.
4. Change the necessary fields in the new rule segment.
5. Save the file and exit vi.
6. Apply the Thresholding rule:

```
vnms -adminClient reload thresholding
```

Notes:

- *There is a default trending rule. If the rule name is "default" or "", then that is a default rule.*
- ***th1Time** is the time to hold the alarm before it was sent to client.*
- ***regex** gives user more criteria flexibility to define more attributes, for example, entityType, conditionType, description, etc.*
- ***entity_type** is optional, the default is "NE"*
- ***cond_type** is optional. If user does not know the condition type, the user can ignore this part. If the user does not know exactly the condition type, the user can put the condition pattern in the alarm-regex section. But, using the exact condition type on rule_cond_type can improve the processing performance.*
- ***cond_type** can be a list of condition types, separated by ",".*

5.9.2 Thresholding Rule Example

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE rules SYSTEM "threshold.dtd">
<rules>

<threshold rule_name="th-sts" th1Time="3s">
        <regex field="aidType" pattern="STS1|STS3C|STS12C|STS48C"/>
                <regex field="severity" pattern="1|2|3"/>
                <regex field="condType" pattern="^(?!CABL_LOS$)"/>
    </threshold>
</rules>
threshold.xml: END
```

5.9.3 Threshold Rules - DTD Syntax File

```
<?xml version='1.0' encoding='us-ascii'?>
<!-- DTD for threshold templates -->
<!ELEMENT rules (threshold*)>
<!ELEMENT threshold (entity_type?,cond_type?, regex*)>
```

```
      <!ELEMENT entity_type EMPTY>
      <!ELEMENT cond_type EMPTY>
      <!ELEMENT regex EMPTY>

                                  <!ATTLIST threshold


       rule_name                          CDATA  #IMPLIED
    th1Time             CDATA   #REQUIRED
>
<!ATTLIST
entity_type
        value           CDATA   #REQUIRED
>
<!ATTLIST cond_type
        value           CDATA   #REQUIRED
>
<!ATTLIST regex

        field
```

(condType|conditionType|neClli|Clli|aid|aidType|direction|dir|locati
on|loc|description|desc|alarmSource|type|serviceAffect|svcAft|severity|comp ID|componentID|subNeAddress)
#REQUIRED

pattern CDATA #REQUIRED > threshold.dtd: END

5.10 Packaging Rules

When an alarm enters the Package Service, the Package Service determines whether the alarm should be associated with a pre-existing Affected Entity (AE) package or a new package should be created. In either case, the alarm is dispatched via JMS to the PON Gateway in the Application Server. The PON Gateway translates the alarm package into a structure that can be recognized by the Package Monitor in the VPNS Web GUI Client. The user can then analyze and work with the package, including the resolving it. When a request to resolve a package is sent back through the PON Gateway to the Package Service, the Entity Alarm Queue is updated and the queue updates the thread.

Root cause alarm is identified and subsequent-related alarms are associated to this package. Aggregate related alarms to a single work item, (alarm 1…. alarm n).

The Package Service acts as the event feeder to the Application Server, it will send packages information and alarms information to application server via the CORBA interface. The Package Service also can receive request from northbound interface to manually resolve an acknowledged package. The Package service will trace the alarms for a certain package, when the alarms' number associated to a certain package decreased to 0, the package service will issue auto-resolve request to application server.

5.10.1 Packaging Service Rule Instruction

To add a new rule or edit an existing rule and then apply the rule:

1. Go to the VPNS Packaging directory:

```
cd ~vnms/vnms_version/etc/xml/packaging
```

2. Open the file that contains the Packaging Rule information:

```
vi packaging.xml
```

The Packaging Rule file appears. Each file segment represents one rule.

3. To add a new rule, copy an existing rule segment and paste it at the end of the file, before the tag "</packagingrules>". Edit or change the necessary fields in the new rule segment.

4. Save the file and exit vi.

5. Apply the Packaging Rules through the syntax below:

```
vnms -adminClient reload packaging
```

Notes:

- *The **entity_type** and **regex** are criteria to match alarms to this rule.*
- *The **entity_type** is optional, and **regex** can be 1 ~ any.*
- *The **package_details** gives user more flexibilities to define the new attributes for the package.*
- *The **rule_entity_type** attribute specifies which entity the package will created on, it can, it can be itself, parent entity or support entity of the contributing entities.*

5.10.2 Packaging Service Rule Example

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE rules SYSTEM "packaging.dtd">
<rules>

        <packagingrule rule_name="package_rule_1"
        rule_entity_type = "NE"
        rule_desc = "Fault description">
        <entity_type value="NE"/>
        <package_details
            pkg_subtype=""
            auto_tt="n"

      svc_affect="y" /> <regex field="conditionType" pattern="CON1"/>
        </packagingrule>
```

```
 </rules>
packaging.xml: END
```

5.10.3 Package Rules - DTD Syntax File

```
      <?xml version='1.0' encoding='us-ascii'?>

      <!-- DTD for package service templates -->

      <!ELEMENT rules (packagingrule*)>
      <!ELEMENT packagingrule (entity_type?,cond_type?, package_details?,regex*)>
      <!ELEMENT entity_type EMPTY>
      <!ELEMENT cond_type EMPTY>
      <!ELEMENT package_details EMPTY>
      <!ELEMENT regex EMPTY>


      <!ATTLIST packagingrule

            rule_name                  CDATA  #REQUIRED
              rule_entity_type     CDATA    #IMPLIED
               rule_cond_type      CDATA    #IMPLIED
            rule_desc              CDATA    #IMPLIED
    >

<!ATTLIST entity_type
      value  CDATA  #REQUIRED >

<!ATTLIST cond_type
      value  CDATA  #REQUIRED >

<!ATTLIST package_details
      pkg_subtype          CDATA #REQUIRED
      auto_tt              ( y|n ) "y"
          svc_affect ( y|n ) "y" >

      <!ATTLIST regex     field
       (condType|conditionType|neClli|Clli|aid|aidType|direction|dir|location|l
      oc|description|desc|alarmSource|type|serviceAffect|svcAft|severity|compID|c omponentID|subNeAddress)
```

```
    #REQUIRED
pattern CDATA #REQUIRED>
```

5.11 Fault Package Detail Information Alarm Packages Triggered by Multiple Alarms

Tabs	Fields	Notes
Basics	Id	Package Identifier
	Description	Package description; text determined by patterning and/or correlation rules
	Condition Type	Condition of package specified in alarm
	Status	Status of entity on which the package was created
	Package Type	Entity type (e.g., NE, Card, Connection)
	Package Sub Type	Sub Type of connecton type package
	Rule Name	Rules used in Fault Package
	Category	Category of Fault Package - Regular, Rogue, Loss of Visibility
	Service is Affected	Service is (Yes) or not affected (No)
Fault Information	Network Condition Start Time	Package Start Time
	Network Condition End Time	Package End Time

Tabs	Fields	Notes
vRepair Ticket Info	ID	Ticket Number
	Created By	ID of person who created ticket
	System Created in	System ticket was created in (VPNS, vRepair, APSC Remedy)
	Creation Time	Time trouble ticket was created
	Status	Status of ticket (open or closed)
	Last Modified By	ID of last person who modified the trouble ticket
	System Last Modified In	System ticket was last modified in (e.g., VPNS)
	Last Modified Time	Time the vRepair was last modified
Network Entity	NE CLLI	Network Entity CLLI code
	NE Type	Type of Network Entity
	Slot	Slot Number
	State	State where NE resides
	Shelf	Shelf Number

In a Correlation scenario (the rules are defined and Correlation is enabled) to create a Package triggered by and/or consisting of a multiple of alarms in a specified time frame:

Q: What happens when some, but not all of the contributing alarms clear?

A: The Package will be kept till the package is resolved manually by user or auto-resolved (all of contributing alarms are cleared).

Q: What happens if the rule specifies or requires certain alarms and/or an alarm sequence in a certain time frame. Does the package get created? How “tightly” is the rule defined? For example, Confidence setting: Trigger correlation or clear if 75% is met.)

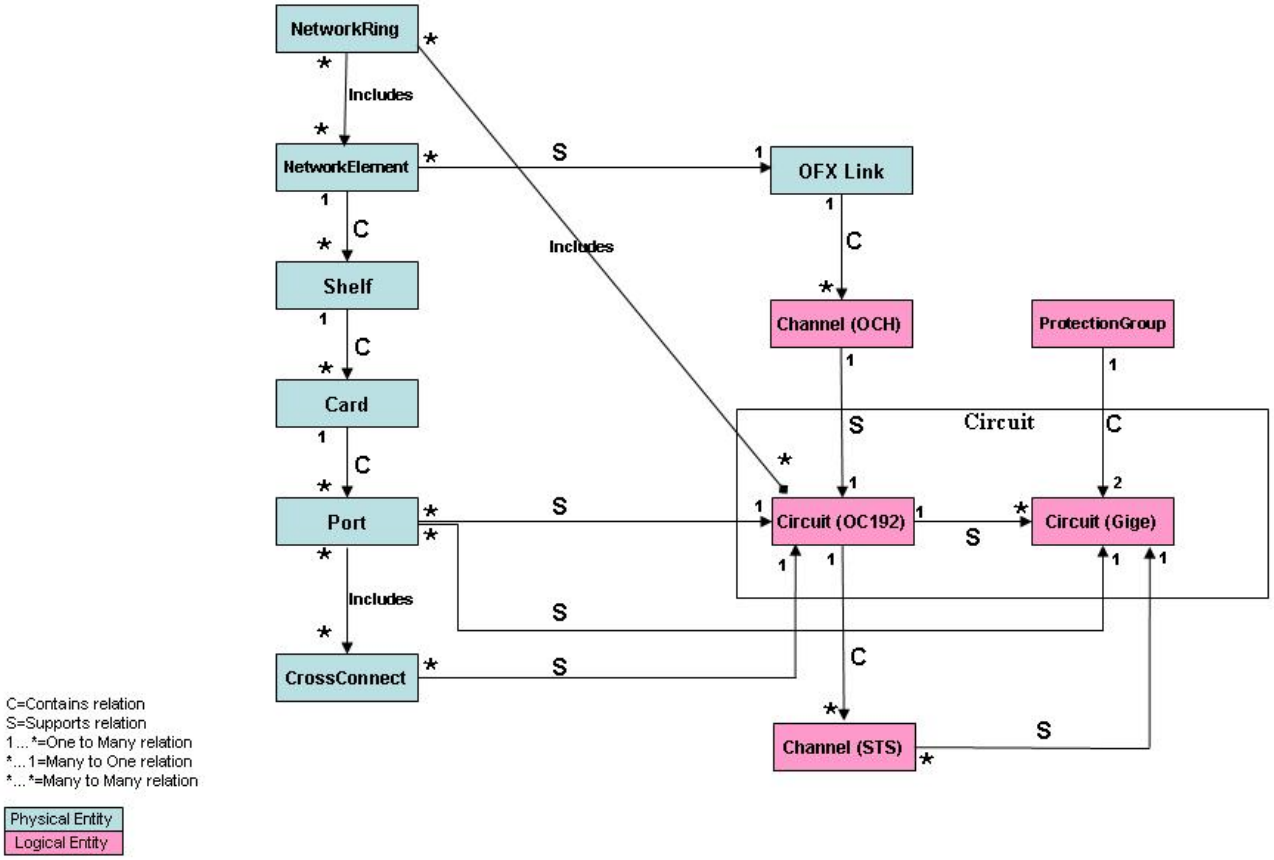
A: For creation of Correlated or Derived alarm, the user can define the percentage. But for package creation, it is a different behavior. The package rule definition only specifies which alarms will be grouped together. There is no percentage concept (for example, if only one contributing alarm comes, it also will create a package).

Q: What happens to package creation if alarms get dropped.

A: If not all contributing alarms were dropped, the package will be kept.

6 VPNS Topology

6.1 VPNS Topology Model (V3)



6.2 Explanation for the Topology Model

- There are three different types of relations used in this model, to relate various entities:
  - ***Contains*** – one to many relation
  - ***Supports*** – many to one relation
  - ***Includes*** – many to many relation

- ***Sonet Network*** – This entity represents the physical SCID/DWID/Sonet ring which can contain one or many Network Elements. This is different from the virtual OC192 ring provisioned for VPNS circuit.

- ***OFX Link*** – This entity represents physical fiber (facility) connection between two Network Elements in a SCID. There will be two such entity created between two adjacent NEs for either direction.

- ***NE, Shelf, Card, Port*** – These are the actual physical entities present in the network.

- ***CrossConnect*** – This represents the Cross connection between different types of card within a Network Element like cross connection from SMTM-U card to RCMM card etc.

- ***Channel*** – There are two types of channel entities defined:
  - ***OCH Channel*** – This represents the Optical channel (lambda) which is used to carry the video signal for that particular VPNS circuit (typically lambda1 to lambda44).
  - ***STS Channel*** – It is the STS channels which are concatenated to form different types of signal like OC192, OC48 etc. There will be one separate entity created for each STS channel number.

- ***Circuit*** – There are two types of Circuit entities defined:
  - ***OC192 Circuit*** – This represents the virtual OC192 circuit which is created over one or more Sonet rings, carrying only OC192 signal.
  - ***VPNS Circuit*** – This represents the VPNS circuit used for Video Broadcast. It has two flavors: One is the working and other one is protection.

  The model shows two Circuit entities, but essentially both are same entity with some change in attributes to distinguish them. Both are shown separately to clarify the relations which are created with them. When FW 4500 and other devices would be introduced which doesn't support OC192 signal then we would create some other type of circuit entity like OC48 circuit etc.

- ***ProtectionGroup*** – It represents the VPNS level protection which is provided for each VPNS circuit (Vid1, Vid2 and Vid3). Both Working and Protection VPNS circuit will be associated with this entity. There will be one such entity for each Working VPNS circuit.

- There will be one OCH Channel entity created for each Sonet ring. Since the name of the channel contains OC192ckt name so eventually all the channels created for each ring are going to have same name, unless there is different lambda number used in different ring.

All the VPNS Add and Drop ports will have supporting relation with its VPNS Circuit, while the remaining ports will be related to OC192 Circuit.

7 Configuration and Log Files

7.1 Overview

The following are VPNS Configuration files:-

- **AlarmSim.xml, TestAlarms.txt** – Generic alarm simulator will use the parameters defined in these files.
- **AlarmSkulker.xml** – Alarm Skulker will use this file to specify the behavior of clearing old alarms from database.
- **DbConnPoolKey.prop** – Define the parameters of db connections
- **MediationSvce.xml** – Contains all the configurable parameters that are required to run Mediation Service. It configures various subsystems of Mediation Service via JMS, JDBC, logging etc; they are different standard services which Mediation Service requires for communicating with various sub-components of Mediation Service.
- **Simulator.properties , simulator-users.xml , ne.txt :** TL1 simulator will use these files for configuration.
- **TopologyLoader.xml** – Contains config details to receive data from BAAIS using JMS transfer the messages received from BAAIS to Topology DB.
- **VpnsInmsAdapter.xml** – Contains details about connection to INMS like JMS server, port and topics, log files, store and forwards parameters, message buffer over flow details, and keep alive message parameters.
- **VpnsInmsResync.xml** – Contains config details to create alarms that have to be sent to INMS for re-synchronization, log file name, resync data file name, resync data path.
- **config.xml** – Defines all variables that are either used by multiple services, default source/target, host server details, JMS and JDBC details.
- **vnms.xml -** Contains details about log files, reference to xml files, thread pool details for the server, and reference to INMS Adapter config files.
- **tl7100.f** – MPE Pie Grammar for Tellabs 7100 devices. It also contains filter rules to be used by mediation service to drop the unwanted alarms.

- **fw4500.f** – MPE Pie Grammar for FW-4500 devices. It also contains filter rules to be used by mediation service to drop the unwanted alarms.

- **vnmsenv** – Contains environment variables.

7.2 Local Configuration Files

- **config.par.local** – Contains values for properties that has to be over-ridden. This file overwrites the parameters available in a standard configuration file.

7.3 Log Files

The following are the VPNS Log files:-
All the below log files are located in *$VNMS_HOME/version/logs.*

- **AdminServer.log** – It is enabled in AdminService.xml. It collects log entries for the Administration Service.

- **mediationSvce.log** – It is enabled in MediationSvce.xml. It collects log entries for the Mediation Service.

- **TopologyLoader.log** – It is enabled in TolologyLoader.xml. It collects log entries for the Topology Loader Service.

- **vnms.log** – It is enabled in vnms.xml. It collects log entries from correlation service.

- **vnmsINMSResync.log** – It is enabled in vnmsINMSResync.xml. It collects log entries while executing VPNS – INMS fault re-synchronization utility.

Each service and utility has a log file associated with it. The name of the log file and other parameters can be set in the configuration file associated with that service.

Example:

```
<!-- ======================================================== -->
<!-- Logging related PARAMETERS -->
<!-- ======================================================== -->

<property name="log.file.name">
      <value>${VNMS_HOME}/logs/vnms.log</value>
</property>
<property name="log.file.maxSize">
      <value>15360</value>
</property>
<property name="log.file.maxNum">
      <value>5</value>
</property>
```

```
        <property name="log.file.append">
              <value>false</value>
        </property>
        <property name="log.level">
              <value>${log_level}</value>
        </property>
        <property name="log.api">
       <value>LOG4J</value>
 </property>
<!-- Accepted values are file and stdout -->
<property name="log.type">
  <value>file</value>
</property>
```

To limit the amount of data that can be logged, there is a size limit which can be configured using log.file.maxSize. To make it easy for loading the file into file editors, maxSize is kept to a reasonable value. But to enable the logging of enough data, the data is logged to multiple files, each capable of holding the maxSize amount of data. The number of connected files is restricted by the log.file.maxNum parameter. Each time a process is spawned, a choice can be made either to overwrite the previous logfile or append to it. The current log file does not have any prefix after "fileName.log"

The above parameter settings will leave the following log files:
vnms.log
vnms.log.1

The two parameter of log file are:-
log.file.maxNum – Allows the user to specify the number of log files.

log.file.maxSize – Allows the user to limit the amount of data that can be logged.
To make it easy for loading data to file editors maxSize is limited to a reasonable value but logged to multiple files, each capable of holding the maxSize data.

7.3.1 Log Levels

Log levels of different services can be independently set in the config.par.local file to one of the following: -

- **Error** – Indicates severe problem that must be addressed by the technician or referred to Verizon IT.
- **warn** – Indicates a problem that must be addressed but will not prevent the service from continuing to function.
- **info** – Indicates the status of a service event.
- **debug** – Used by developers, not intended for customer use.

