

# Event Management and Best Practices

Implement and use best practices for  
event processing

Customize IBM Tivoli products  
for event processing

Diagnose IBM Tivoli Enterprise  
Console, NetView, Switch Analyzer



Tony Bhe  
Peter Glasmacher  
Jacqueline Meckwood  
Guilherme Pereira  
Michael Wallace





International Technical Support Organization

## **Event Management and Best Practices**

June 2004

**Note:** Before using this information and the product it supports, read the information in “Notices” on page ix.

### **First Edition (June 2004)**

This edition applies to the following products:

- ▶ Version 3, Release 9, of IBM Tivoli Enterprise Console
- ▶ Version 7, Release 1, Modification 4 of IBM Tivoli NetView
- ▶ Version 1, Release 2, Modification 1 of IBM Tivoli Switch Analyzer

**Note:** This IBM Redbook is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this IBM Redbook for more current information.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	ix
Trademarks .....	x
 <b>Preface</b> .....	xi
The team that wrote this redbook .....	xi
Become a published author .....	xiii
Comments welcome .....	xiii
 <b>Chapter 1. Introduction to event management</b> .....	1
1.1 Importance of event correlation and automation .....	2
1.2 Terminology .....	4
1.2.1 Event .....	4
1.2.2 Event management .....	4
1.2.3 Event processing .....	5
1.2.4 Automation and automated actions .....	5
1.3 Concepts and issues .....	6
1.3.1 Event flow .....	6
1.3.2 Filtering and forwarding .....	7
1.3.3 Duplicate detection and throttling .....	7
1.3.4 Correlation .....	8
1.3.5 Event synchronization .....	15
1.3.6 Notification .....	16
1.3.7 Trouble ticketing .....	17
1.3.8 Escalation .....	17
1.3.9 Maintenance mode .....	19
1.3.10 Automation .....	19
1.4 Planning considerations .....	20
1.4.1 IT environment assessment .....	21
1.4.2 Organizational considerations .....	21
1.4.3 Policies .....	23
1.4.4 Standards .....	23
 <b>Chapter 2. Event management categories and best practices</b> .....	25
2.1 Implementation approaches .....	26
2.1.1 Send all possible events .....	26
2.1.2 Start with out-of-the-box notifications and analyze reiteratively .....	27
2.1.3 Report only known problems and add them to the list as they are identified .....	27
2.1.4 Choose top X problems from each support area .....	28

2.1.5	Perform Event Management and Monitoring Design	28
2.2	Policies and standards	32
2.2.1	Reviewing the event management process	33
2.2.2	Defining severities	34
2.2.3	Implementing consistent standards	36
2.2.4	Assigning responsibilities	37
2.2.5	Enforcing policies	38
2.3	Filtering	39
2.3.1	Why filter	39
2.3.2	How to filter	40
2.3.3	Where to filter	41
2.3.4	What to filter	41
2.3.5	Filtering best practices	44
2.4	Duplicate detection and suppression	45
2.4.1	Suppressing duplicate events	45
2.4.2	Implications of duplicate detection and suppression	46
2.4.3	Duplicate detection and throttling best practices	50
2.5	Correlation	51
2.5.1	Correlation best practices	51
2.5.2	Implementation considerations	54
2.6	Notification	56
2.6.1	How to notify	56
2.6.2	Notification best practices	58
2.7	Escalation	60
2.7.1	Escalation best practices	60
2.7.2	Implementation considerations	65
2.8	Event synchronization	66
2.8.1	Event synchronization best practices	67
2.9	Trouble ticketing	68
2.9.1	Trouble ticketing best practices	69
2.10	Maintenance mode	72
2.10.1	Maintenance status notification	73
2.10.2	Handling events from a system in maintenance mode	74
2.10.3	Prolonged maintenance mode	75
2.10.4	Network topology considerations	76
2.11	Automation	77
2.11.1	Automation best practices	78
2.11.2	Automation implementation considerations	80
2.12	Best practices flowchart	82
<b>Chapter 3.</b>	<b>Overview of IBM Tivoli Enterprise Console</b>	<b>85</b>
3.1	The highlights of IBM Tivoli Enterprise Console	86
3.2	Understanding the IBM Tivoli Enterprise Console data flow	87

3.2.1 IBM Tivoli Enterprise Console input . . . . .	88
3.2.2 IBM Tivoli Enterprise Console processing . . . . .	89
3.2.3 IBM Tivoli Enterprise Console output . . . . .	90
3.3 IBM Tivoli Enterprise Console components . . . . .	91
3.3.1 Adapter Configuration Facility . . . . .	91
3.3.2 Event adapter . . . . .	91
3.3.3 IBM Tivoli Enterprise Console gateway . . . . .	92
3.3.4 IBM Tivoli NetView . . . . .	92
3.3.5 Event server . . . . .	93
3.3.6 Event database . . . . .	93
3.3.7 User interface server . . . . .	93
3.3.8 Event console . . . . .	93
3.4 Terms and definitions . . . . .	94
3.4.1 Event . . . . .	94
3.4.2 Event classes . . . . .	94
3.4.3 Rules . . . . .	95
3.4.4 Rule bases . . . . .	97
3.4.5 Rule sets and rule packs . . . . .	98
3.4.6 State correlation . . . . .	99
<b>Chapter 4. Overview of IBM Tivoli NetView . . . . .</b>	<b>101</b>
4.1 IBM Tivoli NetView (Integrated TCP/IP Services) . . . . .	102
4.2 NetView visualization components . . . . .	104
4.2.1 The NetView EUI . . . . .	105
4.2.2 NetView maps and submaps . . . . .	106
4.2.3 The NetView event console . . . . .	112
4.2.4 The NetView Web console . . . . .	114
4.2.5 Smartsets . . . . .	117
4.2.6 How events are processed . . . . .	119
4.3 Supported platforms and installation notes . . . . .	120
4.3.1 Supported operating systems . . . . .	121
4.3.2 Java Runtime Environments . . . . .	121
4.3.3 AIX installation notes . . . . .	121
4.3.4 Linux installation notes . . . . .	123
4.4 Changes in NetView 7.1.3 and 7.1.4 . . . . .	124
4.4.1 New features and enhancements for Version 7.1.3 . . . . .	124
4.4.2 New features and enhancements for Version 7.1.4 . . . . .	126
4.4.3 First failure data capture . . . . .	130
4.5 A closer look at the new functions . . . . .	131
4.5.1 servmon daemon . . . . .	131
4.5.2 FFDC . . . . .	134
<b>Chapter 5. Overview of IBM Tivoli Switch Analyzer . . . . .</b>	<b>141</b>

5.1 The need for layer 2 network management. . . . .	142
5.1.1 Open Systems Interconnection model . . . . .	142
5.1.2 Why layer 3 network management is not always sufficient. . . . .	143
5.2 Features of IBM Tivoli Switch Analyzer V1.2.1 . . . . .	144
5.2.1 Daemons and processes . . . . .	144
5.2.2 Discovery . . . . .	146
5.2.3 Layer 2 status . . . . .	156
5.2.4 Integration into NetView's topology map. . . . .	157
5.2.5 Traps. . . . .	159
5.2.6 Root cause analysis using IBM Tivoli Switch Analyzer and NetView. . . . .	160
5.2.7 Real-life example . . . . .	161
<b>Chapter 6. Event management products and best practices . . . . .</b>	<b>173</b>
6.1 Filtering and forwarding events . . . . .	174
6.1.1 Filtering and forwarding with NetView. . . . .	174
6.1.2 Filtering and forwarding using IBM Tivoli Enterprise Console. . . . .	205
6.1.3 Filtering and forwarding using IBM Tivoli Monitoring . . . . .	210
6.2 Duplicate detection and throttling . . . . .	212
6.2.1 IBM Tivoli NetView and Switch Analyzer for duplicate detection and throttling . . . . .	212
6.2.2 IBM Tivoli Enterprise Console duplicate detection and throttling . . . . .	212
6.2.3 IBM Tivoli Monitoring for duplicate detection and throttling. . . . .	217
6.3 Correlation. . . . .	218
6.3.1 Correlation with NetView and IBM Tivoli Switch Analyzer . . . . .	218
6.3.2 IBM Tivoli Enterprise Console correlation. . . . .	232
6.3.3 IBM Tivoli Monitoring correlation. . . . .	244
6.4 Notification. . . . .	244
6.4.1 NetView. . . . .	245
6.4.2 IBM Tivoli Enterprise Console. . . . .	249
6.4.3 Rules. . . . .	251
6.4.4 IBM Tivoli Monitoring. . . . .	260
6.5 Escalation . . . . .	262
6.5.1 Severities . . . . .	263
6.5.2 Escalating events with NetView . . . . .	279
6.6 Event synchronization . . . . .	295
6.6.1 NetView and IBM Tivoli Enterprise Console . . . . .	295
6.6.2 IBM Tivoli Enterprise Console gateway and IBM Tivoli Enterprise Console. . . . .	296
6.6.3 Multiple IBM Tivoli Enterprise Console servers. . . . .	297
6.6.4 IBM Tivoli Enterprise Console and trouble ticketing . . . . .	302
6.7 Trouble ticketing . . . . .	307
6.7.1 NetView versus IBM Tivoli Enterprise Console. . . . .	307
6.7.2 IBM Tivoli Enterprise Console. . . . .	307



6.8 Maintenance mode .....	315
6.8.1 NetView .....	315
6.8.2 IBM Tivoli Enterprise Console .....	328
6.9 Automation .....	338
6.9.1 Using NetView for automation .....	338
6.9.2 IBM Tivoli Enterprise Console .....	351
6.9.3 IBM Tivoli Monitoring .....	354
<b>Chapter 7. A case study .....</b>	<b>357</b>
7.1 Lab environment .....	358
7.1.1 Lab software and operating systems .....	358
7.1.2 Lab setup and diagram .....	359
7.1.3 Reasons for lab layout and best practices .....	362
7.2 Installation issues .....	363
7.2.1 IBM Tivoli Enterprise Console .....	363
7.2.2 NetView .....	364
7.2.3 IBM Tivoli Switch Analyzer .....	364
7.3 Examples and related diagnostics .....	370
7.3.1 Event flow .....	370
7.3.2 IBM Tivoli Enterprise Console troubleshooting .....	377
7.3.3 NetView .....	394
7.3.4 IBM Tivoli Switch Analyzer .....	399
<b>Appendix A. Suggested NetView configuration .....</b>	<b>401</b>
Suggested NetView EUI configuration .....	402
Event console configuration .....	403
Web console installation .....	404
Web console stand-alone installation .....	404
Web console applet .....	406
Web console security .....	407
Web console menu extension .....	408
A smartset example .....	417
<b>Related publications .....</b>	<b>421</b>
IBM Redbooks .....	421
Other publications .....	421
Online resources .....	422
How to get IBM Redbooks .....	422
Help from IBM .....	422
<b>Index .....</b>	<b>423</b>



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

*The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:* INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

 e-server®

ibm.com®

zSeries®


AIX®

DB2 Universal Database™

DB2®

IBM®

NetView®

Redbooks (logo) ™

Redbooks™

S/390®

Tivoli Enterprise™

Tivoli Enterprise Console®

Tivoli®

TME®

WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

# Preface

This IBM Redbook presents a deep and broad understanding about event management with a focus on best practices. It examines event filtering, duplicate detection, correlation, notification, escalation, and synchronization. Plus it discusses trouble-ticket integration, maintenance modes, and automation in regard to event management.

Throughout this book, you learn to apply and use these concepts with IBM Tivoli® Enterprise™ Console 3.9, NetView® 7.1.4, and IBM Tivoli Switch Analyzer 1.2.1. Plus you learn about the latest features of these tools and how they fit into an event management system.

This redbook is intended for system and network administrators who are responsible for delivering and managing IT-related events through the use of systems and network management tools. Prior to reading this redbook, you should have a thorough understanding of the event management system in which you plan to implement these concepts.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Austin Center.