The standard log entry format is as follows: -
<timestamp> <level type> [service]: <log entry>

Example:
04/18/2005 03:55:45 ERROR [com.vz.vnms.services.db.AlarmHandler]: SQL Error getting the write connection

7.4 Parameters / Values

Property File Syntax

For XML file the parameter name is defined under property tag and Value is under Value tag.

Eg: <property>
....
</property>

If it is a plain text configuration file, Eg: *.par or *.par.local then parameter & value are defined as name, value, pairs.

Parameters

VPNS parameters should be configured during the VPNS software installation procedure. For detail information, please refer to the appendix files.

8 Database Schema

The VPNS surveillance system has three databases:

- **Topology** - This database contains real-time topology information about the entities present in the network. These entities are involved in providing Video Broadcast service. Information received from Baais is processed as per the Topology Model and business requirements, and stored in the format which is suitable for other VPNS services. Only the relevant information from xml is extracted, rest all are discarded.
  This database is an **unbuffered database** which means that when a commit is executed, the data is written to the table space immediately and replication occurs immediately. The Topology Database is used by the Correlation Services and the Mediation service.

- **Alarm** - This database is ANSI-compliant and contains up-to-date information about all alarms and alarm packages being processed in the VPNS system. It is continually updated by the Correlation Services and the Package Monitor. This database is a **buffered database** which means that when a commit is executed the data is written to the tablespace immediately but replication does not occur until the buffer specified by Informix is full. The database is kept in a maintainable mode by running the AlarmJanitor script daily and the PackageJanitor script monthly.

- **Raw Message** - This database contains all raw messages received or retrieved by the VPNS Mediation Service. The database can be configured to store messages up to 30 days. The Message Browser in the VPNS Web GUI can search and view raw messages in this database.

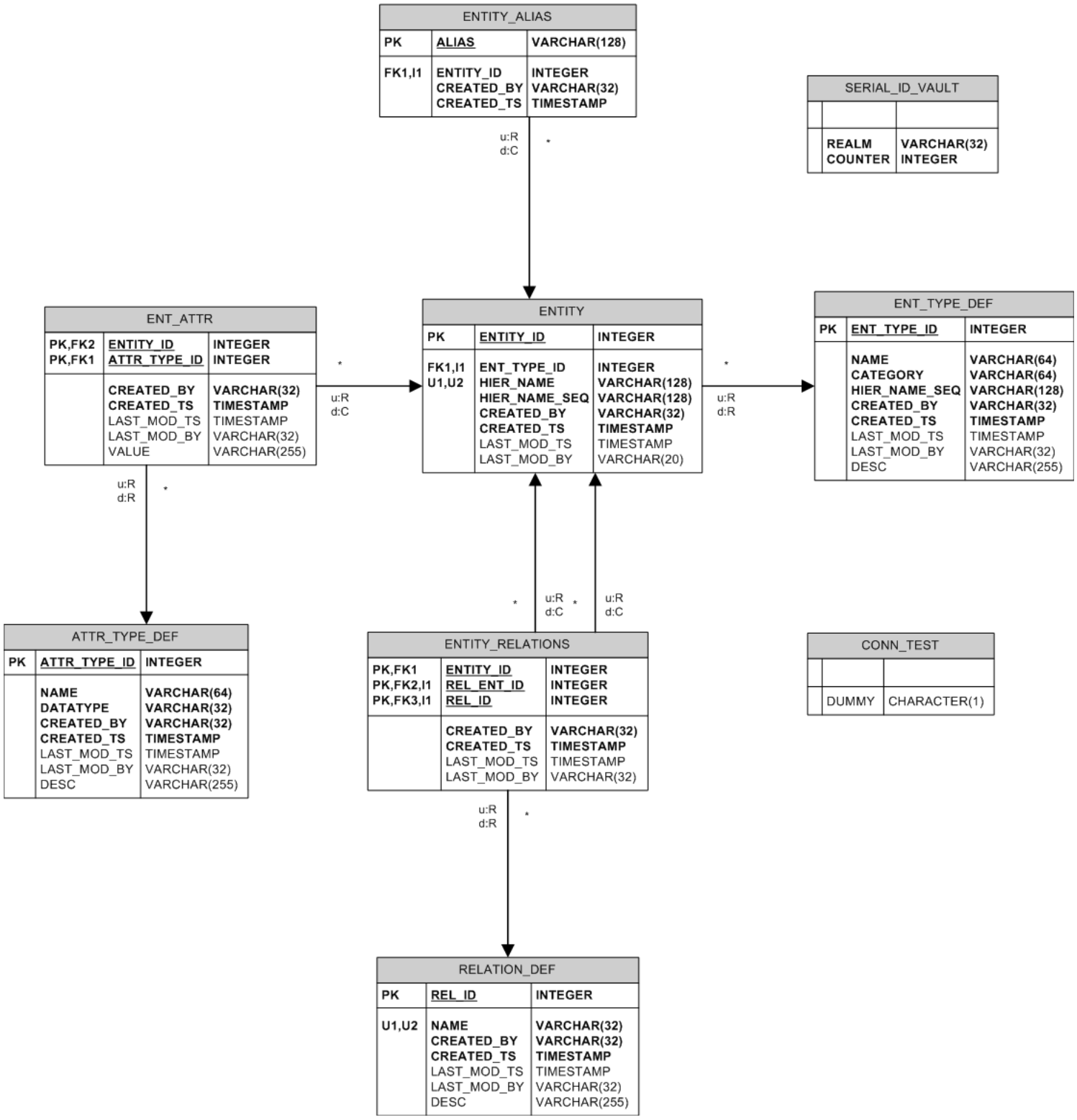
8.1 Topology Database

VPNS Topology Database is the place where all network topology related information is stored. This information is mainly used by Mediation service to connect to the devices in the field and by the correlation service to do posting and correlation. The schema is same as being used in other surveillance systems like Tnhs, S4, Ponnms. There are static and dynamic tables in the DB:

- **Static tables:**
  These tables contain topology metadata like what kind of entities are present in the DB, what are the valid relationships and attributes. These are defined in the code as xml files, one file for each table and are loaded when the topology loader starts up. This is the list of static tables:
  ent_type_def, relation_def, attr_type_def
- **Dynamic tables:**
  Dynamic tables are populated with real topology data as coming from Baais. These do check the availability of metadata in static tables. Following is the list of dynamic tables present:
  entity, entity_alias, ent_attr, entity_relations, serial_id_vault

Even if there is any change in our topology model or if there is some new requirements coming into picture or some new devices being used in the network, there shouldn't be much change in the DB schema. Maximum change which is expected is in the metadata, which doesn't affect the schema.

As you view the topology schema shown here, you will note that key relationships are marked with a small **r** for restricted and **c** for cascade. Indexes are indicated by **I<integer>**.
Figure below illustrates the generic Topology Database for this release.


ENTITY_ALIAS
PK ALIAS VARCHAR(128)
FK1,I1 ENTITY_ID INTEGER
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
u:R
d:C
*
SERIAL_ID_VAULT
REALM VARCHAR(32)
COUNTER INTEGER
ENT_ATTR
PK,FK2 ENTITY_ID INTEGER
PK,FK1 ATTR_TYPE_ID INTEGER
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(32)
VALUE VARCHAR(255)
u:R
d:C
*
ENTITY
PK ENTITY_ID INTEGER
FK1,I1 ENT_TYPE_ID INTEGER
U1,U2 HIER_NAME VARCHAR(128)
HIER_NAME_SEQ VARCHAR(128)
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(20)
u:R
d:R
*
ENT_TYPE_DEF
PK ENT_TYPE_ID INTEGER
NAME VARCHAR(64)
CATEGORY VARCHAR(64)
HIER_NAME_SEQ VARCHAR(128)
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(32)
DESC VARCHAR(255)
u:R
d:R
*
ATTR_TYPE_DEF
PK ATTR_TYPE_ID INTEGER
NAME VARCHAR(64)
DATATYPE VARCHAR(32)
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(32)
DESC VARCHAR(255)
*
u:R
d:C
*
u:R
d:C
ENTITY_RELATIONS
PK,FK1 ENTITY_ID INTEGER
PK,FK2,I1 REL_ENT_ID INTEGER
PK,FK3,I1 REL_ID INTEGER
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(32)
CONN_TEST
DUMMY CHARACTER(1)
u:R
d:R
*
RELATION_DEF
PK REL_ID INTEGER
U1,U2 NAME VARCHAR(32)
CREATED_BY VARCHAR(32)
CREATED_TS TIMESTAMP
LAST_MOD_TS TIMESTAMP
LAST_MOD_BY VARCHAR(32)
DESC VARCHAR(255)

Following is the detail description of each of the tables, their fields and datatypes present in the topology database.
Naming convention being followed:

PK = primary key
FK = foreign key
I<integer> = index

8.1.1 ENTITY

This is the most important table which stores all the topology entities.

ENTITY			
Column type	**Column Name**	**Data Type**	**Description**
PK	ENTITY_ID	INTEGER	Unique ID for any topology entity.
FK,I1	ENTITY_TYPE_ID	INTEGER	Type of entity (as per metadata in ent_type_def table)
U1,U2	HIER_NAME	VARCHAR(128)	Unique name assigned to this entity (as derived from the topology data).
	HIER_NAME_SEQ	VARCHAR(128)	Explanation of the format of the Hierarchal name of this kind of entity such as <ne * shelf>.
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(20)	Name of the person/system who last modified this entity.

8.1.2 ENTITY_ALIAS

For each entity, there would be at least one alias created (through triggers). There might be more than one alias for any entity. All such aliases of all the entities are stored here.

ENTITY_ALIAS			
Column type	**Column Name**	**Data Type**	**Description**
PK	ALIAS	VARCHAR(128)	Another name identifying this entity.
FK,I1	ENTITY_ID	INTEGER	Unique ID for any topology entity.
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.

8.1.3 ENT_ATTR

All the attributes/properties of entity is stored here. Any entity can have any attribute as long as the attribute is defined in the metadata (attr_type_def table). The list of possible attributes for any particular entity is defined in the metadata xml (EntityTypeDefinitions.xml)

ENT_ATTR			
Column type	**Column Name**	**Data Type**	**Description**
PK,FK2	ENTITY_ID	INTEGER	Unique ID for any topology entity.
FK,FK1	ATTR_TYPE_ID	INTEGER	Type of attribute/propery (as per metadata in attr_type_def)
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(32)	Name of the person/system who last modified this entity.
	VALUE	VARCHAR(255)	Value for this attribute as set in AttributeTypesDefinition.xml located in /etc/xml/memtadata/

8.1.4 ENTITY_RELATIONS

This table creates relation between two different entities.

ENTITY_RELATIONS			
Column type	**Column Name**	**Data Type**	**Description**
PK,FK1	ENTITY_ID	INTEGER	Unique ID for any topology entity (parent entity in the relationship)
FK,FK2,I1	REL_ENT_ID	INTEGER	Unique ID for any topology entity (child entity in the relationship)
PK,FK3,I1	REL_ID	INTEGER	Type of the relationship (as per metadata in relation_def)
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(32)	Name of the person/system who last modified this entity.

8.1.5 ENTITY_TYPE_DEF

Static table to store different types of entity which are allowable in the topology DB.

ENTITY_TYPE_DEF			
Column type	**Column Name**	**Data Type**	**Description**
PK	ENT_TYPE_ID	INTEGER	Unique ID assigned to the entity type.
FK,I1	NAME	VARCHAR(64)	Name of the entity type.
	HIER_NAME_SEQ	VARCHAR(128)	Explanation of the Hierarchal name format of this entity type.
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.

	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(32)	Name of the person/system who last modified this entity.
	DESC	VARCHAR(255)	More detailed textual description of this entity type.

8.1.6 ATTR_TYPE_DEF

Static table to store different types of attributes/properties of any entity which are allowable in the topology DB.

ATTR_TYPE_DEF			
Column type	**Column Name**	**Data Type**	**Description**
PK	ATTR_TYPE_ID	INTEGER	Unique ID assigned to the attribute type.
	NAME	VARCHAR(32)	Name of the attribute type.
	DATATYPE	VARCHAR(32)	Explanation of the data format of the attribute type.
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(32)	Name of the person/system who last modified this entity.
	DESC	VARCHAR(255)	More detailed textual description of this attribute type.

8.1.7 RELATION_DEF

Static table to store different types of relationships which are allowable in the topology DB.

RELATION_DEF			
Column type	**Column Name**	**Data Type**	**Description**

PK	REL_ID	INTEGER	Unique ID assigned to this entity relationship.
U1,U2	NAME	VARCHAR(32)	Name of this entity relationship.
	CREATED_BY	VARCHAR(32)	Name of the person/system who created this entity in the database.
	CREATED_TS	TIMESTAMP	Date and time when this entity was created in the database.
	LAST_MOD_TS	TIMESTAMP	Date and time when this entity was last modified.
	LAST_MOD_BY	VARCHAR(32)	Name of the person/system who last modified this entity.
	DESC	VARCHAR(255)	More detailed textual description of this entity relationship.

8.1.8 SERIAL_ID_VAULT

This table acts as counter for other records which would be inserted in the DB. It has one record for entity, where it stores the next entity_id which can be used for inserting new entities in entity table. Every time a new entity is inserted, this counter is increased by one.

SERIAL_ID_VAULT			
Column type	**Column Name**	**Data Type**	**Description**
	REALM	VARCHAR(12)	Stores table names ***for VPNS system use only.***
	COUNTER	INTEGER	ID for next table record***for VPNS system use only.***

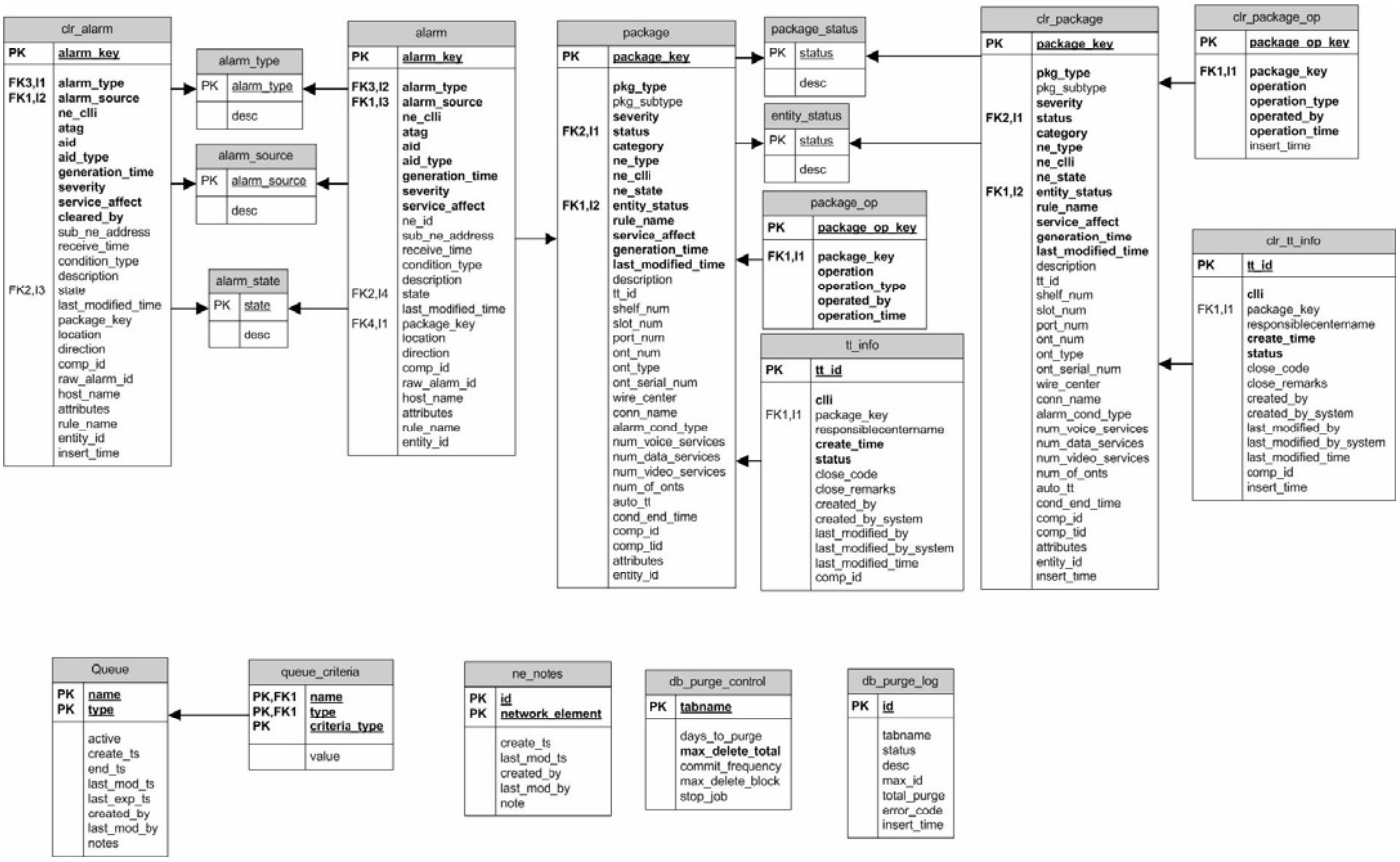
8.1.9 CONN_TEST

CONN_TEST			
Column type	**Column Name**	**Data Type**	**Description**
	DUMMY	CHARACTER(1)	

8.2 Alarm Database

The Alarm Database contains up-to-date information about all alarms and alarm packages being processed in the VPNS system and continually updated by the Correlation Services, Application Server, and through user operations in the Package Monitor.

The Figure shows the relationships among the Alarm Database tables. The four colored boxes show the database views include schema. Tables that include “op” in the title are updated automatically by the Application Server or manually through the Package Monitor. The other tables are modified automatically by the Correlation Services.



The following tables describe the database fields.
Table relationships and indices are indicated as follows:

PK = primary key
FK = foreign key
I<integer> = index

This table is populated by the Application Server and can be modified automatically by the Application Server or manually through user operations in the Package Monitor.

8.2.1 ALARM

ALARM			
Column type	**Column Name**	**Data Type**	**Description**
PK	ALARM_KEY	INT(8)	Unique ID assigned to alarm record.
FK3,J2	ALARM_TYPE	SMALLINT	Every alarm is associated with a type:· 1 = Network (reported by NE) 2 = Pattern (generated internally by VPNS after probable root cause analysis) 3 = trend (generated normally by Trending Service)
FK1,J3	ALARM_SOURCE	SMALLINT	Identifies where the alarm is reported from
	NE_CLLI	VARCHAR(60)	11-20 character identifier of the network name of the NE
	ATAG	VARCHAR(60)	Unique alarm ID associated with NE.
	AID	VARCHAR(60)	Access ID - denotes actual physical entity reporting problem (port, card, terminal)
	AID_TYPE	VARCHAR(20)	The different type of physical entities - port, card, terminal
	GENERATION_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm was generated by the NE
	SEVERITY	SMALLINT	1 = Critical 2 = Major 3 = Minorl 4 = Informational

ALARM			
Column type	**Column Name**	**Data Type**	**Description**
	SERVICE_AFFECT	BOOLEAN	True or false - Is alarm affecting service?
	NE_ID	VARCHAR(20)	Unique identifier assigned to the NE
	SUB_NE_ADDRESS	VARCHAR(40)	Reserved for future use
	RECEIVE_TIME	DATETIME YEAR TO FRACTION(3)	Time alarm received by VPNS.
	CONDITION_TYPE	VARCHAR(60)	Type of alarm, for example, Power loss
	DESCRIPTION	VARCHAR(255)	Detailed textual description of the alarm package
FK2,J4	STATE	SMALLINT	As alarm goes through different VPNS services, the state of the alarm changes. 1 = unproccessed (Raw) 2 = processed and presented to user (Process) 3 = Cleared after user has seen it (Clear state) 4 = Alarm filtered by VPNS without being presented to user (Filter state) 5 = Alarm cleared while being put into Thresholding service. User did not see. (Clear in Threshold state)
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package was last modified.
FK4,J1	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	LOCATION	VARCHAR(20)	Location of the alarm
	DIRECTION	VARCHAR(20)	Direction alarm is coming from.
	Hier_name	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created
	RAW_ALARM_ID	VARCHAR(32)	Unique ID assigned to the raw alarm
	HOST_NAME	VARCHAR(32)	Name of the host where the alarm was received from the network (for fault tolerance use)

	ATTRIBUTES	VARCHAR(255)	Extended attributes of the alarm which are not listed above
	RULE_NAME	VARCHAR(60)	Name of the rule used to generate this package.
	ENTITY_ID	INTEGER	Unique ID assigned to the VPNS entity on which the alarm was posted

8.2.2 ALARM_SOURCE

ALARM_SOURCE			
Column type	**Column Name**	**Data Type**	**Description**
PK	ALARM_SOURCE	VARCHAR(32)	Identifies where the alarm is reported from
	DESC	VARCHAR(255)	Detailed textual description of the alarm package.

8.2.3 ALARM_TYPE

ALARM_TYPE			
Column type	**Column Name**	**Data Type**	**Description**
PK	ALARM_TYPE	INTEGER	Every alarm is associated with a type:· 1 = Network (reported by NE) 2 = Pattern (generated internally by VPNS after probable root cause analysis) 3 = trend (generated normally by Trending Service)
	DESC	VARCHAR(255)	Detailed textual description iof alarm.

8.2.4 ALARM_STATE

ALARM_STATE			
Column type	**Column Name**	**Data Type**	**Description**
PK	STATE	SMALLINT	As alarm goes through different VPNS services, the state of the alarm changes. 1 = unproccessed (Raw) 2 = processed and presented to user (Process) 3 = Cleared after user has seen it (Clear state) 4 = Alarm filtered by VPNS without being presented to user (Filter state) 5 = Alarm cleared while being put into Thresholding service. User did not see. (Clear in Threshold state)
	DESC	VARCHAR(255)	Detailed textual description iof alarm.

8.2.5 CLR_ALARM

CLR_ALARM			
Column type	**Column Name**	**Data Type**	**Description**
	ALARM_KEY	INTEGER	Unique ID assigned to alarm record.
	ALARM_TYPE	VARCHAR(32)	Every alarm is associated with a type:· 1 = Network (reported by NE) 2 = Pattern (generated internally by VPNS after probable root cause analysis)
	ALARM_SOURCE	VARCHAR(32)	Identifies where the alarm is reported from
	NE_CLLI	VARCHAR(60)	11-20 character identifier of the network name of the NE

	ATAG	INTEGER	Unique alarm ID associated with NE.
	AID	INTEGER	Access ID - denotes actual physical entity reporting problem (port, card, terminal)
	AID_TYPE	VARCHAR(32)	The different type of physical entities - port, card, terminal
	GENERATION_TIME	TIMESTAMP	Date and time that the alarm was generated by the NE
	SEVERITY	INTEGER	1 = Critical 2 = Major 3 = Minorl 4 = Informational
	SERVICE_AFFECT	BOOLEAN	True or false - Is alarm affecting service?

CLR_ALARM			
Column type	**Column Name**	**Data Type**	**Description**
	NE_ID	VARCHAR(32)	Unique Id assigned to the NE
	SUB_NE_ADDRESS	VARCHAR(32)	Reserved for future use
	RECEIVE_TIME	DATETIME YEAR TO FRACTION(3)	Time alarm received by VPNS.
	CONDITION_TYPE	VARCHAR(32)	Type of alarm, for example, Power loss
	DESCRIPTION	VARCHAR(255)	Detailed textual description of the alarm package
	STATE	SMALLINT	As alarm goes through different VPNS services, the state of the alarm changes. 1 = unproccessed (Raw) 2 = processed and presented to user (Process) 3 = Cleared after user has seen it (Clear state) 4 = Alarm filtered by VPNS without being presented to user (Filter state) 5 = Alarm cleared while being put into Thresholding service. User did not see. (Clear in Threshold state)
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package was last modified.
	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package

	LOCATION	VARCHAR(32)	Location of the alarm
	DIRECTION	VARCHAR(32)	Direction alarm is coming from.

CLR_ALARM			
Column type	**Column Name**	**Data Type**	**Description**
	HIER_NAME	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created
	RAW_ALARM_ID	VARCHAR(32)	Unique ID assigned to the raw alarm
	HOST_NAME	VARCHAR(32)	Name of the host where the alarm was received from the network (for fault tolerance use)
	ATTRIBUTES	VARCHAR(255)	Extended attributes of the alarm which are not listed above
	RULE_NAME	VARCHAR(60)	Name of the rule used to generate this package.
	ENTITY_ID	INTEGER	Unique ID assigned to the VPNS entity on which the alarm was posted
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time this record was inserted into this table.

8.2.6 PACKAGE

PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
PK	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	PKG_TYPE	VARCHAR(20)	Brief explanation of the type of alarm package
	PKG_SUBTYPE	VARCHAR(40)	Reserved for future use
	SEVERITY	SMALLINT	1 = Critical 2 = Major 3 = Minorl 4 = Informational

FK2,J1	STATUS	SMALLINT	Number that indicates status such as ACTIVE, RESOLVED, or TO BE RESOLVED
	CATEGORY	VARCHAR(20)	Label of alarm package type such as REGULAR, ROGUE, or LOV
	NE_TYPE	VARCHAR(20)	Name of NE type that generated the alarm.
	NE_CLLI	VARCHAR(32)	11-20 character identifier of the network name of the NE
	NE_STATE	SMALLINIT	NE state such as PENDING or ACTIVE

PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
FK1,J2	ENTITY_STATUS	SMALLINT	Number that indicates entity status such as ACTIVE or PENDING
	RULE_NAME	VARCHAR(60)	Name of the rule used to generate this package.
	SERVICE_AFFECT	BOOLEAN	True or false - Is alarm affecting service?
	GENERATION_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm was generated by the NE
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package was last modified.
	DESCRIPTION	VARCHAR(255)	Detailed textual description of the alarm package
	TT_ID	VARCHAR(10)	Unique identifier assigned to trouble ticket for this alarm package
	SHELF_NUM	VARCHAR(20)	Information on the shelf location of the alarm source
	SLOT_NUM	VARCHAR(20)	Information on the slot location of the alarm source
	PORT_NUM	VARCHAR(20)	Information on the port location of the alarm source
	ONT_NUM	VARCHAR(10)	Reserved for future use

	ONT_TYPE	VARCHAR(15)	Reserved for future use
	ONT_SERIAL_NUM	VARCHAR(20)	Reserved for future use
	WIRE_CENTER	VARCHAR(20)	Reserved for future use
	CONN_NAME	VARCHAR(20)	Reserved for future use
	ALARM_COND_TYPE	VARCHAR(60)	Textual description of the alarm condition such as GWC304. GWC indicated the alarm source and 304 is the error code. See Nortel Error Code documentation for error code definitions.

PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
	NUM_VOICE_SERVICES	INTEGER	Reserved for future use
	NUM_DATA_SERVICES	INTEGER	Reserved for future use
	NUM_VIDEO_SERVICES	INTEGER	Reserved for future use
	NUM_OF_ONTS	INTEGER	Reserved for future use
	AUTO_TT	BOOLEAN	True or false, specified in the package rule.
	COND_END_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package condition was resolved
	HIER_NAME	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created
	COMP_TID	VARCHAR(60)	Information on the CLLI location where the alarm originated
	ATTRIBUTES	VARCHAR(255)	Extended attributes of the alarm which are not listed above
	ENTITY_ID	INTEGER	Unique ID assigned to the VPNS entity on which the alarm was posted

8.2.7 PACKAGE STATUS

PACKAGE_STATUS			
Column type	**Column Name**	**Data Type**	**Description**
PK	STATUS	SMALLINT	Number that indicates status such as ACTIVE, RESOLVED, or TO BE RESOLVED
	DESC	VARCHAR(255)	Detailed textual description of alarm.

8.2.8 ENTITY_STATUS

ENTITY_STATUS			
Column type	**Column Name**	**Data Type**	**Description**
PK	STATUS	SMALLINT	Indicates status of entity such as ACTIVE or PENDING
	DESC	VARCHAR(255)	Expanded textual description of entity status.

8.2.9 CLR_PACKAGE

CLR_PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	PKG_TYPE	VARCHAR(20)	Brief explanation of the type of alarm package
	PKG_SUBTYPE	VARCHAR(40)	Reserved for future use
	SEVERITY	VARCHAR(32)	1 = Critical 2 = Major 3 = Minorl 4 = Informational

	STATUS	SMALLINT	Number that indicates status such as ACTIVE, RESOLVED, or TO BE RESOLVED
	CATEGORY	VARCHAR(20)	Label of alarm package type such as REGULAR, ROGUE, or LOV
	NE_TYPE	VARCHAR(20)	Name of NE type that generated the alarm.
	NE_CLLI	VARCHAR(32)	11-20 character identifier of the network name of the NE
	NE_STATE	SMALLINT	NE state such as PENDING or ACTIVE
	ENTITY_STATUS	SMALLINT	Number that indicates entity status such as ACTIVE or PENDING
	RULE_NAME	VARCHAR(60)	Name of the rule assigned to this package.
	SERVICE_AFFECT	BOOLEAN	True or false - Is alarm affecting service?
	GENERATION_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm was generated by the NE
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package was last modified.

CLR_PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
	DESCRIPTION	VARCHAR(32)	Detailed textual description of the alarm package
	TT_ID	VARCHAR(10)	Unique identifier assigned to trouble ticket for this alarm package
	SHELF_NUM	VARCHAR(20)	Information on the shelf location of the alarm source
	SLOT_NUM	VARCHAR(20)	Information on the slot location of the alarm source
	PORT_NUM	VARCHAR(20)	Information on the port location of the alarm source
	ONT_NUM	INTEGER	Reserved for future use

	ONT_TYPE	INTEGER	Reserved for future use
	ONT_SERIAL_NUM	INTEGER	Reserved for future use
	WIRE_CENTER	VARCHAR(20)	Reserved for future use
	CONN_NAME	VARCHAR(20)	Reserved for future use
	ALARM_COND_TYPE	VARCHAR(60)	Textual description of the alarm condition such as GWC304. GWC indicated the alarm source and 304 is the error code. See Nortel Error Code documentation for error code definitions.
	NUM_VOICE_SERVICES	INTEGER	Reserved for future use
	NUM_DATA_SERVICES	INTEGER	Reserved for future use
	NUM_VIDEO_SERVICES	INTEGER	Reserved for future use
	NUM_OF_ONTS	INTEGER	Reserved for future use
	AUTO_TT	BOOLEAN	True or false, specified in the package rule
	COND_END_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the alarm package condition was resolved

CLR_PACKAGE			
Column type	**Column Name**	**Data Type**	**Description**
	HIER_NAME	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created
	COMP_TID	VARCHAR(60)	Information on the CLLI location where the alarm originated
	ATTRIBUTES	VARCHAR(255)	Extended attributes of the alarm which are not listed above
	ENTITY_ID	INTEGER	Unique ID assigned to the VPNS entity on which the alarm was posted
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time this record was inserted into this table.

8.2.10 PACKAGE_OP

PACKAGE_OP			
Column type	**Column Name**	**Data Type**	**Description**
PK	PACKAGE_OP_KEY	SERIAL	Unique ID assigned to the package operation.
FK1, I1	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	OPERATION	VARCHAR(32)	If the operation is COMMENT, this field holds the text of the comment.
	OPERATION_TYPE	SMALLINT	Number assigned to the package operation such as ACK, COMMENT, RESOLVEPACKAGE _REQUEST, RESOLVE.
	OPERATED_BY	VARCHAR(20)	Name of the user who performed the last operation.
	OPERATION_TIME	DATETIME YEAR TO FRACTION(3)	Data and time of the last operation performed.
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time this record was inserted into this table.

8.2.11 CLR_PACKAGE_OP

CLR_PACKAGE_OP			
Column type	**Column Name**	**Data Type**	**Description**
	PACKAGE_OP_KEY	INTEGER	Unique ID assigned to the operation of the alarm package
	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	OPERATION	VARCHAR(32)	Name of the operation
	OPERATION_TYPE	VARCHAR(32)	Name of the type of operation
	OPERATED_BY	VARCHAR(32)	Name of the person who initiated the operation

	OPERATION_TIME	DATETIME YEAR TO FRACTION(3)	Date and time the operation was initiated
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time this record was inserted into this table.

8.2.12 TT_INFO

TT_INFO			
Column type	**Column Name**	**Data Type**	**Description**
PK	TT_ID	VARCHAR(10)	Unique identifier assigned to trouble ticket for this alarm package
	CLLI	VARCHAR(32)	11-20 character identifier of the network name of the N
FK1,J1	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	RESPONSIBLECENTERNAME	VARCHAR(32)	Name of the center responsible for this alarm package
	CREATE_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that this trouble ticket was created
	STATUS	SMALLINT	Number that indicates status such as ACTIVE, RESOLVED, or TO BE RESOLVED
	CLOSE_CODE	VARCHAR(4)	vRepair code that was used to close the trouble ticket
	CLOSE_REMARKS	VARCHAR(255)	Explanation of the resolution of the trouble ticket
	CREATED_BY	VARCHAR(32)	Name of the person that created the trouble ticket
	CREATED_BY_SYSTEM	VARCHAR(32)	Name of the system that created the trouble ticket
	LAST_MODIFIED _BY	VARCHAR(32)	Name of the person who last modified the trouble ticket
	LAST_MODIFIED_BY_SYSTEM	VARCHAR(32)	Name of the system that was last used to modify the trouble ticket
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the trouble ticket was modified.

	HIER_NAME	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created

8.2.13 CLR_TT_INFO

CLR_TT_INFO			
Column type	**Column Name**	**Data Type**	**Description**
	TT_ID	VARCHAR(10)	Unique identifier assigned to trouble ticket for this alarm package
	CLLI	VARCHAR(32)	11-20 character identifier for the network name of the nE
	PACKAGE_KEY	INT(8)	Unique ID assigned to the alarm package
	RESPONSIBLECENTERNAME	VARCHAR(32)	Name of the center responsible for the alarm package
	CREATE_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that this trouble ticket was created
	STATUS	SAMLLINT	Number that indicates status such as ACTIVE, RESOLVED, or TO BE RESOLVED
	CLOSE_CODE	VARCHAR(4)	vRepair code that was used to close the trouble ticket
	CLOSE_REMARKS	VARCHAR(255)	Explanation of the resolution of the trouble ticket
	CREATED_BY	VARCHAR(32)	Name of the person that created the trouble ticket
	CREATED_BY_SYSTEM	VARCHAR(32)	Name of the system that created the trouble ticket
	LAST_MODIFIED _BY	VARCHAR(32)	Name of the person who last modified the trouble ticket
	LAST_MODIFIED_BY_SYSTEM	VARCHAR(32)	Name of the system that was last used to modify the trouble ticket
	LAST_MODIFIED_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that the trouble ticket was modified.
	HIER_NAME	VARCHAR(32)	Unique ID assigned to the component where the alarm package was created

CLR_TT_INFO			
Column type	**Column Name**	**Data Type**	**Description**
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time this record was inserted into this table.

8.2.14 QUEUE

QUEUE			
Column type	**Column Name**	**Data Type**	**Description**
PK	NAME	VARCHAR(100)	Name of the entry in the queue.
PK	TYPE	SMALLINT	Type of queue: Hold or Exception.
	ACTIVE	SMALLINT	Indicates whether the is filter ACTIVE or INACTIVE.
	CREATE_TS	DATETIME YEAR TO FRACTION(3)	Date and time that the item was added to the queue.
	START_TS	DATETIME YEAR TO FRACTION(3)	Date and time that the is item active in the queue.
	END_TS	DATETIME YEAR TO FRACTION(3)	Date and time that the item will be inactive in the queue.
	LAST_MOD_TS	DATETIME YEAR TO FRACTION(3)	Date and time that the item was last modified.
	LAST_EXP_TS	DATETIME YEAR TO FRACTION(3)	Date and time that the item was last inactive.
	CREATED_BY	VARCHAR(30)	Name of the person who added this item to the queue.
	LAST_MOD_BY	VARCHAR(30)	Name of the person who last modified the queue item.
	NOTES	VARCHAR(255)	Notes on the filter.

8.2.15 QUEUE_CRITERIA

QUEUE_CRITERIA			
Column type	**Column Name**	**Data Type**	**Description**
PK,FK1	NAME	VARCHAR(100)	Name of the entry in the queue.
PK,FK1	TYPE	SMALLINT	Type of queue: Hold or Exception
PK	CRITERIA_TYPE	VARCHAR(30)	Description of the type of criteria such as clli. Each entry in the queue must match one or more entries in the criteria table.
	VALUE	VARCHAR(255)	The value of the filter criteria being used.