**Tony Bhe** is an IT Specialist for IBM in the United States. He has eight years of experience in the IT industry with seven years of direct experience with IBM Tivoli Enterprise products. He holds a degree in electrical engineering from North Carolina State University in Raleigh, North Carolina. His areas of expertise include Tivoli performance, availability, configuration, and operations. He has spent the last three years working as a Tivoli Integration Test Lead. One year prior to that, he was a Tivoli Services consultant for Tivoli Performance and Availability products.

**Peter Glasmacher** is a certified Systems Management expert from Dortmund, Germany. He joined IBM in 1973 and worked in various positions including support, development, and services covering multiple operating system platforms and networking architectures. Currently, he works as a consulting IT specialist for the Infrastructure & Technology Services branch of IBM Global Services. He concentrates on infrastructure and security issues. He has more than 16 years of experience in the network and systems management areas. For

the past nine years, he concentrated on architectural work and the design of network and systems management solutions in large customer environments. Since 1983, he has written extensively on workstation-related issues. He has co-authored several IBM Redbooks™, covering network and systems management topics.

**Jacqueline Meckwood** is a certified IT Specialist in IBM Global Services. She has designed and implemented enterprise management systems and connectivity solutions for over 20 years. Her experience includes the architecture, project management, implementation, and troubleshooting of systems management and networking solutions for distributed and mainframe environments using IBM, Tivoli, and OEM products. Jacqueline is a lead Event Management and Monitoring Design (EMMD) practitioner and is an active member of the IT Specialist Board.

**Guilherme Pereira** is a Tivoli and Micromuse certified consultant at NetControl, in Brazil. He has seven years of experience in the network and systems management field. He has worked in projects in some of the largest companies in Brazil, mainly in the Telecom area. He holds a degree in business from Pontificia Universidade Catolica-RS, with graduate studies in business management from Universidade Federal do Rio Grande do Sul. His areas of expertise include network and systems management and project management. He is member of PMI and is a certified Project Management Professional.

**Michael Wallace** is a Enterprise Systems Management Engineer at Shaw Industries Inc. in Dalton, Georgia, U.S.A. He has five years of experience in the Systems Management field and spent time working in the Help Desk field. He holds a degree in PC/LAN from Brown College, MN. His areas of expertise include IBM Tivoli Enterprise Console® rule writing and integration with trouble-ticketing systems as well as event management and problem management.

Thanks to the following people for their contributions to this project:

Becky Anderson  
Cesar Araujo  
Alesia Boney  
Jim Carey  
Christopher Haynes  
Mike Odom  
Brian Pate  
Brooke Upton  
Michael L. Web  
IBM Software Group

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an Internet note to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- Mail your comments to:

IBM® Corporation, International Technical Support Organization  
Dept. JN9B Building 003 Internal Zip 2834  
11400 Burnet Road  
Austin, Texas 78758-3493







# Introduction to event management

This chapter explains the importance of event correlation and automation. It defines relevant terminology and introduces basic concepts and issues. It also discusses general planning considerations for developing and implementing a robust event management system.

## 1.1 Importance of event correlation and automation

From the time of their inception, computer systems were designed to serve the needs of businesses. Therefore, it was necessary to know if they were operational. The critical need of the business function that was performed governed how quickly this information had to be obtained.

Early computers were installed to perform batch number-crunching tasks for such business functions as payroll and accounts receivable, in less time and with more efficiency than humans could perform them. Each day, the results of the batch processing were examined. If problems occurred, they were resolved and the batch jobs were executed again.

As their capabilities expanded, computers began to be used for functions such as order entry and inventory. These mission-critical applications needed to be online and operational during business hours required immediate responses. Companies questioned the reliability of computers and did not want to risk losing customers because of computer problems. Paper forms and manual backup procedures provided insurance to companies that they could still perform their primary business in the event of a computer failure.

Since these batch and online applications were vital to the business of the company, it became more important to ascertain in a timely fashion whether they were available and working properly. Software was enhanced to provide information and errors, which were displayed on one or more consoles. Computer operators watched the consoles, ignored the informational messages, and responded to the errors. Tools became available to automatically reply to messages that always required the same response.

With the many advances in technology, computers grew more sophisticated and were applied to more business functions. Personal computers and distributed systems flourished, adding to the complexity of the IT environment. Due to the increased reliability of the machines and software, it became impractical to run a business manually. Companies surrendered their paper forms and manual backup procedures to become completely dependent upon the functioning of the computer systems.

Managing the systems, now critical to the survival of a business, became the responsibility of separate staffs within an IT organization. Each team used its own set of tools to do the necessary monitoring of its own resources. Each viewed its own set of error messages and responded to them. Many received phone calls directly from users who experienced problems.

To increase the productivity of the support staffs and to offload some of their problem support responsibilities, help desks were formed. Help desks served as

central contact points for users to report problems with their computers or applications. They provided initial problem determination and resolution services. The support staffs did not need to watch their tools for error messages, since software was installed to aggregate the messages at a central location. The help desk or an operations center monitored messages from various monitoring tools and notified the appropriate support staff when problems surfaced.

Today, changes in technology provide still more challenges. The widespread use of the Internet to perform mission-critical applications necessitates 24 X 7 availability of systems. Organizations need to know immediately when there are failures, and recovery must be almost instantaneous. On-demand and grid computing allow businesses to run applications wherever cycles are available to ensure they can meet the demands of their customers. However, this increases the complexity of monitoring the applications, since it is now insufficient to know the status of one system without knowing how it relates to others. Operators cannot be expected to understand these relationships and account for them in handling problems, particularly in complex environments.

There are several problems with the traditional approach to managing systems:

- ▶ Missed problems

Operators can overlook real problems while sifting through screens of informational messages. Users may call to report problems before they are noticed and acted upon by the operator.

- ▶ False alarms

Messages can seem to indicate real problems, when in fact they are not. Sometimes additional data may be needed to validate the condition and, in distributed environments, that information may come from a different system than the one reporting the problem.

- ▶ Inconsistency

Various operators can respond differently to the same type of events.

- ▶ Duplication of effort

Multiple error messages may be produced for a single problem, possibly resulting in more than one support person handling the same problem.

- ▶ Improper problem assignment

Manually routing problems to the support staffs sometimes results in support personnel being assigning problems that are not their responsibility.

- ▶ Problems that cannot be diagnosed

Sometimes when an intermittent problem condition clears before someone has had the chance to respond to it, the diagnostic data required to determine the cause of the problem disappears.

Event correlation and automation address these issues by:

- ▶ Eliminating information messages from view to easily identify real problems
- ▶ Validating problems
- ▶ Responding consistently to events
- ▶ Suppressing extraneous indications of a problem
- ▶ Automatically assigning problems to support staffs
- ▶ Collecting diagnostic data

Event correlation and automation are the next logical steps in the evolution of event handling. They are critical to successfully managing today's ever-changing, fast-paced IT environments with the reduced staffs with which companies are forced to operate.

## 1.2 Terminology

Before we discuss the best ways to implement event correlation and automation, we need to establish the meaning of the terms we use. While several systems management terms are generally used to describe event management, these terms are sometimes used in different ways by different authors. In this section, we provide definitions of the terms as they are used throughout this redbook.

### 1.2.1 Event

Since event management and correlation center around the processing of events, it is important to clearly define what is meant by an event. In the context of this redbook, an *event* is a piece of data that provides information about one or more system resources.

Events can be triggered by incidents or problems affecting a system resource. Similarly, changes to the status or configuration of a resource, regardless of whether they are intentional, can generate events. Events may also be used as reminders to take action manually or as notification that an action has occurred.

### 1.2.2 Event management

The way in which an organization deals with events is known as *event management*. It may include the organization's objectives for managing events, assigned roles and responsibilities, ownership of tools and processes, critical success factors, standards, and event-handling procedures. The linkages between the various departments within the organization required to handle events and the flow of this information between them is the focus of event management. Tools are mentioned in reference to how they fit into the flow of

event information through the organization and to which standards should be applied to that flow.

Since events are used to report problems, event management is sometimes considered a sub-discipline of problem management. However, it can really be considered a discipline of its own, for it interfaces directly with several other systems management disciplines. For example, system upgrades and new installations can result in new event types that must be handled. Maintaining systems both through regularly scheduled and emergency maintenance can result in temporary outages that trigger events. This clearly indicates a relationship between event management and change management.

In small organizations, it may be possible to handle events through informal means. However, as organizations grow both in size of the IT support staffs and the number of resources they manage, it becomes more crucial to have a formal, documented event management process. Formalizing the process ensures consistent responses to events, eliminates duplication of effort, and simplifies the configuration and maintenance of the tools used for event management.

### 1.2.3 Event processing

While event management focuses on the high-level flow of events through an organization, event processing deals with tools. Specifically, the term *event processing* is used to indicate the actions taken upon events automatically by systems management software tools.

Event processing includes such actions as changing the status or severity of an event, dropping the event, generating problem tickets and notifications, and performing recovery actions. These actions are explained in more detail in 1.3, “Concepts and issues” on page 6.

### 1.2.4 Automation and automated actions

*Automation* is a type of actions that can be performed when processing events. For the purposes of this book, it refers to the process of taking actions on system resources without human intervention in response to an event. The actual actions executed are referred to as *automated actions*.

Automated actions may include recovery commands performed on a failing resource to restore its service and failover processes to bring up backup resources. Changing the status or severity of an event, closing it, and similar functions are not considered automated actions. That is because they are performed on the event itself rather than on one or more system resources referred to or affected by the event.

The types of automated actions and their implications are covered in more detail in 1.3, “Concepts and issues” on page 6.

## 1.3 Concepts and issues

This section presents the concepts and issues associated with event processing. Additional terminology is introduced as needed.

### 1.3.1 Event flow

An event cannot provide value to an organization in managing its system resources unless the event is acted upon, either manually by a support person or automatically by software. The path an event takes from its source to the software or person who takes action on it is known as the *event flow*.

The event flow begins at the point of generation of the event, known as the *event source*. The source of an event may be the failing system itself, as in the case of a router that sends information about its health to an event processor. An agent that runs on the system to monitor for and report error conditions is another type of event source. A proxy systems that monitor devices other than itself, such as Simple Network Management Protocol (SNMP) manager that periodically checks the status of TCP/IP devices, and reports a failure if it receives no response, is also considered an event source.

Event processors are devices that run software capable of recognizing and acting upon events. The functionality of the event processors can vary widely. Some are capable of merely forwarding or discarding events. Others can perform more sophisticated functions such as reformatting the event, correlating it with other events received, displaying it on a console, and initiating recovery actions.

Most event processors have the capability to forward events to other event processors. This functionality is useful in consolidating events from various sources at a central site for management by a help desk or operations center. The hierarchy of event processors used to handle events can be referred to as the *event processing hierarchy*. The first receiver of the event is the *entry point into the hierarchy*, and the collection of all the entry points is called the *entry tier of the hierarchy*. Similarly, all second receivers of events can be collectively referred to as the *second tier in the hierarchy* and so forth. For the purposes of this book, we refer to the top level of the hierarchy as the *enterprise tier*, because it typically consolidates events from sources across an entire enterprise.

Operators typically view events of significance from a console, which provides a graphical user interface (GUI) through which the operator can take action on events. Consoles can be proprietary, requiring special software for accessing the

console. Or they can adhere to open standards, such as Web-based consoles that can be accessed from properly configured Web browsers.

The collection of event sources, processors, and consoles is sometimes referred to as the *event management infrastructure*.

### 1.3.2 Filtering and forwarding

Many devices generate informational messages that are not indicative of problems. Sending these messages as events through the event processing hierarchy is undesirable. The reason is because processing power and bandwidth are needed to handle them and they clutter the operator consoles, possibly masking true problems. The process of suppressing these messages is called *event filtering* or *filtering*.

There are several ways to perform event filtering. Events can be prevented from ever entering the event processing hierarchy. This is referred to as *filtering at the source*. Event processors can discard or drop the unnecessary events. Likewise, consoles can be configured to hide them from view.