8.2.16 NE_NOTES

NE_NOTES			
Column type	**Column Name**	**Data Type**	**Description**
PK	ID	VARCHAR(100)	Unique ID assigned to this note.
PK	NETWORK_ELEMENT	VARCHAR(60)	Name of the NE being acted upon.
	CREATE_TS	DATETIME YEAR TO FRACTION(3)	Date and time that this note was created.
	LAST_MOD_TS	DATETIME YEAR TO FRACTION(3)	Date and time that this note was last modified.
	CREATED_BY	VARCHAR(30)	Name of the person who created this note.
	LAST_MOD_BY	VARCHAR(30)	Name of the person who last modified this note.
	NOTE	VARCHAR(255)	Notes on the filter

8.2.17 DB_PURGE_CONTROL

DB_PURGE_CONTROL			
Column type	**Column Name**	**Data Type**	**Description**
PK	TABNAME *	VARCHAR(50)	Name of the table to be purged.

	DAYS_TO_PURGE *	INTEGER	How many days to keep data before it is purged.
	MAX_DELETE_TOTAL	INTEGER	How many total rows will be deleted per purge of this table. This can be configured by users via the script: ChangeSchema.sh.
	COMMIT_FREQUENCY *	INTEGER	How many rows will be deleted before a commit is executed.
	MAX_DELETE_BLOCK *	INTEGER	How many rows will be deleted in each purge select.
	STOP_JOB	CHAR(1)	Flag that can be enabled to stop the job. This can be configured by users via stopJanitors.sh.

*Not configurable by users.

8.2.18 DB_PURGE_LOG

DB_PURGE_LOG			
Column type	**Column Name**	**Data Type**	**Description**
PK	ID	SERIAL	Table record ID.
	TABNAME	VARCHAR(50)	Name of the table to be purged.
	STATUS	VARCHAR(20)	Status of the purge: Start/Complete.
	DESC	VARCHAR(100)	Detailed description of the record to be purged.
	MAX_ID	INTEGER	Used internally.
	TOTAL_PURGE	INTEGER	Total rows that were purged.
	ERROR_CODE	INTEGER	Exit code.
	INSERT_TIME	DATETIME YEAR TO FRACTION(3)	Date and time that this record was added to this table.

8.3 Raw Message Database

The Mediation Service receives or retrieves raw messages, parses them and determines which are alarms that need to be forwarded to the Correlation Services for processing. All raw messages are stored in the Raw Message Database.
Each NE has its own message table. Every Sunday midnight, a new table will be created.
The following is an example of NE TID “ABC” for the week starting 01/07/2007.

vnms.abc_20070107			
PK	msgid	int8 not null	Unique ID assigned to raw message
	condtype	char(60)	Type of alarm, for example, LOS, Loss of Signal
	severity	smallint not null	1 = Critical 2 = Major 3 = Minor 4 = Informational
	aid	char(60)	Access Identifier associated with alarm
	msgtext	char(892)	Text of the raw message

8.4 Database Optimization Schedule

With each new installation or upgrade to VPNS, verify that the following Cron Jobs were created and enabled to improve the DB performance:

```
30 2 * * 1-5 . $HOME/.profile;$HOME/vnms_version/bin/vnms -exec updateDbStats.sh -d alarm_db -low
30 3 * * 1-5 . $HOME/.profile;$HOME/vnms_version/bin/vnms -exec updateDbStats.sh -d topology_db -low
30 2 * * 6,0 . $HOME/.profile;$HOME/vnms_version/bin/vnms -exec updateDbStats.sh -d alarm_db -medium
30 3 * * 6,0 . $HOME/.profile;$HOME/vnms_version/bin/vnms -exec updateDbStats.sh -d topology_db -medium
```

9 Appendix A - Inventory Data description

Inventory data is coming from Baais/iView in the form of xml.
Location of the schema is:
${VNMS_HOME}/etc/xsd/vpns/cktdsn.xsd
${VNMS_HOME}/etc/xsd/vpns/ECOSchemaProv.xsd

Following are some of the important tags in the Topology xml:
<OrderNumber>
This can be used to refer to any OC192/Gige ckt xml while communicating with Baais/iView teams.

<DueDate>
Date when the order is going to be complete and the circuits would be activated by eCO

<CircuitId>
Name of the circuit. If the first token (delimited by '/') starts with BV* then it is OC192 circuit or if it starts with VnA or VnB then its VPNS circuit (where n is 1, 2 or 3). There are six Broadcast Video signals so we have 6 VPNS circuits whose names starts with V1A, V1B, V2A, V2B, V3A or V3B. Alphabets 'A' and 'B' in the first token of circuit name distinguishes between working and protect signals e.g.'V1A' is the working signal while 'V1B' is protect signal for signal # Vid1.

<CircuitLayout>/<Header>/<CircuitStatus>
Tells the status of the circuit. When the order is placed, circuit is in Pending state and when the circuit is complete (activated) the it is in InEffect state.
P = Pending state
IE = InEffect state (In service)
PK = Pending Cancellation

<CircuitLayout>/<Header>/<LogicalScidCode>
Unique number given to a particular set of OC192 and VPNS circuits. This will be same for a OC192 circuit and all the VPNS circuits built on top of that OC192 ckt.
We are using this tag to create link between OC192 ckt xml and VPNS ckt xmls.

<SourceSideInfo>
This tag contains cross connect information between Cisco-6509 (or similar devices) and VHO.

<DestSideInfo>
This tag contains cross connect information between VSO and BMR (or BigBand or similar devices).

<Direction>
Gives list of nodes and their corresponding cross connections which will occur while traversing each direction, on the virtual OC192 ring. All the nodes appearing under this tag are in the same sequence in which they appear in the actual ring. In any circuit xml, there can be only two such tags, one for East direction and another for West direction.

<DirectionName>
East or West

<Scid>

Represents the SCID or DWID ring. Each direction can have one or more Scids depending upon the number of Sonet rings used to form the virtual OC192 circuit.

<Scid>/<Channels>
In OC192 circuit xml, this tag tells the lambda number on which the OC192 circuit is riding. It can be any number between 1 and 44 (as currently 44 lambdas can be handled).
In VPNS ckt xml, this tag tells the range of STS channel number used by that VPNS circuit. E.g 1-21
(in order to form one Gigabit signal, 21 STS1s channels are required).

<Scid>/<LeadingSts>
This first STS channel number used for the VPNS circuit.

<Scid>/<NumOfSts>
Total number of STS1s channels used for this VPNS circuit.

<Node>
Contains details of particular NetworkElement, also the cross-connection details made in that NE.

```
<NodeInfo>
      <CliCode>BLTMMDLB016</CliCode>
      <Tid>BLTMMDLB016</Tid>
      <GatewayNeIp>106.40.129.136</GatewayNeIp>
      <Vendor>TELLABS</Vendor>
      <Model>7100</Model>
      <NodeSeq>3</NodeSeq>
      <NodeType>VSO</NodeType>
</NodeInfo>
```

This gives detail information about the Node.

<CliCode>
Identifier for node. First four characters tell the state and city. It can be either 8 or 11 characters.

<Tid>
Target Identifier. Unique number that can be used to identify any entity in the network. It is 18 characters long and ideally the first 8 characters should be CliCode.

<GatewayNeIp>
IP address of the Gateway NE, that can be used to connect to this device to monitor it. As of now, for Pending OC192 ckt xml, we are not getting this tag, but for InEffect ckt xml, we do get it.

<NodeType>
The function performed by that node in this particular cross-connection. Its value can be VHO, VSO, OLA, CBS etc.

<Workstep>/<ActionType>
Action to be taken about the cross connection.
A = Add, cross-connect is going to be created
W = Working, cross-connect has been created and is in service
R = Repeat, cross-connect information has already occurred in some previous xml and its being repeated here, for the sake of completion.
D = Delete, cross-connect has been deleted. The physical entities like port, card, shelf, NE continue to exist, only the cross connection between them has been removed.

<Trail> and <CrossConnect>
“Trail” is mostly used when there is cross connection between two ports located on two separate devices like Cisco-6509 and VHO or VSO and BMR devices.
“CrossConnect” is used when the cross connections are made on two ports existing on the same network entity.

<TpA>
Starting point (port) of the CrossConnect or Trail

<TpZ>
End point (port) of the CrossConnect or Trail

<Shelf>
Details of the Shelf

<Shelf>/<LogicalName>
Shelf number. For Tellabs-7100 device, its probable values are:
MS-1: for shelf#1
PS-2: for shelf#2

<Slot>
Slot or Card details

<Slot>/<Number>
Slot number which this card is present.

<Port>
Details of the port

<Port>/<Number>
Port number used in this cross connection.

<Port>/<RateCode>
Rate code for that particular port with respect to the type of cross connection.
Probably values are:
VPNS: for 1-10 ports of SMTM card in TL-7100 device
OC192: for 11th port of SMTM card in TL-7100 device
OTS/OFX: for all the ports present on LIAM or LOAM or RCMM card in TL-7100 device

10 Appendix B – Message Processing Engine (MPE)

10.1 Overview

The Message Processing Engine (MPE) is a proprietary multi-purpose text parsing tool that is used to process streaming raw messages. MPE extracts the desired information from the raw messages, and parses and reformats the messages into the VPNS alarm structure. MPE patterns also assigns a unique identifier (atag) to the message. MPE consists of a set of commands, function calls and string operations for manipulating text; a Debugger for validating patterns; and a template MPE pattern for each network element (NE) or element management system (EMS) or network management system (NMS) supported by VPNS.

MPE patterns process ascii text messages such as SCC2 log files from Nortel IEMS and SDM, and ascii messages from NFM. MPE does not process SNMP traps. When the MPE receives raw input, the detection process writes the character stream it receives to a buffer known as the "detection buffer". The characters remain in the buffer until processing arrives at a filter or delete command.

MPE patterns are customizable by VPNS site administrators to handle new or changing messages or changing message handling requirements. When new messages need to be handled typically this is done by editing an existing pattern, with extensive use of the MPE Debugger.

MPE processing is a function of the Mediation Service. The Mediation Service collects raw messages received from the Network Entities for processing and parsing by MPE, saves the messages to the Raw Message Database, and forwards messages classified as alarms to the Correlation Services for correlation and packaging.

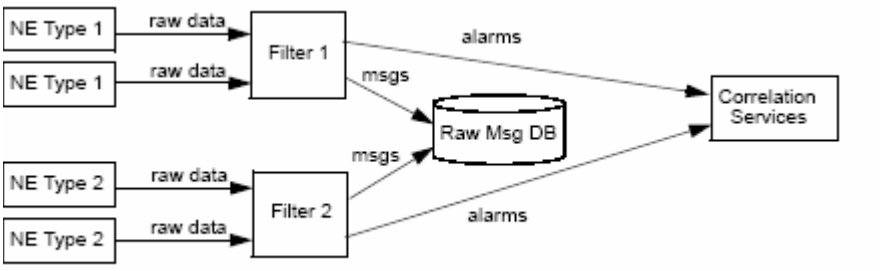


Figure : MPE Pattern File for each Device Type Supported

This chapter explains how to use the MPE Debugger includes a sample MPE pattern file and explains the MPE functions:

- 'AND-OR Structure"
- 'Looping Control Structure"
- 'Cursor-Moving Operations"
- 'Miscellaneous MPE Operations"
- 'Text Capturing Operations"
- 'Variable Manipulation Operations"
- 'Output Operation"

- ‘Function Definition and Calling Operations”
- ‘Variable Scope”
- ‘Built-in String Operations”
- ‘Java Integration”

10.2 AND-OR Structure

The fundamental control structure in MPE is the “AND-OR” structure. This section introduces this important structure.

Message processing behavior is controlled by a group of specifications. Given a block of input text, the specification defines how to edit the text and how to extract useful information from the text. The specification itself is composed of a group of basic operations. The main control structure in MPE is the “AND-OR” structure. This structure is equivalent to the “if … then … else if … then … else …” structure in the conventional programming language, and supports branching.

The following pseudo MPE specification illustrates the “AND-OR” structure:

```
op1 AND op2
OR
op3 AND op4 AND op5
OR
op6
```

In this specification, MPE evaluates op1, if it is true, then MPE evaluates op2, if it is also true, the whole “AND-OR” structure is evaluated to be true and the evaluation of op3, op4, op5, and op6 is skipped. Assuming op1 evaluates to be false, then MPE skips op2 and evaluates the first operation in the next OR branch, which is op3 in the example. If op3 evaluates to be true, MPE evaluates op4 and so on.

In the evaluation process, the MPE keyword “OR” is similar to the “else” clause in the conventional programming languages:

```
If op1 & op2 then
Else if op3 & op4 & op5 then
Else if op6 then
```

An MPE operation returns a Boolean value (true or false), which depends on the success or failure of the operation. Some operations always succeed (for example, som, no_op). To clarify, when we say the operation succeeds, we mean that the message matched the criteria; failure means the message did not match the criteria, not that the pattern failed to work.

The keyword “AND” is optional, and it has higher precedence then “OR”. Parenthesis can be used to change the precedence. For example:

```
op1 op2 OR op3
```

This is the same as:

```
op1 AND op2 OR op3
```

MPE evaluates op1, if it is true, MPE evaluates op2, if op2 is also true, MPE evaluates the whole structure to be true without evaluating op3. If op2 is false, which means op1 AND op2 is false, then MPE evaluates op3 and the evaluation result of op3 is also the evaluation result of the whole structure.

```
op1 (op2 OR op3)
```

// parenthesis is used to change the precedence of “OR”

MPE evaluates op1, if it is true, then the whole structure is true. Otherwise, MPE

evaluates op2, if it is true, then the whole structure is true. If op2 is evaluated to be false, MPE evaluates op3, the evaluation result of op3 is also the evaluation result of the whole structure.
A composite operation is a group of operations surrounded by parenthesis. Both primitive and composite operations are referred to simply as operations.

10.3 Looping Control Structure

Normally, MPE only evaluates an operation once, or it may not evaluate an operation at all. The keyword “while” is used to change this behavior. The operation following the “while” keyword is evaluated again and again as long as its result is true. This behavior is similar to the while loop in a conventional programming language. Two special primitive operations can be used to control the looping: “break” and “next”. Operation “break” terminates the loop immediately, and the whole looping control structure is evaluated to true. This behavior is similar to the “break” statement in a conventional programming language. Operation “next” causes the MPE to skip the rest of operations following it inside the loop, and the control returns to the beginning of the loop. This behavior is similar to the “continue” statement in a conventional programming language.

When MPE reaches “break” or “next” operation inside a looping control structure, the whole structure is evaluated to be true. Otherwise, MPE evaluates the looping control structure to be true only if MPE is able to finish at least one complete cycle of the loop.
The “break” and “next” operations must not be used outside a looping control structure. MPE does not enforce this rule, but its behavior is undefined when the rule is violated.

10.4 Cursor-Moving Operations

MPE has a internal buffer storing the text being edited. Like an interactive editor, it also has a cursor pointing to the current editing position. An interactive editor recognizes special keystrokes, such as “Right Arrow”, “PgUp”, to move its cursor around. Similarly, MPE defines cursor-moving operations to move its cursor within the editing buffer.
This section explains each cursor-moving operation, and gives a line reference to where the operation is used in the example MPE file in this chapter.
The following cursor-moving operations are defined in MPE:

- ‘back(n)”
- ‘eol”
- ‘eom”
- ‘eow”
- ‘find(string)”
- ‘findchar(string)”

- ‘find_back(string)”
- ‘fwd(n)”
- ‘line(n)”
- ‘line_back(n)”
- ‘match(string)”
- ‘matchchars(string)”
- ‘skipchars(string)”
- ‘skipchars_back(string)”
- ‘sol”
- ‘som”
- ‘word(n)”
- ‘word_back(n)”

A cursor motion commands that is impossible to execute returns FALSE and results in no cursor movement.

back(n)
This moves the cursor back n characters, and returns TRUE unless the character is less than n characters after the beginning of the message.
Example: Line 12

eol
This moves the cursor to the end of the current line and always returns TRUE.
Example: Line 30 #4

eom
This command is valid only in Filter Patterns, and always moves the cursor one character past the last character of the message and returns TRUE.
Example: Line 440

eow
This command moves the cursor forward to the first white space character it finds (or to the end of the message), and always returns TRUE. If the cursor is on a white space character, the cursor will not move.
Example: Line 367

find(string)
This command returns TRUE if string is found anywhere ahead of the cursor, and moves the cursor to the first character of the matched string.
Example: Line 297

findchar(string)
This command returns TRUE if ANY character in the string is found ahead of the cursor and moves the cursor to the first character it finds from the string.
Example: Line 341

find_back(string)
This command returns TRUE if string is found anywhere preceding the cursor, and moves the cursor to the first character of the matched string.
Example: Line 2857

fwd(n)
This moves the cursor forward n characters, and returns TRUE unless the cursor is less than n characters before the end of the message.
Example: Line 12

line(n)
This moves the cursor ahead n lines and positions it on the first character of that line. It returns TRUE if the line exists.
Example: Line 438

line_back(n)
This moves the cursor back n lines and positions it on the first character of that line. It returns TRUE if the line exists.
Example: No example

match(string)
This command returns TRUE if string matches the characters in the message text starting at the cursor position, and moves the cursor one character past the last character in the matched string.
Example: Line 12

matchchars(string)
This command returns TRUE if the character at the cursor position matches any character contained in the matched string. Matching will continue one character at a time after the cursor position, and the cursor will move one character beyond the last character that is matched.
Example: Line 439

skipchars(string)
This command always returns TRUE and it moves the cursor ahead to the first character not matching any contained in the string, or to the end of the message.
Example: Line 439

skipchars_back(string)
This command always returns TRUE and it moves the cursor back to the closest preceding character not matching any contained in the string, or to the end of the message.
Example: Line 4531

sol
This moves the cursor to the start of the current line and always returns TRUE.

Example: Line 10

som
This moves the cursor to the first character of the message and returns TRUE, if no checkpoint has been set. If a checkpoint has been set, it returns FALSE and the cursor does not move.
Example: Line 149

word(n)
This moves the cursor to the first character of the nth field ahead of the current position and returns TRUE if the operation was possible. This command does NOT stop at an end of line character (\n).
Example: Line 340

word_back(n)
This moves the cursor to the first character of the n-1 field preceding the current position and returns TRUE if the operation was possible. This command does NOT stop at an end of line character (\n).
Example: line 3453

10.5 Miscellaneous MPE Operations

MPE provides two commands to mark the start point for deleting text. These two commands are used in combination with cursor moving commands to delete unwanted text.
This section explains both commands and gives a line reference to where the operation is used in the example MPE file in this chapter.

mark
This command always returns TRUE. It marks the beginning of text to be deleted.
Example: Line 12

delete
This command always returns TRUE. It deletes text starting from the cursor position where the mark was executed, up to but not including the present cursor position.
Example: Line 12

stop
This command forces the exist from the pattern. The message in the buffer will be filtered out. No alarm is generated.
Example: Line 12

10.6 Work Area Operations

When MPE starts, it creates a default buffer to hold the text being edited. It is possible to save this buffer into a stack, use an alternative buffer to edit, and restore the original buffer from the stack later.
Please note: The text buffer operations are VPNS MPE- specific, TOM does not have these facilities.
Besides the buffer, MPE also defines a few facilities associated with that buffer. The collection is called a work area.
A work area is composed of:

- Text buffer
- Cursor position;
- Marker position (see operation “mark” and “delete”) and
- Recording start positions (see operation “start_rec” and “end_rec”)

This section explains the work area related commands and gives a line reference to where the operation is used in the example MPE file in this chapter.
The following work area related operations are defined in MPE:

push

“push” saves the current work area on top of the MPE internal work area stack. The work area saved by “push” could be restored by operation “pop” later. This operation always evaluates to true.
Example: Line 14

pop

“pop” restores the previously saved work area from the MPE internal work area stack. If the stack is not empty, the work area on top of the stack is restored and the operation returns true. Otherwise, the operation returns false.
Example: Line 22

use(variable)

“use” instructs MPE to copy the value of “variable” to the text buffer in the work area, reset cursor to point to the first character of the text buffer, unset the marker, and unset all recording start positions. This operation returns true if the variable has a value. Otherwise, it returns false.
Example: Line 17

10.7 Text Capturing Operations

It should not be surprising that a programmable editor like MPE supports text capturing. The captured texts are saved in variables that can be referred, printed, or outputted to the external host software system.
The following text capture operations are defined in MPE:

copy(variable)

“copy” copied all characters from the beginning of the text buffer (inclusive) to the position pointed by the cursor (exclusive) to “variable”. Cursor and marker are not moved. This operation always returns true.

Example: No example

cut(variable)

"cut" appends all characters from the beginning of the text buffer (inclusive) to the position pointed by the cursor (exclusive) to "variable". All those characters are then deleted from the buffer. Cursor is moved to the beginning of the text buffer. Marker is not moved. This operation always returns true.
Example: Line 30

end_rec(variable)

This command ends recording text one position before the current cursor position for the identified variable (for example, varname). The text recorded between the positions of the start_rec and end_rec commands are then stored in the identified variable. This command must be preceded by a start_rec(variable) command.
Example: Line 149

start_rec(variable)

This command starts recording text at the current cursor position in the message buffer, and stores the text in the identified variable (for example, varname). This command must be followed by an end_rec() command.
Example: Line 149

10.8 Variable Manipulation Operations

MPE supports variables that store text strings. Many operations are defined in MPE to perform functions like variable comparison, concatenation, dropping a variable, etc. This section explains each variable and references the line in the example MPE pattern in this chapter.
The following variable manipulation operations are defined in MPE:

clear_vars

"clear_vars" deletes all local variables and all visible global variables. It always returns true.
Example: Line 308

eq(string [, string])

This command returns TRUE if string1 is the same as string2. The cursor position is unchanged. string2 is optional.
If string2 is not specified, string1 is compared against the message text, starting at the current cursor position until the length of string1.
So, for example, if you are matching against the string "test" then you would take the first four characters starting from the current cursor position.
Example: Line 318

eq_ic(string [, string])

This command is the same as eq, with the "_ic" parameter (ignore case) adding further control of the variable manipulation.
Example: Line 1074

delete_var(variable)
"delete_var" deletes the variable specified by "variable". It always returns true.
Example: Line 21

ge(string [, string])
This command returns TRUE if string1 has an ASCII value greater than or equal to the value of string2. The cursor position is unchanged. string2 is optional.
If string2 is not specified, string1 is compared against the message text, starting at the current cursor position until the length of string1. It will return TRUE if the message text is greater than or equal to the ASCII value of string1.
Example: No example

ge_ic(string [, string])
This command is the same as ge, with the "_ic" parameter (ignore case) adding further control of the variable manipulation.
Example: No example

gt(string [, string])
This command is similar to ge, but returns TRUE if string1 greater than, but not equal to string2.
If string2 is not specified, string1 is compared against the message text, starting at the current cursor position until the length of string1. It will return TRUE if the message text is greater than, but not equal to string1.
Example: No example

gt_ic(string [, string])
This command is the same as gt, with the "_ic" parameter (ignore case) adding further control of the variable manipulation.
Example: No example

le(string [, string])
This command returns TRUE if the ASCII value of string1 is less than or equal to the value of string2. The cursor position is unchanged. string2 is optional.
If string2 is not specified, string1 is compared against the message text, starting at the current cursor position until the length of string1. It will return TRUE if the message text is less than or equal to string1.
Example: No example

le_ic(string [, string])
This command is the same as le, with the "_ic" parameter (ignore case) adding further control of the variable manipulation.
Example: No example

ne(string [, string])
This command is similar to eq, but returns TRUE if string2 is not equal to string1. If string2 is not specified, string1 is compared against the message text, starting at the current cursor position until the length of string1. It will return TRUE if the message text is not equal to string1.
Example: No example

ne_ic(string [, string])
This command is the same as ne, with the "_ic" parameter (ignore case) adding further control of the variable manipulation.
Example: No example

save(variable, string)
"save" saves the "string" into the "variable". String could be a string literal or concatenation.
Example: Line 1

10.9 Output Operation

In order to integrate with external host software system, MPE outputs a group of name value pairs in a java.util.Map. The names are MPE variable names, saved as key in the java.util.Map, and the values are the values of the MPE variables.
The output operation expects a list of MPE variables whose values are exported.
This section explains the output operations and provides a line reference to the example MPE pattern in this chapter.
The following output operation is defined in MPE:

output(variable [, variable] [, variable] …)
See output example below:
```
output(rawMsg, isParsed, emsIp, neTid, alarmSource, severity,
atag, aidType, aid, generationTime, condType,
componentId, notificationId, desc)
```
Example: Line 306

10.10 Function Definition and Calling Operations

MPE's specification contains a single main body followed by the optional function definitions. The entry point of a specification is the first operation of the main body. The main body is evaluated again and again even if no looping control structure is defined. This is the default behavior of the main body - it exists inside an implicit loop. It is easy to identify the main body: 1) it is defined before any function definitions; 2) it is a whole block; and 3) it is not defined by the "define" keyword. The rest of a specification is function definitions. They are optional, but a non-trivial specification may contain may function definitions. An MPE function is a composite operation (surrounded by a pair of parenthesis) with an optional input parameter list

and output return list.
To define an MPE function, the syntax is

```
define function (variable, variable, …) returns variable, variable, …
(
……
)
```

The variables inside the parenthesis are parameters and the variables after the "returns" are returned to the caller. Please note that empty list is not allowed in MPE. A parameter list or a return list must contain at least one variable. Otherwise, the parenthesis surrounding the parameter list or the "returns" keywords must not appear.
The following syntax is used to define a function without no parameters:

```
define function returns variable, variable…
(
……
)
```

To call an MPE function inside the main body or an MPE function, the syntax is

```
variable, variable, … = call(function(variable, variable, …))
```

Here, the variables to the left of the assignment sign ("=") receive the returned values from the function. Normally, the length of this list should match the length of the "returns" list in the function definition, but this list could be shorter - the extra returned values are ignored. The length of parameter list must match that of the function definition. If no parameter is required, MPE does not allow empty list so the calling syntax should be

```
variable, variable, … = call(function)
```

The whole function body is a composite operation. The evaluation result of the "call" operation equals to the evaluation result of this composite operation.

10.11 Variable Scope

MPE supports both global and local variables inside function body. All variables used inside function body are local unless they are declared as global. All variables used inside main body are global by default.
Global variables are shared among the main body and functions. Inside a function body, MPE does not know a variable is global until it is declared so, and MPE does not trace back to modify a variable's scope when the variable is declared as global.
MPE uses the following syntax to declare a variable to be a global or shared one:

```
global(variable)
```

10.12 Built-in String Operations

MPE operations return true or false. MPE also has a few built-in string operations that return strings. They do string concatenation, sub-string extraction, etc.
This section explains the built-in string operations and provides a line reference to the example MPE pattern in this chapter.
The following built-in string operations are defined in MPE:

+

"+" concatenates string literals and variables. The returned value is a single string. Values of all members are not changed. If a variable is not assigned before it is used in a concatenation, it is treated as a empty string.

$datetime(string)
"$datetime" returns current host system time in specified format "string".
Example: Line 851

$datetime2long(String,String)
$datetime2long" returns the long (time in millis) value of the 1st string. The 2nd string represents the format of the first string.
Example:
save(timeinmillis, $datetime("05-01-29 18:32:10", "yy-MM-dd kk:mm:ss"))

$datetimelong
"$datetimelong" returns the current system time as a long. This time is in milliseconds.
Example:
save(receivedTime, $datetimelong)

$digest(String,String ...)
"$digest" returns a message digest using the supplied strings. The current algorithm used is MD5. The supplied strings are applied to the digest in the order for which they are used in the function.
Example:
save(reptAlarmDigest, $digest(clli, component_id, aid))

$param(string)
"$param" retrieves the parameter value that was given to MPE from the external host software system. The name of the parameter is specified by "string".
Example: Line 304

$replace(string, string, string)
"$replace" substitutes all occurrences of the 2nd "string" in the 1st string by the 3rd string. The 2nd string is a regular expression. The substitution result string is returned.
Example: 1348

$serial
"$serial" returns a unique serial number as a string.
Example: No example

$substr(string, n, [m])
m: This parameter is taken based on length starting from first location of string. Based on n-m difference, substring is extracted.
Example: 1943

$trim(string)
"$trim" removes all white spaces on both sides of the input "string". The white spaces in the middle are not removed. The stripped result string is returned.
Example: Line 855

$tolower(string)

"$tolower" converts all uppercase characters in "string" to their lower case counterparts. The conversion result string is returned.
Example: No example

$toupper(string)

"$toupper" converts all lowercase characters in "string" to their upper case counterparts. The conversion result string is returned.
Example: No example

10.13 Java Integration

The class PIEFactory defines a static member function "compile" that expects an array of MPE specification file names.
public class PIEFactory extends Object { …… public static PIEIntf compile(String[] PIE FileNames) throws Exception { …… }}
As mentioned in the previous section, MPE specification files must have a main body and this main body must appear at the beginning of one specification file; other specification files only contain function definitions. The "compile" function returns a handle to the main body. If main body is not defined in the specification file, the function returns null.

The handle returned by "compile" is a Java interface called PIEIntf.
The following functions are defined inside this interface:
• $public void register(StringBuffer inBuffer, Map inParams, List outList)
• $public void unregister()
• $public void filter(StringBuffer inBuffer, Map inParams, List outList)
• $public void filter(Map inMap, Map inParams, List outList)

Note: Map and List are java.util.Map and java.util.List, respectively
Function "register" attaches an input buffer "inBuffer" containing the text that needs to be edited to MPE. MPE edits the text according the specifications and saves the output in "outList".

10.13.1 Using PIEIntf

PIEIntf itself is an active object. This means that it has an internal thread reading from "inBuffer" and writing to "outList". So "inBuffer" and "outList" are shared between the thread in the host system and PIEIntf internal thread. Obviously, thread synchronization is required to keep data integrity.

One way to accomplish this is:
1. The host system saves text into "inBuffer" and creates a "outList".

2. The host system calls function "register" with "inBuffer" and "outList". This call is not a blocking call, it returns immediately.

3. The host system appends more text to “inBuffer” in a synchronized block and also inform the PIEIntf internal thread that more text is appended. For example:
```
synchronized (inBuffer) { inBuffer.append(text); inBuffer.notify();}
```

4. The host system reads “outList” to get the output generated by MPE “output” operation in another synchronized block. The “outList” could be empty when the host system reads it. Alternatively, the host system may either wait on “outList” until the PIEIntf internal thread wake it up, or poll the “outList” repetitively.

5. The host system calls “unregister” to detach itself with MPE. This step tells MPE there is no more input text and MPE shuts down its internal thread.

6. When “register” is called to link the host system with MPE, MPE implicitly puts a infinite loop around the main body of the specification. This loop runs until “unregister” is called.

7. In filter call, the main body of specification is evaluated only once.

8. The host software system can transfer some parameters to MPE via “inParam”, which is a map contains name value pairs. These parameters are accessible inside MPE specification via the “$param” function.

10.14 Alarm Severities

Five levels of alarm severities are available. The values of the left side of the table are the exact values that must be used in the patterns. MPE patterns can be used to remap alarm severity as needed.

Table: Alarm Severities

Name	Value
CRITICAL	1
MAJOR	2
MINOR	3
INFORMATIONAL	4
IGNORED (message not send to alarms database)	5

10.15 Using the MPE Debugger

The MPE debugger is used to debug pattern files used by the Mediation Service.It performs the following functions:
• Verifies pattern syntax

- Steps through the operations of any given pattern, viewing the current string with the cursor position indicated.

- Sets and clears breakpoints to stop the pattern execution before any pattern operation by specifying the line number, and the number of operations on the line.

10.15.1 Syntax

To launch the debugger, login as vnms and type:
mpedb.sh <pattern file> <input_text_file_name> [<chunk_size>]
The debugger can only load a single pattern file. If there is more than one pattern file, they have to be concatenated to a single file to be debugged.
Another requirement regarding the pattern file is its name (for example, *mevs.d)*. The file name must end with either ".d" or ".f". The debugger runs the main body of the ".d" file in an implicit infinite loop and runs the main body of the ".f" file only once.
The "input_text_file_name" is the file that contains the input text.
The optional command line parameter "chunk_size" instructs the debugger to loading certain number of characters into MPE incrementally. The default chunk size is 8.

10.15.2 Commands

The MPE debugger is an interactive system. It shows a prompt and waits for user's command.
The following debugger commands are supported:

Table: VPNS MPE Debugger Commands

Command	Usage
help	show the Debugger help
r	Starts or resumes running a compiled pattern MPE until encounters a breakpoint
s	Steps the debugger forward one operation
n	Runs to the next operation - the debugger runs the "call" without stepping through the function
l	Lists the pattern file with line numbers for reference
b<line>.<pos>	Sets up a breakpoint Sets up a breakpoint in the current specification file at line "line" operation number "pos". The line and operation are required. For example, to setup a breakpoint at the 2nd operation at line 14, the command is "b 14.2".
d	Drops a breakpoint
vars	Displays the local and global variables

param <name>	Shows all parameters given to MPE from external host software systems
exit	Exits from the debugger

Note: If you run the MPE debugger and get an error message that No Class was found, make sure the following CLASSPATH has been defined in your user profile:

```
export VNMS_HOME=$HOME/vnms_version
export PATH=$VNMS_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:$VNMS_HOME/lib/vpns_1.1.jar
```

10.16 Example MPE Debugger Session

This section shows how to use the MPE Debugger to identify an error in a pattern. The first step in patterning a new message is always to identify the end of message. The raw message in this example uses ^Y as delimiter. However, the pattern is looking for "/032", which is ^Z. As a result, the pattern cannot handle the raw message correctly. This section shows first the raw message, then the MPE Debugger session on the pattern. Note that when you look at a message in the Debugger, the ascii character is automatically translated.

10.16.1 Raw Message from Gateway Controller

^M
^M*C15 GWC 300 2180 TBL GWC Fault
^M Location: GWC-1-UNIT-1
^M NotificationID: 3
^M State: Raise
^M Category: Quality of Service
^M Cause: Underlying resource unavailable
^M Time: Jun 05 14:15:21 2003
^M Component Id: GWC=GWC-1-UNIT-1;Version=PGC09BI;Unit=unit_1;Software=NODE
^M MTC
^M Specific Problem: Unit Out of Service: Service is not available
^M Description: Active unit disabled.
^M
^Y

10.16.2 Using MPE Debugger to find the end of the message and filter it

The delimiter of the message is ^Y. We need to find the end of the message and put it to filter.

1. As user vnms, to start mpedb, execute the following command. “gwc300.txt” is the name of the pattern file” and “200” specifies the chunk size. Chunk size is optional.

$ mpedb.sh mevs1.d /home/mc06/mevs_alarms/gwc300.txt 200

2. This will start the debugger and you will see the prompt:
pie ==>

3. To view help, at the prompt type:
pie ==>help
help show the help
r run
s step
n next
l list
b setup a breakpoint
d drop a breakpoint
vars show all variables
param show/set parameters
exit exit from pie

4. To list the pattern, type:

```
pie ==> l
[1] save(debug, "T") #turn on the debug mode
[2] save(set_trace_on, "F")
[3] save(use_tom_desc, "T")
[4] call(main)
[5]
[6] define main
[7] (
[8] global(rawMsg)
[9]
[10] sol
[11] (
[12] match("\r") back(1) mark fwd(1) delete
[13] or
[14] match("\032") back(1) mark fwd(1) delete push # save work area
[15]
[16] (
[17] use(rawMsg)
[18]
[19] call(filter_dispatch(rawMsg)) #call function to process the message
[20]
[21] delete_var(rawMsg)
[22] pop # restore work area
[23] or
[24] delete_var(rawMsg)
[25] pop
[26] )
[27] or
[28] match("\n") back(1) mark fwd(1) delete
[29] or
[30] eol match("\n") cut(rawMsg)
[31] )
[32] )
```

5. To set a breakpoint at line 14, first operation, type:
pie ==> b 14.1
breakpoint was setup successfully on mevs1.d:14.1

6. To run the debugger, type:
pie ==> r
\n\r*C15 GWC 300 2180 TBL GWC Fault\n\r Location: GWC-1-UNIT-1\n\r Notif...
^
next mevs1.d:14.1: match("^Z")
pie ==> r
.....
pie ==> r
14.1 *** failed ***
\n^Y\n
^
next mevs1.d:14.1: match("^Z")
pie ==> r
14.1 *** failed ***
^Y\n
^
next mevs1.d:14.1: match("^Z")
pie ==> r
14.1 *** failed ***
no more input
$
The End of Line was incorrectly set; it needs to be changed to \031.