The event filtering methods that are available are product specific. Some SNMP devices, for example, can be configured to send all or none of their messages to an event processor or to block messages within specific categories such as security or configuration. Other devices allow blocking to be configured by message type.

When an event is allowed to enter the event processing hierarchy, it is said to be *forwarded*. Events can be forwarded from event sources to event processors and between event processors. Chapter 2, “Event management categories and best practices” on page 25, discusses the preferred methods of filtering and forwarding events.

### 1.3.3 Duplicate detection and throttling

Events that are deemed necessary must be forwarded to at least one event processor to ensure that they are handled by either manual or automated means. However, sometimes the event source generates the desired message more than once when a problem occurs. Usually, only one event is required for action. The process of determining which events are identical is referred to as *duplicate detection*.

The time frame in which a condition is responded to may vary, depending upon the nature of the problem being reporting. Often, it should be addressed immediately when the first indication of a problem occurs. This is especially true in situations where a device or process is down. Subsequent events can then be

discarded. Other times, a problem does not need to be investigated until it occurs several times. For example, a high CPU condition may not be a problem if a single process, such as a backup, uses many cycles for a minute or two. However, if the condition happens several times within a certain time interval, there most likely is a problem. In this case, the problem should be addressed after the necessary number of occurrences. Unless diagnostic data, such as the raw CPU busy values, is required from subsequent events, they can be dropped. The process of reporting events after a certain number of occurrences is known as *throttling*.

### 1.3.4 Correlation

When multiple events are generated as a result of the same initial problem or provide information about the same system resource, there may be a relationship between the events. The process of defining this relationship in an event processor and implementing actions to deal with the related events is known as *event correlation*.

Correlated events may reference the same affected resource or different resources. They may be generated by the same event source or handled by the same event processor.

#### Problem and clearing event correlation

This section presents an example of events that are generated from the same event source and deal with the same system resource. An agent monitoring a system detects that a service has failed and sends an event to an event processor. The event describes an error condition, called a *problem event*. When the service is later restored, the agent sends another event to inform the event processor the service is again running and the error condition has cleared. This event is known as a *clearing event*. When an event processor receives a clearing event, it normally closes the problem event to show that it is no longer an issue.

The relationship between the problem and clearing event can be depicted graphically as shown in Figure 1-1. The correlation sequence is described as follows:

- ▶ Problem is reported when received (Service Down).
- ▶ Event is closed when a recovery event is received (Service Recovered).

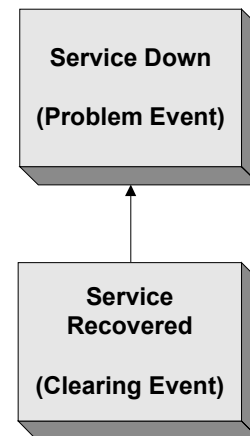


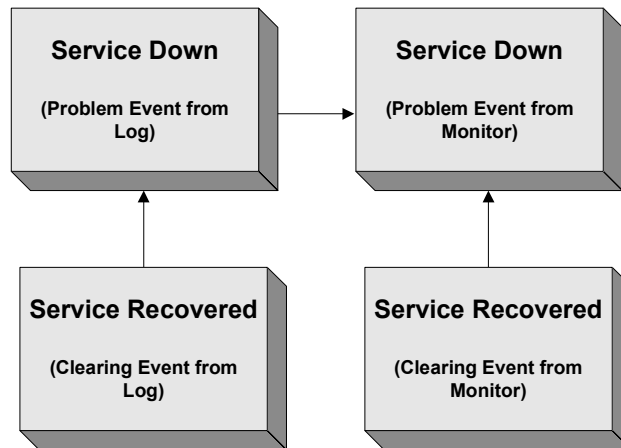
Figure 1-1 Problem and clearing correlation sequence



Taking this example further, assume that multiple agents are on the system. One reads the system log, extracts error messages, and sends them as events. The second agent actively monitors system resources and generates events when it detects error conditions. A service running on the system writes an error message to the system log when it dies. The first agent reads the log, extracts the error messages, and sends it as an event to the event processor. The second agent, configured to monitor the status of the service, detects that it has stopped and sends an event as well. When the service is restored, the agent writes a message to the system log, which is sent as an event, and the monitor detects the recovery and sends its own event.

The event processor receives both problem events, but only needs to report the service failure once. The events can be correlated and one of them dropped. Likewise, only one of the clearing events is required. This correlation sequence is shown in Figure 1-2 and follows this process:

- ▶ A problem event is reported if received from the log.
- ▶ The event is closed when the Service Recovered event is received from the log.
- ▶ If a Service Down event is received from a monitor, the Service Down event from the log takes precedence, and the Service Down event from a monitor becomes extraneous and is dropped.
- ▶ If a Service Down event is not received from the log, the Service Down event from a monitor is reported and closed when the Service Recovered event is received from the monitor.



*Figure 1-2 Correlation of multiple events reporting the same problem*

This scenario is different from duplicate detection. The events being correlated both report service down, but they are from different event sources and most likely have different formats. Duplicate detection implies that the events are of the same format and are usually, though not always, from the same event source. If the monitoring agent in this example detects a down service, and repeatedly sends events reporting that the service is down, these events can be handled with duplicate detection.

## Event escalation

Sometimes multiple events are sent to indicate a worsening error condition for a system resource. For example, an agent monitoring a file system may send a warning message to indicate the file system is greater than 90% full, a second, more severe event when greater than 95% full, and a critical event greater than 98% full. In this case, the event processor does not need to report the file system error multiple times. It can merely increase the severity of the initial event to indicate that the problem has become more critical and needs to be responded to more quickly.

This type of correlation is sometimes called an *escalation sequence*. In Figure 1-3, the escalation sequence is described as follows:

- ▶ The problem on the far left is received and reported.
- ▶ Event severity of the reported event is escalated when subsequent events are received (shown to its right) and those events are dropped.
- ▶ The reported event is closed when the clearing event is received.

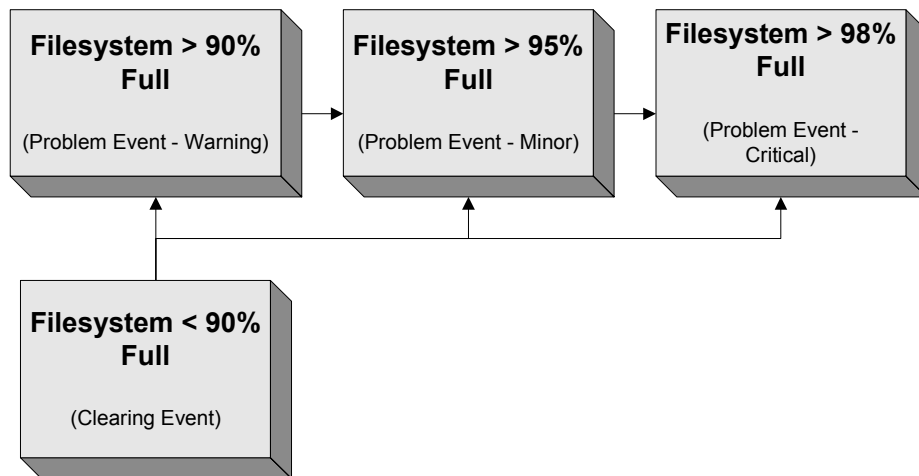


Figure 1-3 Escalation sequence

For example, if Filesystem > 90% Full is received, it is reported as a warning. When Filesystem > 95% Full is received, it is dropped and the reported event is escalated to a severe. Likewise, if Filesystem > 98% Full is received, it is dropped and the reported event is escalated again to a critical.

If Filesystem > 95% Full is the first problem event received, it is reported. The same escalation logic applies. This type of correlation sequence assumes that severities are clearly defined and the allowable time to respond to events of those severities has been communicated within the organization. This is one of the

purposes of the event management process described in 1.2.2, “Event management” on page 4.

### **Root cause correlation**

A problem may sometimes trigger other problems, and each problem may be reported by events. The event reporting the initial problem is referred to as a *root cause*, or *primary event*. Those that report the subsequent problems are called *symptom* or *secondary events*.

At this point, it is important to note the difference between a root cause event and the root cause of a problem. The former is the event that provides information about the first of a series of related, *reported* problems. The latter is what caused the problem condition to happen.

Root cause events and root causes of problems may be closely related. For example, a root cause event reporting a faulty NIC card may be correlated with secondary events such as “Interface Down” from an SNMP manager or “Application unreachable” from a transaction monitoring agent. The root cause of the problem is the broken card.

However, sometimes the two are not as closely associated. Consider an event that reports a *Filesystem Full* condition. The full file system may cause a process or service to die, producing a secondary event. The Filesystem Full event is the root cause event, but it is not the root cause of the problem. A looping application that is repeatedly logging information into the file system may be the root cause of the problem.

When situations such as these are encountered, you must set up monitoring to check for the root cause of the problem and produce an event for it. That event then becomes the root cause event in the sequence. In our example, a monitor that detects and reports looping application logging may be implemented. The resulting event can then be correlated with the others and becomes the root cause event.

Because of this ambiguity in terms, we prefer to use the term *primary event* rather than root cause event.

The action taken in response to a root cause event may automatically resolve the secondary problems. Sometimes, though, a symptom event may require a separate action, depending upon the nature of the problem it reports. Examples of each scenario follow.

### ***Symptom events not requiring action***

Assume that an agent on a UNIX® system is monitoring file systems for adequate space and critical processes for availability. One of the key processes

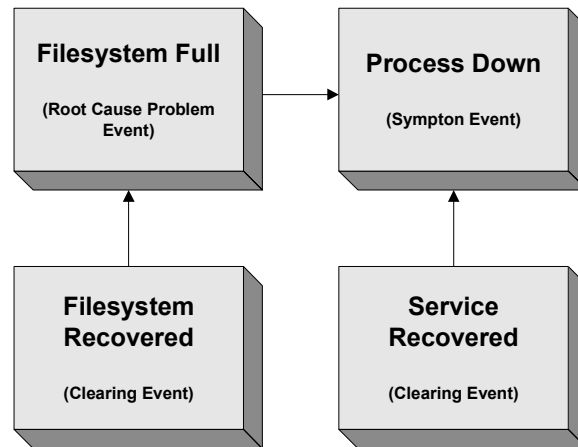
is required to run at all times and is set up to automatically respawn if it fails. The process depends upon adequate free space in the file system where it stores its temporary data files and cannot execute without it.

The file system upon which the process depends fills up, and the agent detects the condition and sends an event. The process dies, and the operating system unsuccessfully attempts to restart it repeatedly. The agent detects the failure and generates a second event to report it.

There are essentially two problems here. The primary problem is the full file system, and the process failure is the secondary problem. When appropriate action is taken on the first event to free space within the file system, the process successfully respawns automatically. No action is required on the secondary event, so the event processor can discard it.

In Figure 1-4, the correlation sequence is described as follows:

- ▶ The Filesystem Full event is reported if received.
- ▶ The Process Down event is unnecessary and is dropped. Since the process is set to respawn, it automatically starts when the file system is recovered.
- ▶ The Filesystem Full event is closed when the Filesystem Recovered clearing event is received.
- ▶ The Service Recovered clearing event is unnecessary and is dropped, since it is superseded by the Filesystem Recovered clearing event.



*Figure 1-4 Correlation sequence in which secondary event does not require action*

### **Symptom events requiring action**

Now suppose that an application stores its data in a local database. An agent runs on the application server to monitor the availability of both the application and the database. A database table fills and cannot be extended, causing the application to hang. The agent detects both conditions and sends events to report them.

The full database table is the primary problem, and the hanging application is the secondary problem. A database administrator corrects the primary problem. However, the application is hung and cannot recover itself. It must be recycled.

Since restarting the application is outside the responsibility of the database administrator, the secondary event is needed to report the application problem to the appropriate support person.

In Figure 1-5, the correlation sequence is as follows:

- ▶ The Filesystem Full event is reported if received.
- ▶ The Process Down event is reported as dependent upon the file system being resolved.
- ▶ The Filesystem Full event is closed when the Filesystem Recovered clearing event is received.
- ▶ The Process Down event is cleared when the Service Recovered clearing event is received.

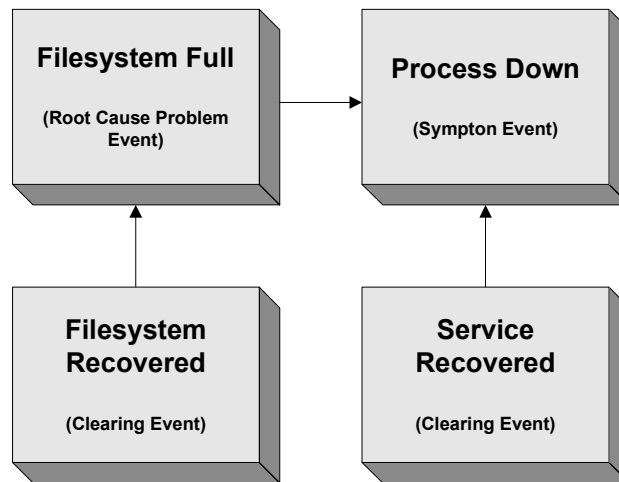


Figure 1-5 Correlation sequence in which secondary event requires action

An important implication of this scenario must be addressed. Handling the secondary event depends upon the resolution of the primary event. Until the database is repaired, any attempts to restart the application fail. Implementation of correlation sequences of this sort can be challenging. Chapter 6, “Event management products and best practices” on page 173, discusses ways to implement this type of correlation sequence using IBM Tivoli Enterprise Console V3.9.

## Cross-platform correlation

In the previous application and database correlation scenario, the correlated events refer to different types of system resources. We refer to this as *cross-platform correlation*. Some examples of platforms include operating systems, databases, middleware, applications, and hardware.

Often, cross-platform correlation sequences result in symptom events that require action. This is because the support person handling the first resource type does not usually have administrative responsibility for the second type. Also, many systems are not sophisticated enough to recognize the system resources affected by a failure and to automatically recover them when the failure is

resolved. For these reasons, cross-platform correlation sequences provide an excellent opportunity for automated recovery actions.

## **Cross-host correlation**

In distributed processing environments, there are countless situations in which conditions on one system affect the proper functioning of another system. Web applications, for example, often rely on a series of Web, application, and database servers to run a transaction. If a database is inaccessible, the transaction fails. Likewise, servers may share data through message queuing software, requiring the creation of the queue by one server before it is accessed from another.

When problems arise in scenarios such as these, events can be generated by multiple hosts to report a problem. It may be necessary to correlate these events to determine which require action. The process of correlating events from different systems is known as *cross-host correlation*.

In the example presented in “Symptom events requiring action” on page 12, the database can easily reside on a different server than the application accessing it. The event processor takes the same actions on each event as described previously. However, it has the additional burden of checking the relationship between hosts before determining if the events correlate. Cross-host correlation is particularly useful in clustered and failover environments. For clusters, some conditions may not represent problems unless they are reported by all systems in the cluster. As long as one system is successfully running an application, for example, no action is required. In this case, the event processor needs to know which systems constitute the cluster and track which systems report the error.

In failover scenarios, an error condition may require action if it is reported by either host. Consider, for example, paired firewalls. If the primary firewall fails and the secondary takes over, each may report the switch, and cross-host correlation may be used to report failure of the primary. However, a hard failure of the primary may mean that the failover event is sent only by the secondary. This event should indicate the failure of the primary firewall as the condition that requires action. Again, the event processor needs to know the relationship between the firewalls before correlating failover events.

See 6.6, “Event synchronization” on page 295, to learn about ways in which cross-host correlation can be implemented using IBM Tivoli Enterprise Console.

## **Topology-based correlation**

When such networking resources as routers fail, they may cause a large number of other systems to become inaccessible. In these situations, events may be reported that refer to several unreachable system resources. The events may be reported by SNMP managers that receive no answer to their status queries or by

systems that can no longer reach resources with which they normally communicate. Correlating these events requires knowledge of the network topology, and therefore are referred to as *topology-based correlation*.

This type of correlation, while similar to cross-host correlation, differs in that the systems have a hierarchical, rather than a peer, relationship. The placement of the systems within the network determines the hierarchy. The failure of one networking component affects the resources downstream from it.

Clearly, the event reporting the failing networking resource is the primary, or root, cause event and needs to be handled. Often, the secondary events refer to unreachable resources that become accessible once the networking resource is restored to service. In this case, these events may be unnecessary. Sometimes, however, a downstream resource may need to be recycled to resynchronize it with its peer resources. Secondary events dealing with these resources require corrective action.

Since SNMP managers typically discover network topology and understand the relationships between devices, they are often used to implement topology-based correlation. In 6.3, “Correlation” on page 218, we discuss how these products perform topology-based correlation.

### **Timing considerations**

An important consideration in performing event correlation is the timing of the events. It is not always the case that the primary event is received first. Network delays may prevent the primary event from arriving until after the secondary is received. Likewise, in situations where monitoring agents are scheduled to check periodically for certain conditions, the monitor that checks for the secondary problem may run first and produce that event before the root cause condition is checked.

To properly perform event correlation in this scenario, configure the event processor to wait a certain amount of time to ensure that the primary condition does not exist before reporting that action is required for the secondary event. The interval chosen must be long enough to allow the associated events to be received, but short enough to minimize the delay in reporting the problem.

See Chapter 6, “Event management products and best practices” on page 173, to learn about methods for implementing this using IBM Tivoli Enterprise Console.

## **1.3.5 Event synchronization**

When events are forwarded through multiple tiers of the event management hierarchy, it is likely that different actions are performed on the event by different

event processors. These actions may include correlating, dropping, or closing events.

Problems can arise when one event processor reports that an event is in a certain state and another reports that it is in a different state. For example, assume that the problem reported by an event is resolved, and the event is closed at the central event processor but not at the event processors in the lower tiers in the hierarchy. The problem recurs, and a new event is generated. The lower-level event processor shows an outstanding event already reporting the condition and discards the event. The new problem is never reported or resolved.

To ensure that this situation does not happen, status changes made to events at one event processor can be propagated to the others through which the event has passed. This process is known as *event synchronization*.

Implementing event synchronization can be challenging, particularly in complex environments with several tiers of event processors. Also, environments designed for high availability need some way to synchronize events between their primary and backup event processors. Chapter 6, “Event management products and best practices” on page 173, addresses the event synchronization methods available in IBM Tivoli Enterprise Console V3.9, with its NetView Integrated TCP/IP Services Component V7.1.4 and IBM Tivoli Switch Analyzer V1.2.1.

### 1.3.6 Notification

Notification is the process of informing support personnel that an event has occurred. It is typically used to supplement use of the event processor’s primary console, not to replace it. Notification is useful in situations when the assigned person does not have access to the primary console, such after hours, or when software licensing or system resource constraints prevent its use. It can also be helpful in escalating events that are not handled in a timely manner (see 1.3.8, “Escalation” on page 17).

Paging, e-mail, and pop-up windows are the most common means of notification. Usually, these functions exist outside the event processor’s software and must be implemented using an interface. Sometimes that interface is built into the event processor. Often, the event processor provides the ability to execute scripts or BAT files that can be used to trigger the notification software. This is one of the simplest ways to interface with the notification system.

It is difficult to track the various types of notifications listed previously, and the methods are often unreliable. In environments where accountability is important, more robust means may be necessary to ensure that support personnel are informed about events requiring their action.



The acceptable notification methods and how they are used within an organization should be covered in the event management process, which is described in 1.2.2, “Event management” on page 4.

### 1.3.7 Trouble ticketing

Problems experienced by users can be tracked using *trouble tickets*. The tickets can be opened manually by the help desk or operations center in response to a user’s phone call or automatically by an event processor.

Trouble ticketing is one of the actions that some event processors can take upon receipt of an event. It refers to the process of forwarding the event to a trouble-ticketing system in a format that system can understand. This can typically be implemented by executing a script or sending an e-mail to the trouble-ticketing system’s interface or application programming interface (API).

The trouble-ticketing system itself can be considered a special type of event processor. It can open trouble tickets for problem events and close them when their corresponding clearing events are received. As such, it needs to be synchronized with the other event processors in the event management hierarchy. The actions of opening and closing trouble tickets are also referred to as *trouble ticketing*.

In environments where accountability is important, robust trouble-ticketing systems may provide the tracking functions needed to ensure that problems are resolved by the right people in a timely manner.

### 1.3.8 Escalation

In 1.3.4, “Correlation” on page 8, we discuss escalating the severity of events based on the receipt of related events. This escalation is handled by the event source, which sends increasingly more critical events as a problem worsens. There are a few kinds of event escalation that require consideration.

#### **Escalation to ensure problems are addressed**

An event is useless in managing IT resources if no action is taken to resolve the problem reported. A way to ensure that an event is handled is for an event processor to escalate its severity if it has not been acknowledged or closed within an acceptable time frame. Timers can be set in some event processors to automatically increase the severity of an event if it remains in an unacknowledged state.

The higher severity event is generally highlighted in some fashion to draw greater attention to it on the operator console on which it is displayed. The operators

viewing the events may inform management that the problem has not been handled, or this notification may be automated.

In addition to serving as a means of ensuring that events are not missed, escalation is useful in situations where the IT department must meet service-level agreements (SLAs). The timers may be set to values that force escalation of events, indicating to the support staff that the event needs to be handled quickly or SLAs may be violated.

For escalation to be implemented, the allowable time frames to respond to events of particular severities and the chain of people to inform when the events are not handled must be clearly defined. This is another purpose of the event management process described in 1.2.2, “Event management” on page 4.

## **Business impact escalation**

Events can also be escalated based upon business impact. Problems that affect a larger number of users should be resolved more quickly than those that impact only a few users. Likewise, failures of key business applications should be addressed faster than those of less important applications.

There are several ways to escalate events based upon their business significance:

- ▶ **Device type**

An event may be escalated when it is issued for a certain device type. Router failures, for example, may affect large numbers of users because they are critical components in communication paths in the network. A server outage may affect only a handful of users who regularly access it as part of their daily jobs. When deploying this type of escalation, the event processor checks to see the type of device that failed and sets the severity of the event accordingly. In our example, events for router failures may be escalated to a higher severity while events of servers remain unchanged.

- ▶ **Device priority**

Some organizations perform asset classifications in which they evaluate the risk to the business of losing various systems. A switch supporting 50 users may be more critical than a switch used by five users. In this escalation type, the event processor checks the risk assigned to the device referenced in an event and increases the severity of those with a higher rating.

- ▶ **Other**

It is also possible to perform escalation based on which resources a system fails, assigning different priorities to the various applications and services that run on a machine. Another hybrid approach combines device type and priority to determine event severity. For example, routers may take higher priority than

servers. The routers are further categorized by core routers for the backbone network and distributed routers for the user rings, with the core routers receiving heavier weighting in determining event severity.

An organization should look at its support structure, network architecture, server functions, and SLAs to determine the best approach to use in handling event escalation.

### 1.3.9 Maintenance mode

When administrative functions performed on a system disrupt its normal processing, the system is said to be in *maintenance mode*. Applying fixes, upgrading software, and reconfiguring system components are all examples of activities that can put a system into maintenance mode.

Unless an administrator stops the monitoring agents on the machine, events continue to flow while the system is maintained. These events may relate to components that are affected by the maintenance or to other system resources. In the former case, the events do not represent real problems, but in the latter case, they may.

From an event management point of view, the difficulty is how to handle systems that are in maintenance mode. Often, it is awkward to reconfigure the monitoring agents to temporarily ignore only the resources affected by the maintenance. Shutting down monitoring completely may suppress the detection and reporting of a real problem that has nothing to do with the maintenance. Both of these approaches rely on the intervention of the administrator to stop and restart the monitoring, which may not happen, particularly during late night maintenance windows.