10.17 Example VPNS MPE Pattern File

This example of a VPNS MPE pattern file demonstrates the MPE functions listed earlier in this chapter. Lines have been numbered to
facilitate referring to the appropriate section in the file.
The AND-OR structure, Cursor-Moving Operations (match, sol, som), Work Area Operations (pop, use), Output Operation, Function Definition (define), Variable Scope (global) are used in this example.
The database schema includes: rawMsg, isParsed, emsIp, neTid, alarmSource, severity, atag, aidType, aid, generationTime, condType, componentId, notificationId, desc. Looking at the software code, the start of the filtering follows the line (appearing in bold) **define dmssns_f11(rawMsg).**
The procedure of filtering runs the rawMsg into the first subroutine (dmssns_lnp_300_304) and if it does not match, proceeds to the next subroutine until it finds a match.
VPNS MPE maintains a “current work area” (see ‘Miscellaneous MPE Operations”). The rawMsg is placed in the “current work area”, and all the patterns are matched against the rawMsg.
An example of information on logs in SCC2 header format appears near the end of this code sample. Search for the words **scc2 log report.**

To add/modify/debug a pattern, create the subroutine and add a CALL command in the upper part of the pattern.

Table: Alarm Table from the Alarm Database Schema

Parameter	Description
alarm_key	Unique ID
alarm_type	Every alarm associated with type:· 1 = Network (reported by Network Element) 2 = Pattern (generated internally by VPNS after probable root cause)
alarm_source	Identifies where the alarm is reported from
ne_clli	11-20 character identifier of the network name of the Network Element
atag	Unique alarm ID associated with Network Element
aid	Access ID - denotes actual physical entity reporting problem (port, card, terminal)
aid_type	The different type of physical entities - port, card, terminal
generation_time	Time alarm generated by NE
severity	Informational, Level 1; Minor, Level 2; Major, Level 3; Critical, Level 4
service_affect	Is alarm affecting service?
sub_ne_address	This parameter is used to add complementary information to aid
receive_time	Time alarm received by VPNS
condition_type	Type of alarm, for example, Power loss. Condition is in the detailed typevendor documentation of the installed NEs.
description (desc)	Detailed description in alarm
state	As alarm goes through different VPNS services, the state of the alarm changes. 1 = unproccessed (Raw) 2 = processed and presented to user (Process) 3 = Cleared after user has seen it (Clear state) 4 = Alarm filtered by VPNS without being presented to user (Filter state) 5 = Alarm cleared while being put into Thresholding service. User did not see. (Clear in Threshold state)
last_modified_time	Time alarm last modified in VPNS
package_key	Unique ID of package which alarm is associated

Parameter	Description
location	Used in Correlation Services. Network location: • NEND - near end •FEND -far-end
direction	Used in Correlation Services. Direction the alarm is coming from: •trmt •rcve

11 Appendix C - Configuration files

The following files are copies of the configuration files summarized in this chapter.

11.1 AlarmSkulker.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<properties>
```

```
      <propertyfile filename="${VNMS_HOME}/etc/config/config.xml"
            contenttype="xml" optional="false"/>


      <!--  Batch size  -->
      <property name="BatchSize">
            <value>20000</value>
      </property>

      <!-- When an operation is performaed on a locked row, -->
      <!-- the connection                                   -->
      <!-- waits for the given time before it fails. Time   -->
      <!-- in secs.                                         -->
      <property name="LOCK_MODE_TO_WAIT_TIME">
            <value>30</value>
      </property>

      <property name="SkulkeringPolicy">
            <value>generation_time</value>   <!-- or last_modified_time -->
      </property>

      <property name="RunAlarmSkulker">
            <value>true</value>
      </property>

      <property name="RetainDaysNullPackage">
            <value>3</value>
      </property>

      <property name="RetainDaysNonNullPackage">
            <value>30</value>
      </property>


      <!-- Logging related parameters. -->
      <property name="log.file.name ">
            <value>${VNMS_HOME}/logs/AlarmSkulker.log</value>
      </property>
      <!-- Load local overridefile. -->
      <propertyfile filename="${VNMS_HOME}/etc/config/AlarmSkulker.par.local"
            contenttype="nv" optional="true"/>
</properties>
```

11.2 DbConnPoolKey.prop

```
#-------------------------------------------------------------------
# Parameter: SQL query to use for testing a connection
#-------------------------------------------------------------------
VALIDATION_QUERY=select dummy from dummy

#-------------------------------------------------------------------
#   **** FOLLOWING PROPERTIES ARE FOR OBJECT POOLING          *****
# ------------------------------------------------------------------
#-------------------------------------------------------------------
# Parameter: maxActive controls the maximum number of objects that
#            can be borrowed from the pool at one time.
#            When non-positive, there is no limit to the number of
#            objects that may be active at one time. When maxActive
#            is exceeded, the pool is said to be exhausted
#-------------------------------------------------------------------
MAX_ACTIVE=20
```

```
#-----------------------------------------------------------------
# Parameter: maxIdle controls the maximum number of objects that
#            can sit idle in the pool at any time. When negative,
#            there is no limit to the number of objects that may be
#            idle at one time.
#-----------------------------------------------------------------
MAX_IDLE=10

#-----------------------------------------------------------------
# Parameter: minIdle indicates minimum number of "sleeping" instances
#            in the pool before the evictor thread (if active) spawns
#            new objects.
#-----------------------------------------------------------------
MIN_IDLE = 5

#-----------------------------------------------------------------
# Parameter: When whenExhaustedAction is WHEN_EXHAUSTED_BLOCK
#      If a positive maxWait value is supplied, the borrowObject()
#      will block for at most that many milliseconds, after which
#      a NoSuchElementException will be thrown.
#      If maxWait is non-positive, the borrowObject() method will
#      block indefinitely.
#-----------------------------------------------------------------
MAX_WAIT=60000

#-----------------------------------------------------------------
# Parameter: whenExhaustedAction specifies the behaviour of the
#            borrowObject() method when the pool is exhausted:
#    When whenExhaustedAction is WHEN_EXHAUSTED_FAIL, borrowObject()
#        will throw a NoSuchElementException
#    When whenExhaustedAction is WHEN_EXHAUSTED_GROW, borrowObject()
#        will create a new object and return it (essentially making
#        maxActive meaningless.)
#    When whenExhaustedAction is WHEN_EXHAUSTED_BLOCK, borrowObject()
#        will block (invoke Object.wait(long) until a new or idle
#        object is available.
#-----------------------------------------------------------------
WHEN_EXHAUSTED_ACTION=WHEN_EXHAUSTED_BLOCK

#-----------------------------------------------------------------
# Parameter: When testOnBorrow is set, the pool will attempt to
#            validate each object before it is returned from the
#            borrowObject() method. (Using the provided factory's
#            PoolableObjectFactory.validateObject(java.lang.Object)
#            method.) Objects that fail to validate will be dropped
#            from the pool, and a different object will be borrowed.
#-----------------------------------------------------------------
TEST_ON_BORROW=true

#-----------------------------------------------------------------
# Parameter: When testOnReturn is set, the pool will attempt to
#            validate each object before it is returned to the pool
#            in the returnObject(java.lang.Object) method. (Using
#            the provided factory's
#            PoolableObjectFactory.validateObject(java.lang.Object)
#            method.) Objects that fail to validate will be dropped
#            from the pool.
#-----------------------------------------------------------------
TEST_ON_RETURN=false

#-----------------------------------------------------------------
# Parameter: testWhileIdle indicates whether or not idle objects
#            should be validated using the factory's
```

```
#            PoolableObjectFactory.validateObject(java.lang.Object)
#            method. Objects that fail to validate will be dropped
#            from the pool.
#-------------------------------------------------------------------
TEST_WHILE_IDLE=true

#-------------------------------------------------------------------
# Parameter: timeBetweenEvictionRunsMillis indicates how long the
#            eviction thread should sleep before "runs" of examining
#            idle objects. When non-positive, no eviction thread
#            will be launched
#-------------------------------------------------------------------
EVICTION_TIME_CYCLE=900000

#-------------------------------------------------------------------
# Parameter: minEvictableIdleTimeMillis specifies the minimum amount
#            of time that an object may sit idle in the pool before
#            it is eligable for eviction due to idle time. When
#            non-positive, no object will be dropped from the pool
#            due to idle time alone.
#-------------------------------------------------------------------
MIN_EVICTION_IDLE_TIME=1800000

#-------------------------------------------------------------------
# Parameter: number of objects to examine per run in the idle object
#            evictor
#-------------------------------------------------------------------
NUM_TESTS_PER_EVICTION_RUN=3
```

11.3 MediationSvce.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<properties>
      <propertyfile filename="${VNMS_HOME}/etc/config/config.xml"
            contenttype="xml" optional="false"/>

            <!--  TL1 connection time constants  -->
            <property name="TL1AuditIntervalMsec">
            <value>300000</value>
        </property>

        <property name="TL1NESleepMinutes">
            <value>3</value>
        </property>


        <property name="TL1NECmdExpireSeconds">
            <value>90</value>
        </property>


            <!--  TCP connection Interval in mini-seconds  -->
            <property name="TcpSleepInterval">
            <value>300</value>
        </property>

        <property name="TcpConnectionInterval">
            <value>90000</value>
        </property>
```

```
        <!--  sanityCheckInterval in seconds  -->
    <property name="SanityCheckInterval">
        <value>120</value>
    </property>

        <!--  Port Number -->
        <property name="PortNum">
              <value>${medsvc.orbport}</value>
        </property>

        <!--  Console Thread Port  -->

        <property name="ConsolePort">
              <value>8008</value>
        </property>

        <!-- MPE config File Name -->

        <property name="PieConfigFileName">
              <value>${VNMS_HOME}/etc/xml/pie/pieConfig.xml</value>
        </property>

        <!-- Connection Matrix File Name -->

        <property name="MediationMatrixFileName">
              <value>${VNMS_HOME}/etc/xml/medsvc/mediationMatrix.xml</value>
        </property>

        <property name="MaxBookKeepFileSize">
              <value>100000000</value>
        </property>


        <property name="MaxTcpLogFileSize">
              <value>10000</value>
        </property>


        <property name="MaxTcpLogFileNumber">
              <value>4</value>
        </property>
        <!-- Input file for loading NE from file-->
    <property name="neFile">
            <value>${VNMS_HOME}/etc/config/ne.txt</value>
    </property>
        <!-- Option to load NE form File, DB or Both-->
        <!-- 1=DB, 2=File, 3=Both-->
    <property name="load_option">
            <value>1</value>
    </property>


        <!-- Topology External Interface Corba URL -->
        <property name="LocalTopologyAccessURL">

  <value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/TopoDbExtService</value>

        </property>
        <property name="RemoteTopologyAccessURL">

  <value>corbaloc:iiop:1.2@${REMOTEH}:${ponserver.orbport}/TopoDbExtService</value>
```

```
            </property>

            <!-- ======================================================== -->
            <!--  properties for Message Service Helper API -->
            <!-- ======================================================== -->
    <property name="WebMessageDestination">
        <value>vnms.webmessages</value>
    </property>
    <property name="vnms.webmessages.dataDirection">
                        <value>SEND_AND_RECEIVE</value>
                </property>

    <property name="vnms.topologyupdates.dataDirection">
                <value>RECEIVE</value>
            </property>



            <property name="AlarmClearCount">
                  <value>5</value>
            </property>

            <property name="AlarmCountThreshold">
                  <value>1843200</value>
            </property>

            <!-- interval in second between checking the alarm flow to detect flooding -
->
            <property name="AlarmDetectInterval">
                  <value>60</value>
            </property>


            <!-- ======================================================== -->
            <!--  Database related parameters                        -->
            <!-- ======================================================== -->

            <!-- RawMsgDb URL  -->
            <property name="RawMsgDb">
                  <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${rawmsg_dbname}:INFORMIXSERVER=${DB_LOCALH}</value>
            </property>

            <!-- SystemDb URL -->
            <property name="SystemDb">
                  <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${sys_dbname}:INFORMIXSERVER=${DB_LOCALH}</value>
            </property>

            <!-- Raw Message Db Space(data) -->
            <property name="TableSpaces">
                  <value>rawmsg_dbs1 rawmsg_dbs2 rawmsg_dbs3 rawmsg_dbs4</value>
            </property>

            <!-- Raw Message Db Space(index) -->
            <property name="IndexSpaces">
                  <value>rawidx_dbs1 rawidx_dbs2 rawidx_dbs3 rawidx_dbs4</value>
            </property>

            <!-- Initial extent size when creating tables -->
            <property name="ExtentSize">
                  <value>300000</value>
            </property>
```

```
            <!-- Next extent size when expanding tables -->
            <property name="NextSize">
                  <value>100000</value>
            </property>

            <!-- Watermark of free db space(in percentage), below which medsvc will
issue warnning  -->
            <property name="DbThresholdHigh">
                  <value>20</value>
            </property>

            <!-- Watermark of free db space(in percentage), below which medsvc will drop
old tables -->
            <property name="DbThresholdLow">
                  <value>10</value>
            </property>

            <!-- Number of Raw Msg Threads -->
            <property name="NumberOfRawMsgThreads">
                  <value>3</value>
            </property>

            <!-- ======================================================== -->
            <!-- JMS related paramters, Check Weblogic JMS server setup -->
            <!-- ======================================================== -->
            <!-- Start JMS or not default is true-->
            <property name="MedJMSMode">
                  <value>true</value>
            </property>

            <!-- JMS Provider URL -->
            <property name="JMSProviderURL">
                  <value>t3://${LOCALH}:${JMSProviderPort}</value>
            </property>

            <!-- JNDI context factory -->
            <property name="JNDI_FACTORY">
                  <value>weblogic.jndi.WLInitialContextFactory</value>
            </property>

            <!-- JMS connection factory -->
            <property name="JMS_FACTORY">
                  <value>com.vz.vnms.jms.TopicConnectionFactory_1</value>
            </property>

            <!-- JNDI Name of RawMessageTopic -->
            <property name="RawMsgTopic">
                  <value>vnms.rawmsg</value>
            </property>



            <!-- ======================================================== -->
            <!-- Logging related PARAMETERS -->
            <!-- ======================================================== -->

            <property name="log.file.name">
                  <value>${VNMS_HOME}/logs/mediationSvce.log</value>
            </property>

            <property name="log.file.maxSize">
                  <value>16384</value>
            </property>
```

```
            <!-- Load local overridefile. -->
            <propertyfile filename="${VNMS_HOME}/etc/config/MediationSvce.par.local"
                        contenttype="nv" optional="true"/>
</properties>
```

11.4 TopologyLoader.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<properties>

        <!-- include config.xml file which has got configurations common across TnhsGige
applications;
                        config.xml in turn includes config.par.local file     -->
        <propertyfile filename="${VNMS_HOME}/etc/config/config.xml" contenttype="xml"
optional="false"/>



        <!-- ======================================================== -->
        <!-- Topology DB related parameters -->
        <!-- ======================================================== -->
        <!-- all the relvant parameters have been moved to the common file "config.xml"
-->



        <!--
=================================================================================================      -->
        <!--  Following properties are for JMS TopicSubscriber (for getting Inventory
from Baais)     -->
        <!--
================================================================================== -
->

                <!-- Name of TopicSubscriber on which Inventory data would be published -
->
                <property name="InventoryUpdateMessageDestination">
                        <value>vnms.baaisUpdates</value>
                </property>
        <property name="vnms.baaisUpdates.topologyTopic">
                        <value>vnms.baaisUpdates</value>
                </property>
        <property name="vnms.baaisUpdates.brokerUrl">
                 <value>t3://${LOCALH}:${JMSProviderPort}</value>
            </property>
        <property name="vnms.baaisUpdates.messagingType">
                 <value>JMS</value>
            </property>
        <property name="vnms.baaisUpdates.dataDirection">
                 <value>RECEIVE</value>
            </property>
        <property name="vnms.baaisUpdates.applicationId">
```

```
            <value>vpns</value>
        </property>
    <property name="vnms.baaisUpdates.connectionFactory">
            <value>weblogic.jms.ConnectionFactory</value>
        </property>
    <property name="vnms.baaisUpdates.initialContextFactory">
            <value>weblogic.jndi.WLInitialContextFactory</value>
        </property>
    <property name="vnms.baaisUpdates.defaultRequestTimeout">
            <value>80000</value>
        </property>
    <property name="vnms.baaisUpdates.receiveCheckInterval">
            <value>2000</value>
        </property>
    <property name="vnms.baaisUpdates.connectRetryInterval">
            <value>80000</value>
        </property>
    <property name="vnms.baaisUpdates.sendQueueSize">
            <value>500</value>
        </property>
    <property name="vnms.baaisUpdates.sendRetryInterval">
            <value>10000</value>
        </property>
    <property name="vnms.baaisUpdates.hearbeatInterval">
            <value>70000</value>
        </property>

    <property name="BaaisUpdtClientId">
           <value>VPNSTopoBaaisUpdtClientId</value>
    </property>

    <property name="BaaisUpdtSubsId">
           <value>VPNSTopoBaaisUpdtSubsId</value>
    </property>

    <!-- ========================================================================   -->
    <!-- Following properties are for Publishing Topology updates to JMS Topic   -->
    <!-- ========================================================================   -->

           <property name="TopologyUpdateMessageDestination">
                   <value>vnms.topologyupdates</value>
           </property>
    <property name="vnms.topologyupdates.brokerUrl">
            <value>t3://${LOCALH}:${JMSProviderPort}</value>
        </property>
    <property name="vnms.topologyupdates.messagingType">
            <value>JMS</value>
        </property>
    <property name="vnms.topologyupdates.dataDirection">
            <value>SEND</value>
        </property>
    <property name="vnms.topologyupdates.applicationId">
            <value>vpns</value>
        </property>
    <property name="vnms.topologyupdates.connectionFactory">
            <value>weblogic.jms.ConnectionFactory</value>
        </property>
```

```
        <property name="vnms.topologyupdates.initialContextFactory">
                <value>weblogic.jndi.WLInitialContextFactory</value>
            </property>
        <property name="vnms.topologyupdates.defaultRequestTimeout">
                <value>80000</value>
            </property>
        <property name="vnms.topologyupdates.receiveCheckInterval">
                <value>2000</value>
            </property>
        <property name="vnms.topologyupdates.connectRetryInterval">
                <value>80000</value>
            </property>
        <property name="vnms.topologyupdates.sendQueueSize">
                <value>500</value>
            </property>
        <property name="vnms.topologyupdates.sendRetryInterval">
                <value>10000</value>
            </property>
        <property name="vnms.topologyupdates.hearbeatInterval">
                <value>70000</value>
            </property>



        <!-- ======================================================== -->
        <!-- Logging related parameters -->
        <!-- ======================================================== -->