Another problem is that maintenance may cause a chain reaction of events generated by other devices. A server that is in maintenance mode may only affect a few machines with which it has contact during normal operations. A network device may affect large portions of the network when maintained, causing a flood of events to occur.

How to predict the effect of the maintenance, and how to handle it are issues that need to be addressed. See 2.10, “Maintenance mode” on page 72, for suggestions on how to handle events from machines in maintenance mode.

### 1.3.10 Automation

You can perform four basic types of automated actions upon receipt of an event:

- Problem verification

It is not always possible to filter events that are not indicative of real problems. For example, an SNMP manager that queries a device for its status may not receive an answer due to network congestion rather than the failure of the device. In this case, the manager believes the device is down. Further processing is required to determine whether the device is really operational. This processing can be automated.

- Recovery

Some failure conditions lend themselves to automated recovery. For example, if a service or process dies, it can generally be restarted using a simple BAT file or script.

- Diagnostics

If diagnostic information is typically obtained by the support person to resolve a certain type of problem, that information can be gathered automatically when the failure occurs and merely accessed when needed. This can help to reduce the mean-time to repair for the problem. It is also particularly useful in cases where the diagnostic data, such as the list of processes running during periods of high CPU usage, may disappear before a support person has time to respond to the event.

- Repetitive command sequences

When operators frequently enter the same series of commands, automation can be built to perform those commands. The automated action can be triggered by an event indicating that it is time to run the command sequence. Environments where operators are informed by events to initiate the command sequences, such as starting or shutting down applications, lend themselves well to this type of automation.

Some events traverse different tiers of the event processing hierarchy. In these cases, you must decide at which place to initiate the automation. The capabilities of the tools to perform the necessary automated actions, security required to initiate them, and bandwidth constraints are some considerations to remember when deciding from which event processor to launch the automation.

## 1.4 Planning considerations

Depending upon the size and complexity of the IT environment, developing an event management process for it can be a daunting task. This section describes some points to consider when planning for event correlation and automation in support of the process.

### 1.4.1 IT environment assessment

A good starting point is to assess the current environment. Organizations should inventory their hardware and software to understand better the types of system resources managed and the tools used to manage them. This step is necessary to determine the event sources and system resources within scope of the correlation and automation effort. It is also necessary to identify the support personnel who can assist in deciding the actions needed for events related to those resources.

In addition, the event correlation architect should research the capabilities of the management tools in use and how the tools exchange information. Decisions about where to filter events or perform automated actions, for example, cannot be made until the potential options are known.

To see the greatest benefit from event management in the shortest time, organizations should target those event sources and system resources that cause the most pain. This information can be gathered by analyzing the volumes of events currently received at the various event processors, trouble-ticketing system reports, database queries, and scripts can help to gain an idea about the current event volumes, most common types of errors, and possible opportunities for automated action.

IBM offers a service to analyze current event data. This offering, called the Data Driven Event Management Design (DDEMD), uses a proprietary data-mining tool to help organizations determine where to focus their efforts. The tool also provides statistical analysis to suggest possible event correlation sequences and can help uncover problems in the environment.

### 1.4.2 Organizational considerations

Any event correlation and automation design needs to support the goals and structure of an organization. If event processing decisions are made without understanding the organization, the results may be disappointing. The event management tools may not be used, problems may be overlooked, or perhaps information needed to manage service levels may not be obtained.

To ensure that the event correlation project is successful, its design and processes should be developed with organizational considerations in mind.

#### **Centralized versus decentralized**

An organization's approach to event management is key to determine the best ways to implement correlation and automation. A centralized event management environment is one in which events are consolidated at a focal point and

monitored from a central console. This provides the ability to control the entire enterprise from one place. It is necessary to view the business impact of failures.

Since the operators and help desk personnel at the central site handle events from several platforms, they generally use tools that simplify event management by providing a common graphical interface to update events and perform basic corrective actions. When problems require more specialized support personnel to resolve, the central operators often are the ones to contact them.

Decentralized event management does not require consolidating events at a focal point. Rather, it uses distributed support staffs and toolsets. It is concerned with ensuring that the events are routed to the proper place. This approach may be used in organizations with geographically dispersed support staffs or point solutions for managing various platforms.

When designing an event correlation and automation solution for a centralized environment, the architect seeks commonality in the look and feel of the tools used and in the way events are handled. For decentralized solutions, this is less important.

## **Skill levels**

The skill level of those responsible for responding to events influences the event correlation and automation implementation. Highly skilled help desk personnel may be responsible for providing first level support for problems. They may be given tools to debug and resolve basic problems. Less experienced staff may be charged with answering user calls and dispatching problems to the support groups within the IT organization.

Automation is key to both scenarios. Where first level support skills are strong, semi-automated tasks can be set up to provide users the ability to easily execute the repetitive steps necessary to resolve problems. In less experienced environments, full automation may be used to gather diagnostic data for direct presentation to the support staffs who will resolve them.

## **Tool usage**

How an organization plans to use its systems management tools must be understood before event correlation can be successfully implemented. Who will use each tool and for what functions should be clearly defined. This ensures that the proper events are presented to the appropriate people for their action.

For example, if each support staff has direct access to the trouble-ticketing system, the event processor or processors may be configured to automatically open trouble tickets for all events requiring action. If the help desk is responsible for dispatching support personnel for problems, then the events need to be presented to the consoles they use.

When planning an event management process, be sure that users have the technical aptitude and training to manage events with the tools provided to them. This is key to ensuring the success of the event processing implementation.

1.4.3 Policies

Organizations that have a documented event management process, as defined in 1.2, “Terminology” on page 4, may already have a set of event management policies. Those that do not should develop one to support their event correlation efforts.

Policies are the guiding principles that govern the processing of events. They may include who in the organization is responsible for resolving problems; what tools and procedures they use; how problems are escalated; where filtering, correlation, and automation occur; and how quickly problems of various severities must be resolved.

When developing policies, the rationale behind them and the implications of implementing them should be clearly understood, documented, and distributed to affected parties within the organization. This ensures consistency in the implementation and use of the event management process.

Table 1-1 shows an example of a policy, its rationale, and implication.

Table 1-1 Sample policy

Policy	Rationale	Implication
Filtering takes place as early as possible in the event life cycle. The optimal location is at the event source.	This minimizes the effect of events in the network, reduces the processing required at the event processors, and prevents clutter on the operator consoles.	Filtered events must be logged at the source to provide necessary audit trails.

It is expected that the policies need to be periodically updated as organizations change and grow, incorporating new technologies into their environments. Who is responsible for maintaining the policies and the procedure they should follow should also be a documented policy.

1.4.4 Standards

Standards are vital to every IT organization because they ensure consistency. There are many types of standards that can be defined. System and user names,

IP addressing, workstation images, allowable software, system backup and maintenance, procurement, and security are a few examples.


Understanding these standards and how they affect event management is important in the successful design and implementation of the systems management infrastructure. For example, if a security standard states that only employees of the company can administer passwords and the help desk is outsourced, procedures should not be implemented to allow the help desk personnel to respond to password expired events.

For the purposes of event correlation and automation, one of the most important standards to consider is a naming convention. Trouble ticketing and notification actions need to specify the support people to inform for problems with system resources. If a meaningful naming convention is in place, this process can be easily automated. Positional characters within a resource name, for example, may be used to determine the resource's location, and therefore, the support staff that supports that location.

Likewise, automated actions rely on naming conventions for ease of implementation. They can use characters within a name to determine resource type, which may affect the type of automation performed on the resource. If naming conventions are not used, more elaborate coding may be required to automate the event handling processes.

Generally, the event management policies should include reference to any IT standards that directly affect the management of events. This information should also be documented in the event management policies.





## Event management categories and best practices

Event management issues need to be addressed when an organization begins monitoring an IT environment for the first time, decides to implement a new set of systems management tools, or wants to rectify problems with its current implementation. Often it is the tool implementers who decide the approach to use in handling events. Where multiple tools are implemented by different administrators, inconsistent policies and procedures arise.

The purpose of this chapter is to provide best practices for both the general implementation approach an organization uses to monitor its environment and the specific event management concepts defined in Chapter 1, “Introduction to event management” on page 1. Product-specific best practices are addressed in Chapter 6, “Event management products and best practices” on page 173.

This chapter is a compilation of best practices in regards to event management and the rationale behind them. When reading this chapter, keep in mind the following rules:

- ▶ Place more weight on the reasons that affect organizations when determining which best practices to implement.
- ▶ Use of business impact software may affect which best practices to implement.

Specifically, do not blindly start using a best practice from this chapter. Keep the context of the best practice in mind and apply it to your specific needs. Be sure to weigh the rationales given for these best practices.

## **2.1 Implementation approaches**

There are many approaches for implementing an event management design. Those that are used generally arise from the knowledge, experience, and creativity of the systems management product implementers. Each approach has its advantages and disadvantages which organizations must weigh when deciding how to proceed.

Regardless of the approach selected, effective communication within the organization, clear direction from management, a culture that does not assign blame for or finger-point over missed problems, and clear event management policies are critical to its success (see 2.2, “Policies and standards” on page 32).

Five of the more common approaches, and their pros and cons are discussed in the following sections.

### **2.1.1 Send all possible events**

When implementing tools, sending events is enabled for all monitors. All error messages are extracted from logs of interest and forwarded through the event management hierarchy.

This approach, while generally easy to implement, has many drawbacks. The event management consoles, trouble-ticketing systems, or both become cluttered with messages that may not represent actual problems. Operators and support staffs are left with the challenge of sifting through large amounts of information to determine which events or trouble tickets represent real problems and require action. Bandwidth and processing cycles are wasted in handling the extraneous events.

Of all the methods deployed, this one usually results in the most frustration to the tool users. These users often ignore events, close large numbers of them at once, or cease using the tools to deal with the large event volumes.

### **2.1.2 Start with out-of-the-box notifications and analyze reiteratively**

The default settings provided with the systems management tools are used as a starting point to determine which events to handle. Received events are analyzed periodically to determine whether they are truly relevant to an organization.

This strategy relies on the belief that the vendor has chosen to report meaningful information and that the values chosen result in a reasonable event volume. If the tool vendor has done a good job in selecting defaults, this approach can be a good one. If not, the tool users may still have the challenge of finding the real problems among many extraneous events.

Another downside of choosing tool defaults as the starting point is that it does not consider conditions or applications that are unique to the organization. For example, if a homegrown application is used, no vendor-supplied tool will supply defaults as to which application errors to report to the event management processors.

### **2.1.3 Report only known problems and add them to the list as they are identified**

In this method, only events that indicate real problems are reported. When new problems occur, the technicians responsible for resolving them determine whether a log message can be extracted or a monitor deployed to check for the error condition. The new events are implemented. When the problem reoccurs, the events received are analyzed to ensure that they successfully reported the condition in a timely manner.

This approach is sometimes used when an organization enlists an application or systems support department to pilot use of the systems management tools. Presenting the support people with real problems that they know how to handle makes their jobs easier, helps reduce the mean-time to repair, and hopefully makes them advocates of the tools to other support departments.

Support personnel like this approach. The events they receive are those that actually require action. This eliminates the situation where administrators spend time researching events only to find they do not require any corrective action. If this occurs too often, the support personnel eventually ignore the events presented to them.

Adding small numbers of new events at a time minimizes the possibility of event floods, and therefore, problem tickets or service dispatches. The events report conditions that the administrator has already resolved, so the resolution to the problems are usually known and easily handled. Finally, since those informed of the problems are already responsible for fixing them, they do not have the impression that the tool is giving them additional work.

The drawback of this approach is that the problem must occur and be identified as a real problem before it is reported. This relies on non-automated methods such as user phone calls to report the problem. Thus, when an error occurs for the first time, it is not automatically detected or reported.

### **2.1.4 Choose top X problems from each support area**

This is a variation of the previous approach. Representatives from each support area provide information about their top problems. The conditions can be situations on which they spend the most time handling, or those that, while infrequent, can have the greatest impact on the systems for which they are responsible.