                <!-- set the log level (trace|debug|info|warn|error)  -->
                <property name="log.level">
                        <value>debug</value>
                </property>

                <!-- name and location of the log file used by TopoLoader -->
                <property name="log.file.name">
                        <value>${VNMS_HOME}/logs/topologyLoader.log</value>
                </property>

                <!-- maximum permitted size of the log file (in mb)  -->
                <property name="log.file.maxSize">
                        <value>15360</value>
                </property>
                <property name="log.file.maxNum">
                        <value>5</value>
                </property>
                <property name="log.file.append">
                        <value>true</value>
                </property>



        <!--
================================================================================================      -->
        <!--  Location of various directories for initial dump, metadata, INMS dump,
backup, xslt,  -->
```

```
        <!--
================================================================================================      -->

                <!-- ${TOPO_HOME} should be set through environment variable (during
installtion itself) -->

                <!-- location where Baais will put initial dump of topology data; Loader
will pick it from here -->
                <property name="TopoDataLocIn">
                 <value>${TOPO_HOME}/topodata/initialDump</value>
            </property>

                <!-- location of metadata xmls; TopoLoader will pick these xmls from here
for loading metadata -->
                <property name="TopoMetaDataLocation">
                        <value>${VNMS_HOME}/etc/xml/topology_metadata</value>
                </property>

            <!-- location of back up where processed xml files would be stored for
future references -->
            <property name="TopoDataLocBackup">
                 <value>${TOPO_HOME}/topodata/backup</value>
            </property>

                <!-- where the Topology Messages (xml files) will be dumped for INMS -->
                <property name="TopoDataINMSDumpLoc">
                 <value>${TOPO_HOME}/topodata/inms</value>
            </property>

            <property name="InternalTopologyDataLocation">
                 <value>${TOPO_HOME}/internal_topo_data</value>
            </property>

            <!-- absolute path of the stylesheet to be used by TopoLoader -->
                <property name="xslSource">
                        <value>${VNMS_HOME}/etc/xslt/vpns_topo.xsl</value>
                </property>



    <!-- check the file system for any Inventory data files after at this interval (unit
is minute) -->
    <property name="TopoDataCheckingInterval">
        <value>60</value>
    </property>

        <!-- application ID -->
    <property name="serviceid">
        <value>VPNSTopoLoader</value>
    </property>

        <!-- DCI Wrapper to choose for creating appropriate Data Collection
implementation (like JMS) -->
    <property name="dciwrapper.factory.handler">
        <value>com.vz.vpns.baais.VPNSBaaisDCIWrapper</value>
    </property>
```

```
        <!-- whether to load Metadata while loading Topology data;
                        set it true for the first time load or whenever something changes
in metadata -->
        <property name="LoadTopoMetadata">
                <value>true</value>
        </property>



        <!-- Stylesheet transformer factory implementation type; should be changed if
the parser is changed -->
        <property name="javax.xml.transform.TransformerFactory">
                <value>com.icl.saxon.TransformerFactoryImpl</value>
        </property>

    <property name="TOPO_DATAFILE_NAMEHEADER">
        <value>vpns_topo_data.xml</value>
    </property>

        <property name="MessageServiceType">
                <value>JMS</value>
        </property>

        <!-- name of the corba service (used by VNMS TopoLoader codebase) -->
        <property name="TopoCUDInterfaceKey">
                <value>TopoCUDInterface</value>
        </property>

        <!-- Load local overridefile. -->
        <propertyfile filename="${VNMS_HOME}/etc/config/TopologyLoader.par.local"
                contenttype="nv" optional="true"/>

</properties>
```

11.5 VpnsInmsAdapter.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<!--
#  *** TnhsINMSAdapter.xml defines all variables that are used  ***
#  *** by TnhsINMSAdapter                                       ***
#  *** configured by installation script                        ***
#  *** Note : Properties unique to a service and not part       ***
#  ***        part installation time configuration should       ***
#  ***        be placed in corresponding par file               ***

#  **  FOLLOWING PROPERTIES ARE PLACE-HOLDER ONLY.               **
#  **  You must define the actual properties in                  **
#  **  config.par.local. Typically created during                **
#  **  server installation process                               **
-->

<properties>
```

```
    <!-- ========================================================= -->
    <!-- ** event logger processing interval                     ** -->
    <!-- ========================================================= -->

    <property name="event.logger.processing.interval">
         <value>5</value>
    </property>
    <!-- ========================================================= -->
    <!-- ** Message handle class name list                       ** -->
    <!-- ** Each subsystem should have a config.par.local file   ** -->
    <!-- ** to overwrite this config                             ** -->
    <!-- ========================================================= -->
    <property name="message.handle.class.list">
        <value>com.vz.vpns.adapters.inms.fault.VpnsXMLMsgBuilder</value>
    </property>

        <!-- ========================================================= -->
    <!-- ** INMS Adapter DB properties                           ** -->
    <!-- ========================================================= -->
    <property name="local_db_name">
        <value>alarm_db</value>
    </property>
    <property name="alarm_db.dbhandle.class">
        <value>com.vz.vpns.adapters.inms.fault.db.VpnsDbHandle</value>
    </property>
    <property name="alarm_db.dbhandle.jdbc.driver">
        <value>${jdbc.driver}</value>
    </property>
    <property name="alarm_db.dbhandle.jdbc.url">
        <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${local_db_name}:INFORMIXSERVER=${DB_LOCALH}</value>
    </property>
     <property name="alarm_db.dbhandle.dirtyread">
        <value>true</value>
    </property>
       <property name="alarm_db.dbhandle.conn.class">
        <value>com.vz.vnms.db.InformixJdbcConnection</value>
    </property>
    <!-- days of tickets retained in database -->
    <property name="ticket_retention_history">
        <value>180</value>
    </property>
    <!-- days of event logging retained in database -->
    <property name="event_logging_retention_history">
        <value>15</value>
    </property>

    <!-- ========================================================= -->
    <!-- ** INMSEventPublisher related config                    ** -->
    <!-- ========================================================= -->
    <!-- Parameter: Specify all the INMS instances to which I will -->
    <!--            forward events.                                -->
    <!-- Value is intentionally left blank. It should be assigned  -->
    <!-- at the time of installation.                              -->
    <!-- Use comma separated list to specify more than 1 receivers -->
    <!-- e.g. inmsbh,inmsfh                                        -->
```

```
<!-- Out of the box following configuration values are included-->
<!-- inmsbh    : INNMS blue hill production instance           -->
<!-- inmsfh    : INNMS freehold production instance            -->
<!-- inmswal   : INNMS waltham dev instance.                   -->
<!--             ** Availability and Configuration can change  -->
<!--             without notice. Consult IT before using this  -->
<!--             receiver.                                     -->
<!-- inmslocal : *ONLY* In case need to interface with INNMS 2.0-->
<!--             Do not use multiple receivers in INNMS 2.0 mode-->
<!--             2.0 use ONLY one receiver 'inmslocal'         -->
<!-- ========================================================= -->
<property name="EventReceivers">
    <!-- *** PLEASE FIX-ME ***                                 -->
    <value>${Inms_EventReceivers}</value>
</property>
    <!-- JNDI context factory -->
    <property name="JNDI_FACTORY">
           <value>weblogic.jndi.WLInitialContextFactory</value>
    </property>
    <property name="JMS_FACTORY">
           <value>weblogic.jms.ConnectionFactory</value>
    </property>
    <property name="inms.FaultTopic">
    <value>fault.tnhs-gige</value>
</property>

<!-- ========================================================= -->
<!--  JMS connection URLs for different setups.                -->
<!-- ========================================================= -->
<property name="inmsbh.brokerUrl">
    <value>t3s://105.252.8.147:8005</value>
</property>
<property name="inmsfh.brokerUrl">
    <value>t3s://106.22.4.53:8005</value>
</property>
<property name="inmswal.brokerUrl">
    <value>t3s://132.197.96.15:8005</value>
</property>
<property name="inmsfw.brokerUrl">
    <value>t3s://113.134.218.112:8005</value>
</property>
<property name="inmslocal.brokerUrl">
    <value>t3://che.gte.com:9500</value>
</property>


<!-- ========================================================= -->
<!--  Store and Forward parameters                             -->
<!-- ========================================================= -->
<!-- How many events to buffer. This is the "Store" in "S&F"   -->
<!-- functionality. When buffer overflows because of loss of   -->
<!-- connectivity to message destination, entire buffer is     -->
<!-- flushed, buffering disabled and resync done (if enabled)  -->
<!-- at later point well after the connection is regained.     -->
<!-- Note : Higher the number, greater will be the load on     -->
<!--        memory requirements and also flood of events upon  -->
<!--        re-connection. Too low a number can mean many      -->
```

```
    <!--        avoidable resyncs.                                      -->
    <property name="inmsbh.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="inmsfh.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="inmswal.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="inmslocal.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="inmsfw.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="inmsge.eventBufferSize">
        <value>1000</value>
    </property>
    <property name="eventBufferSize">
        <value>1000</value>
    </property>
    <!-- If incoming rate is greater than outgoing even without    -->
    <!-- losing connectivity, messages may need to be dropped.     -->
    <!-- This property controls sending notification upon such     -->
    <!-- case.                                                     -->
    <!-- Bigger buffer size or JMS server tuning or network        -->
    <!-- route analysis may remedy the problem situation           -->
    <property name="enableBufferOverflowMessage">
        <value>true</value>
    </property>
    <property name="bufferOverflowInterval">
        <value>30</value>
    </property>
    <!-- ========================================================= -->
    <!--     Following properties are for  KeepAlive functionality -->
    <!-- ========================================================= -->
    <!-- Whether to send keep alive messages or not                -->
    <property name="keepalive.enable">
        <value>true</value>
    </property>
    <!-- time interval for a keep alive message is to be sent. Unit-->
    <!-- is in mins                                                -->
    <property name="keepalive.interval">
        <value>5</value>
    </property>
    <property name="keepAlive.enhancedformat">
        <value>false</value>
    </property>


       <!-- ========================================================= -->
    <!--     Following properties are for event delivery tuning    -->
    <!-- ========================================================= -->
    <!-- How many events to be sent in one package event group     -->
    <property name="LiveFaultEventGroupSize">
        <value>1</value>
```

```
    </property>
    <!-- How many milliseconds to wait before sending events       -->
    <!-- Higher the value, better utilization of the groupsize     -->
    <!-- This property will be ignored if Group(Pack) size is 1    -->
    <property name="FaultEventGroupingDelay">
        <value>30000</value>
    </property>

</properties>
```

11.6 VpnsInmsResync.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<!--
#  *** TnhsINMSResync.xml defines all variables that are used   ***
#  *** by TnhsINMSResync                                        ***
#  *** configured by installation script                        ***
#  *** Note : Properties unique to a service and not part       ***
#  ***        part installation time configuration should       ***
#  ***        be placed in corresponding par file               ***

#  **  FOLLOWING PROPERTIES ARE PLACE-HOLDER ONLY.               **
#  **  You must define the actual properties in                  **
#  **  config.par.local. Typically created during                **
#  **  server installation process                               **
-->

<properties>
        <propertyfile filename="${VNMS_HOME}/etc/config/vnms.xml" contenttype="xml"
optional="false"/>
    <!-- Load Main Tnhsge INMS Adapter config file here -->

    <property name="log.file.name">
        <value>${VNMS_HOME}/logs/VpnsINMSResync.log</value>
    </property>

    <property name="resync_source_name">
        <value>${PrimaryHost1}</value>
    </property>

    <property name="resync.file.path">
        <value>${VPNS_FAULT_DIR}</value>
    </property>

    <property name="resync.file.nameheader">
        <value>Vpns_INMS_Full_Resync.xml</value>
    </property>