The approach differs from the previous one in that the problems can be conditions that have not yet happened but are so critical or pervasive in nature that they require immediate action if they occur. Also, monitors are implemented in an attempt to prevent them from occurring at all.

Again, administrators like this approach because they control which notifications they receive. Their most time-consuming and repetitive problem determination and recovery tasks can be automated, freeing them for more interesting challenges. Finally, they can stop manually monitoring for the situations that can potentially cause the most serious outages or problems.

The downside is that the condition must be already known to the support staff before it is reported. It does not catch problem conditions of which the administrator is not yet aware.

### **2.1.5 Perform Event Management and Monitoring Design**

Using the Event Management and Monitoring Design (EMMD) methodology, all possible events from sources in the IT environment are analyzed and event processing decisions are made for them.

This approach, while most time-consuming to implement, is the most thorough. It offers the advantages of the other approaches while addressing their drawbacks. Again, support personnel are only notified of real problems and have control over the notifications they receive. Reviewing all possible events may highlight previously unidentified potential problems and monitors may be implemented to detect them. This makes this approach more proactive than the others.

Because this solution is the most encompassing, we recommend that you use this one. Organizations can employ the methodology themselves, or contract with IBM Global Services to lead the effort and use proprietary tools for the event analysis, documentation, and implementation.

## Event Management and Monitoring Design

Since events are typically obtained from network and system monitoring agents, event management and monitoring are related topics. The proper monitors must be implemented to receive meaningful events into an event management hierarchy at a rate at which they can be handled.

Therefore, another method of implementing event correlation and automation is to simultaneously analyze the monitors that are available or already implemented and the events they produce. IBM developed a proprietary methodology and patented set of tools to address both monitoring and event processing. The methodology and tools are available to clients as IBM Global Services offering *Event Management and Monitoring Design*.

### **EMMD approach**

The EMMD approach systematically analyzes monitors and events based on either the resources that comprise a business function or the types of agents that produce events. The client's environment typically determines the manner in which the EMMD is used. In an organization with a high degree of separation by business application, it makes sense to perform a *service decomposition* to catalog which system resources and monitoring agents are applicable to critical business functions, and analyze those using the methodology. The support personnel responsible for the application are also accountable for the server performance and can make decisions on the proper handling of both kinds of events.

Alternately, monitoring agent analysis may be done in organizations where support is by platform type or component rather than application. In these environments, a support group handles all the events of a certain type, regardless of the application or server that produces them. Therefore, the support group can take responsibility for all events produced by the monitoring agents they handle.

### **Methodology**

Regardless of which way you use EMMD, you must follow these steps in the methodology:

1. **Set scope:** Determine the scope of the project. Decide which services or monitoring agent *event sources* to analyze. Often, critical or high-visibility business functions are chosen when using the service decomposition approach. For component analysis, the monitoring sources that produce the most events, or those that report problems for the least stable platforms, are typically selected.
2. **Determine event handling policies:** As discussed throughout this redbook, best practices dictate that a set of guidelines exists to determine how to handle events. This provides consistency across the organization and makes

it easier to decide which action to take for a particular event. In this step, key members of the organization work together to develop these policies.

3. **Document event repertoires:** The events that can be produced by event sources are compiled into worksheets used to document decisions about the events. These lists can include all possible events from a given source, or those routinely received at the event processors. Typically, all events from a source are analyzed if there is a manageable number.

For sources that can produce a plethora of possible events, the events are usually limited in one of two ways. The limitation can be based on the event policies such as “Filter events related to configuration changes” or “Do not report information only events”. Alternately, the events to be analyzed can be limited to those that are typically generated by the source. This list is comprised of each type of event produced by the source within a representative time frame such as two to four weeks.

4. **Select monitors and thresholds:** Review existing monitors for relevance, and suggest new monitors to ensure necessary events are reported. Meaningful thresholds are set based on both best practices and on the baseline performance of the machines to be monitored.
5. **Document event handling decisions:** Appropriate subject matter experts (SMEs) decide the filtering, forwarding, notification, and automation actions for each event in the event repertoire. These are documented in the event repertoire worksheets. The captured information is reported based on the level in the event processing hierarchy at which the processing should occur. It includes such things as event severities, trouble-ticket priorities, and automation script names.
6. **Conduct event correlation analysis:** Determine which events correlate together, and assign primary, secondary, or clearing status to them. The SMEs can suggest possible correlation sequences based upon the meaning of the various events, and upon their experience in solving past problems that may have produced the events. Help Desk personnel are also invaluable in determining which events occur together since they frequently view all events and visually correlate them to determine which require trouble tickets and which should be dispatched to support staffs. In addition, you can use an IBM-patented data mining tool to determine statistically which events often occur together. This same tool can suggest possible correlation sequences. The event relationships are depicted diagrammatically using Visio.
7. **Review the deliverables:** The project deliverables include the event handling policies, completed event repertoire worksheets, and correlation diagrams. Review these to ensure that they are understood both by those responsible for handling the events and the implementers of the monitoring agents and event processors.

8. **Define an implementation plan:** Discuss ways to implement the design and develop an implementation plan. The plan includes, among other things, the order in which the event sources should be configured, the tasks required to complete the implementation, responsible parties, and testing and backout procedures.

### **Tools**

To aid in the various steps of the methodology, IBM developed and patented a set of tools. These tools serve to automate the design steps wherever possible and produce configuration information that can be used in the implementation:

- ▶ **EMMD tool:** This is a Visual Basic tool that automates the creation of event repertoire worksheets. Blank worksheets may be produced that are used by the IBM practitioner to document the events from a given source. The tool can also populate worksheets with event information from Simple Network Management Protocol (SNMP) Management Information Bases (MIBs), NetView trapd files, and IBM Tivoli Enterprise Console BAROC files. The worksheets include such information as the event name, description, filtering decisions, throttling parameters, forwarding, and automation commands or script names. They also include notification and trouble-ticketing targets and methods for each tier or event processor in the event management hierarchy.

Additionally, the tool can help to generate Visio stencils that represent each event that requires correlation. These are used in the Visio diagrams to document the event correlation sequences.

- ▶ **EMMD workbooks:** Based on Microsoft® Excel, these workbooks contain macros that assist in the documentation of the event handling decisions. The functions may include shadowing events that are filtered, propagating information between the sheets that represent the various tiers of the event management hierarchy, and generating IBM Tivoli Enterprise Console classes and rules based on predefined templates to implement the design using IBM Tivoli Enterprise Console.
- ▶ **EMMD diagrams:** The Visio diagrams depict the relationships among events, showing primary and secondary problem events and those that clear them. Using the stencils generated by the EMMD tool, the practitioner creates a multi-page diagram that shows the event sequences. The diagram includes a table of contents for easy navigation among the pages. Also, macros defined in a base diagram allow for such functions as generating IBM Tivoli Enterprise Console rules to implement the correlation sequences. These rules are a starting point for correlating events and should be modified to fit into a modular IBM Tivoli Enterprise Console rulebase when implementing the EMMD design.
- ▶ **Data Driven Event Management Design (DDEMD) tool:** This data mining tool can help to process lists of commonly received events. The events to be

processed are input via ASCII files. They can come from a wide variety of sources such as Microsoft Windows® Event Logs, IBM Tivoli Enterprise Console `wtdump`1 output, and log files.

The events are parsed to determine event type and relevant information within the event. The various analysis functions of the tool, including reporting event frequency by type and host, event rates by time-of-day, and statistical correlation of events, can use the event details. There are also predictive functions within the tool that enable the practitioner to see the impact of implementing various filtering and correlation rules for the given list of events.

## 2.2 Policies and standards

Critical to the success of event management is the process of creating event processing policies and procedures and tracking compliance with them. Without this, an organization lacks consistency and accountability. When different support groups implement their own event management, the tools used are not configured to standards, making them difficult to configure and maintain. Inconsistent tool use can affect measurements, such as mean-time to repair, make accountability more difficult, or skew the problem counts that may be used to determine staffing in the various support groups.

Each event handling action—filtering, forwarding, duplicate detection, correlation, escalation, synchronization, notification, trouble ticketing, and automation—should be described in a documented policy. This makes it easier to make event processing decisions and implement systems management tools.

In this section, we discuss important policies and procedures to develop and document in addition to those that specifically describe the major event handling actions of filtering, duplicate detection, correlation, escalation, and automation. For each, we recommend that you list the policy and its implications.

Note that some implications always follow from the policy, and others depend upon your systems management toolset or organizational structure. Table 2-1 shows an example in which the implications always follow from the policy.



Table 2-1 Policy and implications

Policy	Rationale	Implications
Automatically page for all severity one events.	Improves mean-time to repair.	To be meaningful, the notification should include the trouble-ticket number.
	Minimizes assigning the wrong support person to the problem.	An audit trail is required to ensure the process is working properly and to record who is assigned to the event.

If your problem management system is incapable of performing paging, you may add an implication stating that you need to develop an interface to send the trouble-ticket number to whichever event processor will trigger paging.

### 2.2.1 Reviewing the event management process

You must first keep in mind the dynamic nature of the event management process. Organizations grow and change, developing new requirements and outgrowing existing ones. While the event management process is developed to address known issues, time, organizational changes, and experience often bring others to light, requiring changes to the event management process.

These changes can be made to the event handling guidelines documented in the policies and procedures, or to the implementation of the event processors that filter, forward, correlate, automate, notify, and trouble ticket events. Periodically review these at time intervals related to the rate at which your organization changes.

#### Updating policies and procedures

Assign the responsibility for the iterative review and update of policies and procedures to one person. This individual should understand your organizational structure, be privy to information about upcoming changes, know the roles of the various support groups, and be able to work with them to gather new requirements and integrate them into the existing policies and processes.

The reviews can be scheduled to occur periodically, such as once a year. Or they can be triggered by events such as reorganizations within the company, mergers and acquisitions, changes in upper management (and hence visions and goals), outsourcing, or additions of lines of business or applications.

#### Modifying systems management tool implementations

There are two types of tool modifications. The first relates to how the tools address event management policies and procedures. The second relates to how they handle specific events.

Updates to the event management policies and processes often imply changes to the tools that implement them. Hence, these types of changes frequently coincide. Other factors that affect this type of change are implementation of new systems management tools or upgrades to existing ones. Typically, the new or upgraded software is capable of providing new function or more effectively implementing existing ones. Sometimes this means that the tool must be implemented or reconfigured to enforce the policies and procedures previously defined. Other times, the improved tools provide a better means of addressing event management, implying changes to the underlying policies and processes.

Changes to how the tools process individual events are affected by several factors and should be iterative to account for them. These changes are more frequent than the changes that address policies and processes and should be implemented more often:

- ▶ New event sources

As new hardware and software are added to the environment, new events may be generated, or there may be changes to the use of existing ones. Review the new events through a methodology such as IBM EMMD, document the decisions for handling the events, and implement the changes.

- ▶ Problem post mortems

These are ideal situations to help identify the causes and resolutions for problems. Use these sessions constructively to identify ways to monitor for the future failures, new events to forward, and correlation sequences, and implement them.

- ▶ Experience of operators/staff using the events and their correlations

Those who monitor the consoles for events often have a good sense of which events flood, occur together, or can be ignored safely. Use their ongoing input to tweak your event management implementation.

Often event processing decisions are initially made based on information in messages manuals or on the educated guesses of SMEs. When an error condition occurs, it may behave differently than anticipated. For example, the message manual states the meaning of an error message that indicates an important failure for which notification is desired. However, it fails to mention that the subsystem will retry the failing process five times per second and generate the error message each time it fails. Those watching the console detect this and can provide the feedback necessary to suppress the duplicate messages or find another means of detecting the condition.

## 2.2.2 Defining severities

The severity assigned to an event is an indication of the how critical a problem is that it reports, which relates, in turn, to how quickly service must be restored to

the failing system or resource. Event processors may use different terminology, but most provide a range of severities that can be used to designate varying degrees of criticality, ranging between “immediate attention required” and “this can wait”.