    <property name="resync_cutoff_time">
        <value>4</value>
    </property>

    <property name="status_list">
        <value>4</value>
```

```
    </property>

    <!-- Load local file here -->
       <propertyfile filename="${VNMS_HOME}/inms/VpnsINMSResync.par.local"
contenttype="nv"
        optional="true"/>
</properties>
```

11.7 config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<!--
#  *** config.par defines all variables that are either used    ***
#  *** by multiple services or properties that need to be       ***
#  *** configured by installation script                        ***
#  *** Note : Properties unique to a service and not part       ***
#  ***        part installation time configuration should       ***
#  ***        be placed in corresponding par file               ***

#  **  FOLLOWING PROPERTIES ARE PLACE-HOLDER ONLY.               **
#  **  You must define the actual properties in                  **
#  **  config.par.local. Typically created during                **
#  **  server installation process                               **
-->
<properties>
    <!-- The primary server's primary hostname -->
    <property name="PrimaryHost1">
        <value>Put_Primary_Server_Name_Here</value>
    </property>
    <!-- The primary server's secondary hostname -->
    <property name="PrimaryHost2">
        <value>dummy</value>
    </property>
    <!-- The secondary server's primary hostname -->
    <property name="SecondaryHost1">
        <value>Put_Secondary_Server_Name_Here</value>
    </property>
    <!-- The secondary server's secondary hostname -->
    <property name="SecondaryHost2">
        <value>dummy</value>
    </property>
    <!-- This parameter must be properly set for Primary and Backup servers.
    true for the primary server and false for the backup.
    -->
    <property name="IsPrimaryServer">
        <value>Put_Is_Primary_Flag_Here</value>
    </property>
    <!-- Name of the local server.-->
    <property name="LOCALH">
        <value>localhost</value>
    </property>
    <!-- Name of the remote server. -->
    <property name="REMOTEH">
        <value>Put_Remote_Server_Name_Here</value>
```

```
    </property>
    <!-- Is this fault tolerant installation. -->
    <property name="DualServerMode">
        <value>Put_DualServerMode_Choice_Here</value>
    </property>

    <!-- ============================================================  -->
    <!-- **  FOLLOWING PROPERTIES ARE SHARED PROPERTIES            **  -->
    <!-- ============================================================  -->
    <property name="HierNameDelimiter">
        <value>.</value>
    </property>
    <!-- app_name=# defined in vnmsenv[.local]. do not redefine here    -->

    <!--  Server ports  -->
    <!-- ============================================================  -->

    <!-- NameServicePort = # defined in vnmsenv[.local]. do not         -->
    <!-- redefine here                                                 -->

    <property name="JMSProviderPort">
        <value>9020</value>
    </property>
    <property name="adminserver.orbport">
        <value>52590</value>
    </property>
    <property name="ponserver.orbport">
        <value>52591</value>
    </property>
    <property name="medsvc.orbport">
        <value>52592</value>
    </property>
    <property name="topoloader.orbport">
        <value>52593</value>
    </property>

    <!-- DB params -->
    <property name="dbowner">
        <value>vnms</value>
    </property>
    <property name="sys_dbname">
        <value>sysmaster</value>
    </property>
    <property name="rawmsg_dbname">
        <value>rawmsg_db</value>
    </property>
    <property name="alarms_dbname">
        <value>alarm_db</value>
    </property>
    <property name="topology_dbname">
        <value>topology_db</value>
    </property>
    <property name="jdbc_port">
        <value>1531</value>
    </property>
    <property name="dbuserparfile">
        <value>${VNMS_HOME}/etc/config/dbuser.par</value>
```

```
</property>

<!-- Admin Service params -->
<!-- user id, its more meaningful for GUI when real user logs in -->
<property name="userId">
    <value>${app_name}</value>
</property>
<!-- admin related XSL stylesheets directory -->
<property name="adminxsldir">
    <value>${VNMS_HOME}/etc/xslt</value>
</property>
<!-- admin related DTD directory -->
<property name="admindtddir">
    <value>${VNMS_HOME}/etc/dtd</value>
</property>
<!--
every service will resend status every following number of milliseconds
if normal attempt fails
-->
<property name="admin.status.update.retry.interval">
    <value>30000</value>
</property>
<!--
# when the service is stopped through administrative actions
# or unix kill command, it triggers Shutdown sequence,
# if for some reason the clean shutdown fails to complete,
# within reasonable time, we need to stop it with a hard stop
# following value (in millisecs) must be in tune with max time allowed
# for graceful shutdown in stopall.sh (currently 90 sec)
# This value also has an impact on Administratively done stop.
# longer the value slower (or even timedout) response to the end-user.
-->
<property name="forcedhalt.waitime">
    <value>30000</value>
</property>
<!-- where to log loaded properties (if log.level is set to debug) -->
<property name="application.props.dump">
    <value>${VNMS_HOME}/logs/props</value>
</property>

<!-- ========================================================= -->
<!-- ** FOLLOWING PROPERTIES ARE FOR Common Logging         ** -->
<!-- ========================================================= -->
<!-- Currently only LOG4J is supported. -->
<property name="log.api">
    <value>LOG4J</value>
</property>
<!-- Accepted values are file and stdout -->
<property name="log.type">
    <value>file</value>
</property>
<!-- Values (trace, debug, info, warn, error) -->
<property name="log.level">
    <value>debug</value>
</property>
<!-- This assumes that -Dapp_name is given on the command line. -->
<property name="log.file.name">
```

```
        <value>${VNMS_HOME}/logs/${app_name}.log</value>
    </property>
    <!-- Size is in KB -->
    <property name="log.file.maxSize">
        <value>1024</value>
    </property>
    <!-- Current + Current.N files. -->
    <property name="log.file.maxNum">
        <value>4</value>
    </property>
    <!-- error logging xml files directory -->
    <property name="errorfilesdir">
        <value>${VNMS_HOME}/etc/error</value>
    </property>

    <!--  CORBA interface Nameservice registration keys -->
    <!-- Alarm Query key -->
    <property name="NewAlarmQueryKey">
        <value>NewAlarmQuery</value>
    </property>

    <!-- Ping service key -->
    <property name="PingServiceKey">
        <value>PingStateService</value>
    </property>

    <!-- State service key  -->
    <property name="StateServiceKey">
        <value>StateService</value>
    </property>

    <!-- Package Operations service key -->
    <property name="PackageOpsServiceKey">
        <value>PackageOpsService</value>
    </property>

    <!-- Alarm receiver service key  -->
    <property name="AlarmReceiverKey">
        <value>AlarmReceiver</value>
    </property>

    <!-- Topology loader interface on corelation server -->
    <property name="TopologyLoaderKey">
        <value>TopologyLoader</value>
    </property>

    <!-- NameService url -->
    <property name="NameService">
        <value>corbaloc:iiop:1.2@${LOCALH}:${NameServicePort}/NameService</value>
    </property>

    <!-- ======================================================= -->
    <!--  Various JDBC URLS -->
    <!-- ======================================================= -->
    <property name="DB_LOCALH">
        <value>${LOCALH}</value>
    </property>
```

```
    <property name="DB_PrimaryHost1">
        <value>${PrimaryHost1}</value>
    </property>

    <property name="DB_SecondaryHost1">
        <value>${SecondaryHost1}</value>
    </property>
    <property name="PrimaryAlarmDb">
        <value>jdbc:informix-
sqli://${PrimaryHost1}:${jdbc_port}/${alarms_dbname}:INFORMIXSERVER=${DB_PrimaryHost1}</
value>
    </property>
    <property name="SecondaryAlarmDb">
        <value>jdbc:informix-
sqli://${SecondaryHost1}:${jdbc_port}/${alarms_dbname}:INFORMIXSERVER=${DB_SecondaryHost
1}</value>
    </property>
    <property name="TopologyDb">
        <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${topology_dbname}:INFORMIXSERVER=${DB_LOCALH}</value>
    </property>
    <property name="AlarmDb">
        <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${alarms_dbname}:INFORMIXSERVER=${DB_LOCALH}</value>
    </property>
    <!--  CORBA interface URLs -->
    <property name="AlarmReceiverURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${AlarmReceiverKey}</value>
    </property>
    <property name="LocalAlarmReceiverURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${AlarmReceiverKey}</value>
    </property>
    <property name="RemoteAlarmReceiverURL">

<value>corbaloc:iiop:1.2@${REMOTEH}:${ponserver.orbport}/${AlarmReceiverKey}</value>
    </property>
    <property name="NewAlarmQueryURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${NewAlarmQueryKey}</value>
    </property>
    <property name="PrimaryTopologyLoaderURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${TopologyLoaderKey}</value>
    </property>
    <property name="SecondaryTopologyLoaderURL">

<value>corbaloc:iiop:1.2@${REMOTEH}:${ponserver.orbport}/${TopologyLoaderKey}</value>
    </property>
    <property name="LocalStateServiceURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${StateServiceKey}</value>
    </property>
    <property name="RemoteStateServiceURL">
```

```
<value>corbaloc:iiop:1.2@${REMOTEH}:${ponserver.orbport}/${StateServiceKey}</value>
    </property>
    <property name="LocalPingServiceURL">

<value>corbaloc:iiop:1.2@${LOCALH}:${ponserver.orbport}/${PingServiceKey}</value>
    </property>
    <property name="RemotePingServiceURL">

<value>corbaloc:iiop:1.2@${REMOTEH}:${ponserver.orbport}/${PingServiceKey}</value>
    </property>

    <!-- ======================================================== -->
    <!--  properties for Message Service Helper API -->
    <!-- ======================================================== -->
    <property name="messagingType">
        <value>JMS</value>
    </property>
    <property name="jmsVendor">
        <value>weblogic</value>
    </property>
    <property name="TopologyUpdateMessageDestination">
        <value>vnms.topologyupdates</value>
    </property>
    <property name="AdminMessageDestination">
        <value>vnms.administration</value>
    </property>
    <property name="PackageUpdateMessageDestination">
        <value>vnms.packageupdates</value>
    </property>
    <property name="local.brokerUrl">
        <value>t3://${LOCALH}:${JMSProviderPort}</value>
    </property>
    <property name="remote.brokerUrl">
        <value>t3://${REMOTEH}:${JMSProviderPort}</value>
    </property>
    <property name="local.nameServiceUrl">
        <value>corbaloc:iiop:1.2@${LOCALH}:${NameServicePort}/NameService</value>
    </property>
    <property name="remote.nameServiceUrl">
        <value>corbaloc:iiop:1.2@${REMOTEH}:${NameServicePort}/NameService</value>
    </property>
    <property name="vnms.administration.dataDirection">
        <value>SEND_AND_RECEIVE</value>
    </property>

    <!-- VNMS Topology DB properties-->
    <property name="jdbc.driver">
        <value>com.informix.jdbc.IfxDriver</value>
    </property>
    <!-- Internal Interface -->
    <property name="topodb.dbhandle.class">
        <value>com.vz.vnms.db.topdb.TopoDbHandle</value>
    </property>
    <property name="topodb.dbhandle.jdbc.driver">
        <value>${jdbc.driver}</value>
    </property>
```

```
    <property name="topodb.dbhandle.jdbc.url">
        <value>${TopologyDb}</value>
    </property>
     <property name="topodb.dbhandle.dirtyread">
        <value>true</value>
    </property>
       <property name="topodb.dbhandle.conn.class">
        <value>com.vz.vnms.db.InformixJdbcConnection</value>
    </property>

    <!-- Alarm db properties -->
    <property name="alarmdb.dbhandle.class">
        <value>com.vz.vpns.services.db.AlarmDbHandle</value>
    </property>
    <property name="alarmdb.dbhandle.jdbc.driver">
        <value>${jdbc.driver}</value>
    </property>
    <property name="alarmdb.dbhandle.jdbc.url">
        <value>${AlarmDb}</value>
    </property>
     <property name="alarmdb.dbhandle.dirtyread">
        <value>true</value>
    </property>
       <property name="alarmdb.dbhandle.conn.class">
        <value>com.vz.vnms.db.InformixJdbcConnection</value>
    </property>

    <!-- External Interface -->
    <property name="topoextintf.dbhandle.class">
        <value>com.vz.vnms.db.topdb.TopoDbHandle</value>
    </property>
    <property name="topoextintf.dbhandle.jdbc.driver">
        <value>${jdbc.driver}</value>
    </property>
    <property name="topoextintf.dbhandle.jdbc.url">
        <value>${TopologyDb}</value>
    </property>
    <property name="topodbextintf.dbhandle.dirtyread">
        <value>true</value>
    </property>
       <property name="topodbextintf.dbhandle.conn.class">
        <value>com.vz.vnms.db.InformixJdbcConnection</value>
    </property>

    <!-- ======================================================== -->
    <!-- Alarm Source/Severity Definitions                        -->
    <!-- ======================================================== -->
    <property name="AlarmSourceDefinitions">
        <value>4:CS2000,5:PP8600,6:PP15000,7:SAM21,8:GWC,9:UAS,10:RTP_Portal,11:JUNIPER_
M40,12:MG9000,13:MCS5200,14:USP,15:CICM,16:APPSVR,17:NWKSVR,18:VMAILSVR,19:SBC,20:MS2010
,21:CMT,22:SS8,23:VSMART</value>
    </property>

    <property name="AlarmSeverityDefinitions">
        <value>1:1,1:CRITICAL,2:2,2:MAJOR,3:3,3:MINOR,4:4,4:INFO,5:5,5:CLEAR</value>
    </property>
```

```
    <!-- ======================================================== -->
    <!-- The following properties are for OpenORB. -->
    <!-- ======================================================== -->
    <!-- Logging level. -->
    <property name="openorb.debug.trace">
        <value>0</value>
    </property>
    <property name="openorb.debug.level">
        <value>0</value>
    </property>
    <!-- Thread pool settings. -->
    <property name="openorb.server.maxThreadPoolSize">
        <value>30</value>
    </property>

    <!-- ======================================================== -->
    <!-- The following properties are for the alarm cleanup thread. -->
    <!-- ======================================================== -->

    <!-- Entity alarm cleanup interval (in hours) -->
    <property name="EntityAlarmCleanupInterval">
        <value>12</value>
    </property>

    <!-- Entity alarm alive period (in hours) -->
    <property name="DetectedAlarmExpireTime">
        <value>48</value>
    </property>


    <!-- Load local file here -->
    <propertyfile filename="${VNMS_HOME}/etc/config/config.par.local" contenttype="nv"
        optional="false"/>

    <!-- ======================================================== -->
    <!-- Topology DB related parameters -->
    <!-- ======================================================== -->
    <property name="jdbc.driver">
        <value>com.informix.jdbc.IfxDriver</value>
    </property>

    <property name="vpns_dbname">
        <value>topolog_db</value>
    </property>

    <property name="vpns_db.dbhandle.class">
        <value>com.vz.vpns.db.VPNSDbHandle</value>
    </property>

    <property name="vpns_db.dbhandle.jdbc.driver">
        <value>${jdbc.driver}</value>
    </property>

    <property name="vpns_db.dbhandle.jdbc.url">
        <value>jdbc:informix-
sqli://${LOCALH}:${jdbc_port}/${topology_dbname}:INFORMIXSERVER=${DB_LOCALH}</value>
    </property>
```

```
    <property name="vpns_db.dbhandle.jdbc.user">
        <value>vnms</value>
    </property>

    <property name="vpns_db.dbhandle.jdbc.pass">
        <value>vnms</value>
    </property>

    <property name="vpns_db.dbhandle.dirtyread">
        <value>true</value>
    </property>

    <property name="vpns_db.dbhandle.conn.class">
        <value>com.vz.vnms.db.InformixJdbcConnection</value>
    </property>

    <property name="vpns_db.dbmgr.conn.pool.test.onReturn">
        <value>false</value>
    </property>

    <property name="vpns_db.dbmgr.conn.pool.test.onRequest">
        <value>true</value>
    </property>

    <property name="vpns_db.entity.cache.max">
        <value>1500</value>
    </property>
</properties>
```

11.8 vnms.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtdpath:${VNMS_HOME}/etc/dtd/properties.dtd">
<properties>
      <propertyfile filename="${VNMS_HOME}/etc/config/config.xml"
            contenttype="xml" optional="false"/>
      <!-- Pon server always starts on this port -->
      <property name="PortNum">
            <value>${ponserver.orbport}</value>
      </property>
      <!--ORB Related parameters -->
      <property name="PONContainerPOA">
            <value>PONContainerPOA</value>
      </property>
      <!-- recover alarms and packages on startup -->
      <property name="RecoverOnStartup">
            <value>true</value>
      </property>

            <!-- ================================================== -->
```

```
            <!--  properties for Message Service Helper API -->
            <!-- ================================================== -->
          <property name="vnms.topologyupdates.dataDirection">
                    <value>RECEIVE</value>
            </property>
          <property name="vnms.packageupdates.createChannel">
                    <value>true</value>
            </property>
          <property name="vnms.packageupdates.dataDirection">
                    <value>SEND</value>
            </property>

      <!-- ############################################################ -->
      <!-- State Check and State Recovery related Parameters -->
      <!-- Frequency and timeouts in milli secs. -->
      <!-- ############################################################ -->
      <!--Time to allow db replication to finish before starting recovery -->
      <property name="WaitTimeBeforeStateUpdate">
            <value>5000</value>
      </property>
      <property name="StateRecoveryFrequency">
            <value>300000</value>
      </property>
      <property name="StateCheckRoundTripTimeOut">
            <value>30000</value>
      </property>
      <property name="PlatformSocketTimeOut">
            <value>15000</value>
      </property>
      <property name="StateCheckFrequency">
            <value>5000</value>
      </property>
      <!--The following two params should be modified based on the above two params. -->
      <!--StateCheckThreshold * StateCheckFrequency is the amount of time the server -->
      <!--will take to decide its mate is down. -->
      <property name="StateCheckThreshold">
            <value>5</value>
      </property>
      <!--NumOfFailuresThreshold * (StateCheckRoundTripTimeout + StateCheckFrequency) --
>
      <!--is the amount of time the server will take to decide there are network
problems. -->
      <property name="NumOfFailuresThreshold">
            <value>30</value>
      </property>
      <!-- ################################################## -->
      <!--Alarm Filtering Parameters -->
      <!-- ################################################## -->
      <property name="FilterAlarmsAfterPosting">
            <value>true</value>
      </property>
      <property name="FilterAlarms">
            <value>true</value>
      </property>
      <property name="FilterPackages">
            <value>true</value>
      </property>
```

```
<!-- ======================================================== -->
<!-- Alarm Query API configuration parameters                 -->
<!-- ======================================================== -->
<property name="MaxResponseGroups">
    <value>250</value>
</property>
<property name="AlarmQueryHandler">
    <value>com.vz.vpns.services.alarmquery.AlarmQueryImpl</value>
</property>
<property name="AlarmQueryConnectionPool">
    <value>${VNMS_HOME}/etc/config/AlarmQueryConnPool.prop</value>
</property>
<property name="AlarmQueryConnPoolKey">
    <value>AlarmQueryConnPoolKey</value>
</property>
<property name="MaxQueryGroups">
    <value>6</value>
</property>
<property name="AlarmQueryIorFileName">
    <value>${VNMS_HOME}/logs/alarmQuery.ior</value>
</property>
<!-- Size of the time window for querying alarms.           -->
<!-- Default: If input is missing starttime and/or endtime  -->
<!--          72 Hours = 259200000ms                        -->
<property name="NewAlarmQueryWindow">
    <value>259200000</value>
</property>
<!-- For active alarm we will respect start and endtime     -->
<!-- supplied by the caller, but for cleared alarms we will -->
<!-- reserve the right to limit the time window to a        -->
<!-- configured maximum.                                    -->
<property name="TimewindowOverrideForClearedAlarms">
    <value>false</value>
</property>
<!-- ###################################################### -->
  <!-- Thread related parameters -->
  <!-- ###################################################### -->
  <!-- max number of threads in thread pool. -->
  <property name="MAX_THREAD_NUM">
        <value>10</value>
  </property>
  <property name="NUM_FREE_THREADS_ALLOWED">
        <value>10</value>
  </property>
  <!-- ###################################################### -->
  <!-- Alarm databases (with db pool) -->
  <!-- ###################################################### -->
  <property name="db.conn.pool">
        <value>${VNMS_HOME}/etc/config/DbConnPoolKey.prop</value>
  </property>
  <property name="DbConnPoolKey">
        <value>DbConnPoolKey</value>
  </property>
  <!--Batch size  -->
  <property name="BatchSize">
        <value>20000</value>
  </property>
```

```
<!--When an operation is performaed on a locked row, the connection -->
<!--waits for the given time before it fails. Time in secs. -->
<property name="LOCK_MODE_TO_WAIT_TIME">
      <value>30</value>
</property>
<!-- ######################################################## -->
<!-- Entity related parameters -->
<!-- ######################################################## -->
<!--Active entity status values -->
<property name="ActiveStatusList">
      <value>INSTALLED, AVAILABLE, IN_SERVICE, ACTIVE</value>
</property>
<!--Pending entity status values -->
<property name="PendingStatusList">
      <value>NOT_AVAILABLE</value>
</property>
<!-- Topology Reloaded -->
<property name="TopologyDataReloaded">
      <value>false</value>
</property>

<!-- ######################################################## -->
<!-- Rule files related parameters -->
<!-- ######################################################## -->
<!-- Threshold rules -->
<property name="ThresholdRules">
      <value>${VNMS_HOME}/etc/xml/threshold</value>
</property>
<!-- Pattern rules -->
<property name="PatternRules">
      <value>${VNMS_HOME}/etc/xml/pattern</value>
</property>
<!-- Alarm Filtering rules -->
<property name="FilteringRules">
      <value>${VNMS_HOME}/etc/xml/filter</value>
</property>
<!-- Posting service rules -->
<property name="PostingRules">
      <value>${VNMS_HOME}/etc/xml/posting</value>
</property>
<!-- Matching rules -->
<property name="MatchingRules">
      <value>${VNMS_HOME}/etc/xml/mapping</value>
</property>
<!-- Trending rules -->
<property name="TrendingRule">
      <value>${VNMS_HOME}/etc/xml/trending</value>
</property>
<!-- Packaging service rules -->
<property name="PackagingRules">
      <value>${VNMS_HOME}/etc/xml/packaging</value>
</property>
<!-- Event Conditions List -->
<property name="EventConditionsList">
      <value>${VNMS_HOME}/etc/config/EventConditions.txt</value>
</property>
```

```
<!-- ################################################## -->
<!-- Posting Service Policy     -->
<!-- ################################################## -->
<property name="PostOnExactEntOnly">
      <value>true</value>
</property>
<!-- ################################################## -->
<!-- Server Healthy Console  Port   -->
<!-- ################################################## -->
<property name="ServerHealthyConsolePort">
      <value>8005</value>
</property>

<!-- ################################################## -->
<!-- Trend service time window configuration (min)     -->
<!-- ################################################## -->
<property name="TrendingSvcTimeWindow">
      <value>1</value>
</property>

<!-- ################################################## -->
<!-- Entity Attribute Name definition                   -->
<!-- ################################################## -->
<property name="EntAttrInventoryStatus">
      <value>InventoryStatus</value>
</property>
<property name="EntAttrPort">
      <value>port_number</value>
</property>
<property name="EntAttrCard">
      <value>Card</value>
</property>
<property name="EntAttrSlot">
      <value>slot_number</value>
</property>
<property name="EntAttrShelf">
      <value>shelf_number</value>
</property>
<property name="EntAttrNe">
      <value>clli_code</value>
</property>
<property name="EntAttrNESpecName">
      <value>ne_type</value>
</property>
<property name="EntAttrLongNESpecName">
      <value>LongNESpecName</value>
</property>
<property name="EntAttrState">
      <value>State</value>
</property>
<property name="EntAttrLocationCode">
      <value>LocationCode</value>
</property>
<property name="EntAttrEquipmentCode">
      <value>EquipmentCode</value>
</property>
```

```
      <!-- ################################################ -->
      <!-- Logging related parameters. -->
      <!-- ################################################ -->
      <property name="log.file.name">
            <value>${VNMS_HOME}/logs/vnms.log</value>
      </property>
      <property name="log.file.maxSize">
          <value>10240</value>
      </property>

      <!-- ################################################ -->
      <!-- VNMSBroker related parameters. -->
      <!-- ################################################ -->
      <property name="vnmsbrokerparfile">
            <value>${VNMS_HOME}/etc/config/Broker.xml</value>
      </property>

      <propertyfile filename="${VNMS_HOME}/etc/config/VpnsInmsAdapter.xml"
contenttype="xml" optional="true"/>

      <!-- Load local overridefile. -->
      <propertyfile filename="${VNMS_HOME}/etc/config/vnms.par.local"
            contenttype="nv" optional="true"/>
</properties>
```

12 Appendix D - Definitions, Acronyms, and Abbreviations

Acronym	Definition
DWDM	Dense Wavelength Division Multiplexing
FTTP	Fiber-to-the-Premise
VPNS	Verizon Packet Network Surveillance
GigE	Gigabit Ethernet
INMS	Integrated Network Management System
TCA	Threshold Crossing Alerts
PM	Performance Monitoring
OSS	Operations Support System
SCP	Secure Copy
LOV	Loss Of Visibility
NE	Network Entity
IOF	Inter-Office Facility
Och-DPRING	Optical Channel Dedicated Protection Ring
UPSR	Uni-directional Path Switched Ring