Severity levels may already be defined as part of documented problem or change management processes or service level agreements. In this case, use the existing definitions. Otherwise, define severities and acceptable recovery times as part of the event management process. For each severity, choose notification methods appropriate for events of those critical levels.

Consider mapping severities to business impact. For example, define *fatal* to imply a key business process is down and affects many users. Designate *warning* to mean problems with non-critical resources or those that affect few users.

This is important for two major reasons. First, it is easier to decide the severity to assign to an event. When implementing event management, personnel from the support organization may be called upon to review events and identify those for which they desire notification. Part of this process is to assign severities to the events and choose a notification type. Knowing the meanings of the various severities simplifies this process.

For example, when the server that runs a key business application fails, it must be handled immediately. If a *fatal* severity is defined to mean “respond immediately” and “inform by paging”, the administrator can easily decide this is the proper severity to assign to events reporting outages involving that server.

Second, clear definitions facilitate tool setup and maintenance. When the same types of notifications are performed on all events of the same severity, the event processing tool can be configured to merely check an event’s severity and take the appropriate action. If there is no consistency, the event processor must check additional parts of the event before making its determination, which complicates the implementation.

When event processors use different severities, define mappings to show how they relate. This is necessary to ensure events forwarded from one event processor to another are assigned the proper severity at the receiver. Show how the trouble-ticket severities map between all event processors in the hierarchy, from the event sources and monitoring agents, through the correlation engines, to the trouble-ticketing system. See Chapter 6, “Event management products and best practices” on page 173, for a sample severity mapping.

### 2.2.3 Implementing consistent standards

One of the primary goals of the event management process is to automate the filtering, duplicate detection, notification, correlation, and escalation of events and the recovery of systems. Automation of any kind is easier when the resources that are handled adhere to standards.

Organizations that already adhere to standards can more easily implement event management than those that have not yet defined or implemented them. If your organization is in the process of developing standards while implementing event management at the same time, you may have to initially set up two types of processing: one for machines that follow the convention, and one for machines that do not follow the convention. This allows you to proceed with event management without waiting until your environment is completely converted to the new standards.

Use standards in your event management process, and keep them consistent across the enterprise. There are always special cases which arise, but these should be kept to a minimum. This minimizes the processing cycles that are required and simplify configuring and maintaining tools.

The following sections document two examples of how commonly implemented standards can facilitate event management.

#### **Naming conventions**

Naming conventions is one of the most important standards to have. Machines, scripts, files, domains, and the like should all have consistent names across an organization because they are commonly used or referenced in automation.

Depending upon your support structure, you may need to notify, page, or route a trouble ticket to people based upon the geographic location of the failing machine or by its device type. Having a standard naming convention simplifies this process.

If the standard is to use a geographic location or device type in either the domain names or in positional characters within the host name, the event processor can easily determine whom to notify. When a failure occurs to RDUWWS01, for example, automation can determine from the first three characters that the device is in Raleigh and from the next two that it is a Windows Web server, and notify based on this information.

It is much easier to determine the proper support group for a problem based on geography or device type than by individual host. If the trouble ticket queue names or e-mail user IDs follow a convention similar to the machine names, it may be possible to create the notification target from values extracted from the host name, avoiding the need to maintain the information separately in

spreadsheets or files for individual hosts. This is advantageous, particularly in large organizations with many systems located across many sites.

### **System configurations**

Automation can be consistently applied to machines that have the same configuration. For example, when a performance problem occurs on a machine, it may be desirable to gather diagnostic data from the time that degradation occurs. Otherwise, the condition may clear and the information disappear, leaving the assigned support person unable to research the problem.

The tools needed to gather the diagnostics may be installed on some systems but not others. Therefore, the automation can only be applied to a subset of the environment. The systems management tool implementer must then determine which systems have the required diagnostic tools installed, and implement two types of event handling (one with diagnostics and one without) for the same event.

Standard machine configurations eliminates this problem by ensuring that predefined groups of systems are all configured the same.

## **2.2.4 Assigning responsibilities**

Potentially many people may be involved in the event management process, including support staffs, tool implementers, help desk and operations center personnel, and managers. All have opinions about how to handle events based their on experience in their jobs.

Definitely use their input when developing the event management process. Including their ideas helps to ensure a robust solution that meets the needs of its users. People who feel that they have influence are also more likely to embrace the resulting processes, facilitating their enforcement.

However, assign specific responsibilities to the involved parties and designate a final arbiter who will decide issues in dispute. Otherwise, the development process may stall as the involved people seek consensus (which is often difficult to obtain, especially in large groups).

Some of the roles to assign include:

- ▶ **Process owner:** Heads the development of the policies referenced in this section. Has ultimate responsibility for the success of the event management process.
- ▶ **Systems management architect:** Designs the technical solution to meet the event processing requirements while adhering to appropriate policies and standards.

- ▶ **Tool implementers:** Install, configure, and support the systems management tools to process events to the specifications of the architect's design.
- ▶ **Subject matter experts:** Supply knowledge about a particular platform or system and determine the processing required for events within their areas of expertise.
- ▶ **Support staff:** Handles problems with platforms and systems.
- ▶ **Help desk:** Provides the first level of support for users and give feedback that is used by the SMEs to make event processing decisions.
- ▶ **Managers:** Enforce adherence to policy by their staffs and ensure problems are addressed in a timely manner by responding to escalation procedures.

Consider that one person may fill multiple roles. For example, the SMEs are usually part of the support staff that resolves problems for their areas of expertise.

Assign people to the roles that defined in the previous list. This ensures that the appropriate person is handling a given task, eliminates duplication of effort, and provides accountability within your organization.

## 2.2.5 Enforcing policies

Given the importance of both the automated event handling policies and those defined in this section, it is crucial to see they are followed. Therefore, define, implement, and track compliance with policies. This ensures consistency of design and ease of tool implementation and maintenance, resulting in a successful event management endeavor.

The ramifications of not following policies and procedures vary with the policy itself. Data validity, for example, may be adversely affected by not following the policy requiring operators and administrators to close problems when they are resolved. Only closed events are recorded in the Tivoli Enterprise Data Warehouse database. Leaving problems in an open state can prevent them from being recorded and reported within the warehouse, leading to incomplete or misleading service-level reports.

One implication of enforcing policy is the necessity of a method of tracking adherence to it. Record who takes which actions on an event. This lets you know who to contact for the status of open problems and provides a means of determining who invoked wrong actions on an event so you can ensure it does not recur.

## 2.3 Filtering

Filtering is, without question, an essential part of work in each event management effort. This section discusses filtering, which in IT terms is described as the process of blocking information based on a defined set of rules and standards. In general, we define filtering as the part of the whole engagement, where we try to remove as much redundant and unnecessary data as possible.

The most difficult part is to determine what the right data is that we need to effectively implement an event management which signals possible alert situations. The following sections discuss the aspects of filtering in a systems management environment. They also discuss best practices for the filtering task itself. They do not cover device specific filtering methods and best practices.

### 2.3.1 Why filter

Obviously there is a need to restrict the data entering our event management system, or filter them somewhere on their path toward the event management system. Otherwise, we would not dedicate a whole section to this topic.

There are several reasons why filtering of events is most recommended:

- The pure amount of data produced by managed objects

In a complex IT environment, the amount of events produced by managed objects can reach a high amount of discrete events being sent to a single management instance. Many devices provide, by default, all events they are available to offer, which, for some of those management agents can easily be a list of several hundred events. If this is multiplied by the number of different managed devices in a managed environment, we see that these amount of possible events cannot seriously be managed.

- Redundant information produced by various monitoring agents inside a single managed object

Often, various monitoring instances on a device provide the same information and send them in the form of events. For example, the syslog subsystem on a UNIX server provides critical information, while the SNMP agent running on that server provides trap information about the same event.

- Network and bandwidth considerations, event server limitations

In a large and complex distributed environment the event-related traffic is basically unwanted waste of resources. This applies both to the traffic produced from status polling of devices and the traffic generated by devices sending asynchronous unsolicited events over the network. Event-related traffic can occupy a reasonable amount of bandwidth. In most environments,

network bandwidth is still a precious resource and is normally reserved for productive matters. An increased system management traffic can be treated itself as an degrading event.

Also, the receiving instance, whether a simple event console or a more sophisticated business management system, cannot accept an unlimited number of events per time frame. Often, the receiving management system itself polls managed objects in regular intervals to monitor critical resources on that object. If a threshold is exceeded, this information is translated into an event and enters the event stream. Obviously, if management stations have a limited capability to receive events, they have a limitation on the amount of polling operations per time frame.

You can find a discussion about specific products and their capabilities in Chapter 6, “Event management products and best practices” on page 173.

► Manageability

As a first rule, keep the event management system as simple as possible. Too many events from a large number of different sources can lead to confusion. An operator can soon start to ignore incoming events if they alarm for duplicate events or, even worse, lead to wrong alarms and actions.

All of these points are reason enough to limit the amount of data arriving in the event management system. Together they make the need for filtering essential.

## 2.3.2 How to filter

The main question we must ask is: “Which events do you need to do your job?” Or better, we should ask: “Because most IT organizations today are service units for the various other departments, which events do you need to fulfill the agreements you made with your customers?”

In general, if a piece of information arrives in our management system and does not indicate a loss or a degradation of services, it should not appear and should be blocked. If it does not affect your service level agreements, remove it.

Keep in mind, that a particular event, in which you are *not* interested, may be of some importance to other departments. Therefore, preventing the event from being generated may not be the best idea.

Suppressing the delivery of an event, without making it completely unavailable to other recipients, makes the simple term filtering more difficult. Everyone may agree on filtering itself, but where the actual filter is applied can be vary from one viewpoint to the other.



### 2.3.3 Where to filter

Now we must ask: “Where do you need to filter unwanted events?” If we remember the discussion in “Why filter” on page 39, we considered the occupied network bandwidth as a good reason for filtering. The possible large amount of events was another reason.

This can only lead to one rule: *Filter as close to the source as possible*. Filtering as close to the source is, in most cases, the best method to block an event from being delivered to the rest of the world. It saves bandwidth, helps to keep event management systems manageable, and saves system resources needed for production.

Filtering as close as possible, preferably directly at the source, should be the first choice. But sometimes, you cannot achieve this goal for the following reasons:

- ▶ The event may be of some importance to other management instances. For example, network operations may be interested in a toggling integrated services digital network (ISDN) interface. The organization running the corporate wide event console is, in most cases, not interested as long as the network performance is not degraded.
- ▶ The event cannot be filtered at the source, because the event generator itself is an all-or-nothing implementation. Either you buy all the events or you block all of the events.
- ▶ Events generated as a result of a status poll operation are normally not of a particular importance on the focal point level of the management system, in case it is an event console implementation such as the IBM Tivoli Enterprise Console. The status information is definitely needed for the actual status poller to maintain a list of the current states of the managed objects. Status information is also required if the focal point for the event management is a system dedicated to business impact management. Business impact management systems keep a record about the actual state of its managed object to monitor complete business environments.
- ▶ Trying to filter the event at the source can result in an effort which is more costly than just trying to catch the event on a higher level of event management. For example, after a rollout of a high number of devices, it turns out all the devices are, by default, configured to a *positive forward all* state. Remotely accessing these devices and blocking the unwanted event one-by-one at the source can be time consuming.

### 2.3.4 What to filter

Now we need to specify what to filter. This is by far the most time consuming task related to filtering. Under normal circumstances, the various resources to be

managed are well known. But regardless of whether these resources are capable of providing valuable information to a management system in form of events is not necessarily known.

After the set of event sources is specified, we need to address all events of each event source and analyze them for their value to the event management. Of course, we do not limit the analysis of the events to filter. Decisions, correlation candidates, and throttling parameters may be discussed and documented during this stage.

Speaking of best practices, two suggested approaches exist to make filter decisions and the correct correlation and escalation decisions. Refer to 2.5, “Correlation” on page 51, and 2.7, “Escalation” on page 60, for more information about correlation and escalation.

The first approach applies to environments, where little or no event processing takes place. It may also apply to environments where events are generated, but are treated as unwanted noise until a working systems management environment is set up. In this case, you must complete these tasks:

1. Identify and define the event sources who are important to the management organization. Often it helps if element chains are documented and there is a given management view in place.
2. For each event source, build a list of all events offered by the event source.
3. Find an expert for the resource being analyzed and discuss the importance (or lack of importance) of each event.
4. Find an expert who is responsible for the management of that particular resource. Often this is the same person who knows the events that are needed. Discuss whether the events should be included in the event management.
5. Document these decisions.

The approach is somewhat static because it defines a set of event sources to be analyzed and the results being implemented. If no iterative process is setup after the initial work, the result is quickly outdated.

Define a process that, after the initial analysis, detects changes in the event flow or additions and deletions in the set of event sources. It should also ensure that the event management process is iterative.

This approach can be called *filter by SMEs*. The analysis and the resulting filter decisions depend on the expertise of the people who are responsible for a given resource.

Another approach to obtain fundamental information about the events appearing in a given environment is to analyze the events itself using a data mining approach:

1. Obtain information about all events received in your organization over a time frame of at least three months to have a solid base for analysis.
2. Normalize the data by extracting only the relevant information, such as:
  - Time stamp
  - Event type
  - Event name

Try to limit the event information to a minimum. It is sufficient if the event can be uniquely identified.

3. With a small part of the whole data repository, typically the events of a two-week time frame, run an initial analysis. Make sure that the data contains the information you expect.
  - Use the whole data and analyze it.
  - Are there any large amounts of a single event?
  - Is there another event from the same source having the same or a similar count?

Such a pattern often signals a violation event and its corresponding clearing event.

- Are there groups of events from different event sources appearing with similar counts?

This can be a initial problem causing other, secondary exceptions to occur. What makes them correlation candidates?

- Are there more than two different events from one event source appearing with the same or a similar count?

This can be a primary or secondary condition, too. For example, an interface down SNMP trap sent by a network device is often followed by an interface down event produced by the SNMP network manager, generated by a status poll operation against this interface. An unsuccessful poll for status against an interface can result in a node down event being generated if the interface was the object's only interface.

This type of group of events is a good filter candidate. You really need only one indication to signal a problem. The same applies to the associated clearing events. You often find an interface up trap, an interface up event, and a node up event.

4. Define filtering. After you run such an event analysis, you still need SMEs for the particular event source to finally define the filter conditions. Having solid

data about event occurrence and the amount of events for a particular situation helps to keep the discussion short.

One last way to perform filter analysis is through *problem post mortems*. Analysis of situations where a service degradation occurred and nobody was informed may help to revise or find some filter decisions that were not made before.

Regardless of the method used to determine events to filter, filtering is never implemented perfectly on the first try. You must continuously evaluate and redefine your filtering methodology for filtering to be most effective. As business needs and requirements change, you must also update your filtering methodology.

### 2.3.5 Filtering best practices

Up to now, we discussed the need to filter and different methods to eliminate unwanted events. There are some best practices for which events *not* to filter and which events should never find their way into the management system.

Here are some best practices to implement for filtering:

- ▶ Do not filter or block events that have an exactly defined meaning, where an automatic action can be issued. Nor should you filter the corresponding clearing event.
- ▶ Do not filter clearing events for problem events you report. Administrators do not know when an event has been resolved if they do not receive the clearing event. Also, during correlation, a problem event may not be closed if a clearing event is not received. Remember that clearing events are essential for de-escalation purposes.
- ▶ Report any exception from the normal process only once. For example, we mentioned the interface down trap, which causes an interface down and a node down event. Only one event should be passed to the management system. If possible, the primary event should be passed.

There is an exception to this rule. Sometimes it is useful to take the double events to verify the situation. A node down may result from timing or network problems. The interface down trap always signals a real exception. When the interface down trap does not find its way into the management system, the interface down and node down events are the only indications of a failing interface.

- ▶ When using business impact software, do not filter status change events. This renders the business impact software useless for providing status of objects.
- ▶ Always pass actionable events and their clearing events. An *actionable event* must be acted upon by either automation or administrator intervention.

- ▶ Do not *double monitor* a resource. Having multiple monitors check for a single condition causes processing overhead and produces redundant events. Only one problem should be reported to prevent more than one support person from handling the same issue.

A possible exception to this rule is when multiple agents are needed to validate that a problem is real and not a false positive. In this case, it is acceptable to double monitor the resource as long as the events produced by each monitor are correlated and only one reported.

- ▶ Filter false positives if possible to avoid unwanted and unneeded alerts and events. If you page someone at 3 a.m., you better be sure it's a real problem.

## 2.4 Duplicate detection and suppression

*Duplicate event detection* is the process of determining which events represent the same instance of the same problem, and summarizing or suppressing them in a way that simplifies the event management process. Its purpose is to save cycles and system resources on event processors, and minimize bandwidth used to send unnecessary events.

### 2.4.1 Suppressing duplicate events

Duplicate detection and suppression can be done in more than one way. Depending on the case, the best practice can vary. Here are some common methods to perform duplicate detection (sometimes referred to as *de-duplication*):

- ▶ Send the first event and suppress the others. This approach is typically used with events that state a failure in a system or equipment. Immediately reporting the event minimizes the mean-time to repair.
- ▶ Send an event only after a predefined number is received. This practice, commonly referred to as *throttling*, is often used for events that represent peak conditions.

While some monitored variables, such as processor utilization, occasionally reach peaks, this is not a problem unless sustained for a period of time. For these types of events, do not send the first event. Summarize it with the subsequent events and send one event if they reach a threshold. After the event is sent, drop future occurrences until the time period expires or the condition clears. For example, if more than five peak events are received in 10 minutes, there may be a problem that requires notification. Count the events and send the fifth along with a count of the times it occurred.

- Send an event only after a predefined number is received and send all future occurrences. While similar to the previous method, this differs in that all events are forwarded when the threshold is reached.

This approach may be used when it is necessary to know the actual values the variable reached. For these events, do not send the first event. Summarize it with the subsequent events and send one event if they reach a threshold. For example, if more than five peak events are received in 10 minutes, it may represent a problem and you need to be notified. When subsequent events are received, extract the relevant values from them and update the reported event.

This method is not generally used because it requires sending and processing all events generated after the predefined number. In general, if all the monitored values are required for problem determination or trending, this information should be provided by another type of tool and not by events.

## 2.4.2 Implications of duplicate detection and suppression

Duplicate detection and suppression, when well done, are useful and can give fast results for the event management process. However, in some cases, it is important to ensure that you do *not* miss information, as illustrated in the following examples.

### Time window considerations

The first two examples illustrate how time windows affect duplicate detection and suppression for peak events. In Figure 2-1, a duplicate detection and suppression process was created for a processor utilization variable. In this case, when one peak event arrives, it is buffered and a time window is created. If four more events occur inside the time window, one event is sent to the event management system. Note that no event is sent because only three events occurred during the time window.

The problem in this case is that the last five events occurred over a time period that is shorter than the defined time window, but no event is sent. This is because one time window was opened upon receipt of the first event and no others were started until the first one expired.

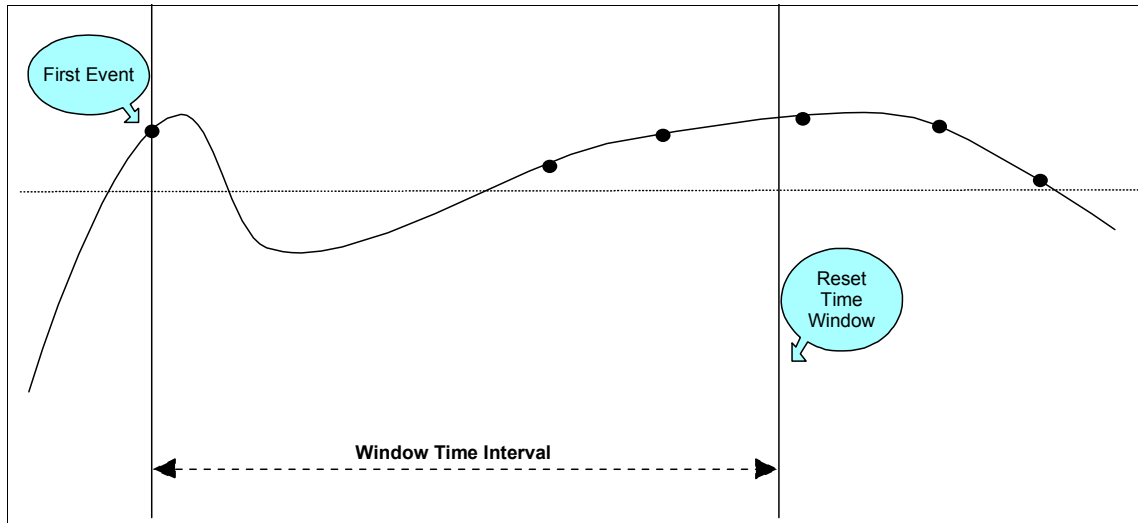


Figure 2-1 Static time window for peak events

Figure 2-2 shows the same situation, but with another duplicate detection and suppression process. For this example, every time a new event arrives, a new time window is created. During the first time window, three events occur, the same as in the last scenario. However, during the second window, five events occur and one is sent.

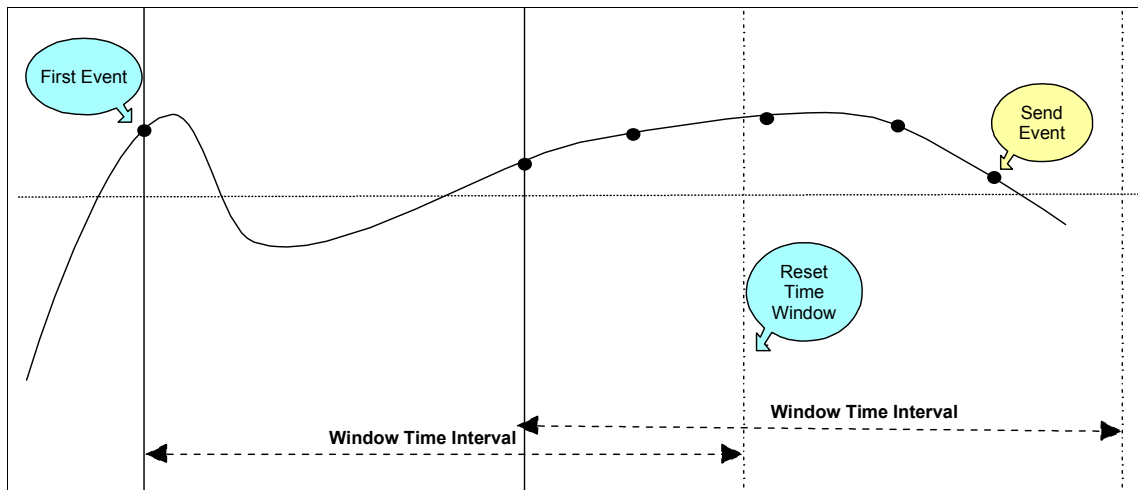


Figure 2-2 Multiple time windows for peak event

There are appropriate times to use each of these methods to create time windows:

- ▶ Use static time windows for situations that generate many (more than the threshold) consecutive events in a short time. It is also appropriate when you have severe performance constraints, since it does not create many time windows, reducing overhead in the event processors.
- ▶ Use multiple time windows when the performance constraints are not so rigid, fewer events arrive during the time windows, and the trigger number of occurrences must always send an event.

Obviously, the methods employed depend upon the capabilities of the event processing tool used.

### **Effect of clearing events on failure reporting**

Make sure that resolved problems are closed or that duplicate detection suppresses new occurrences of same problem, considering an outstanding problem already reported. The next two examples discuss the effect of clearing events on failure reporting.

Based on 2.4.1, “Suppressing duplicate events” on page 45, when a failure event arrives, the first event reporting should be sent to the event management system. Duplicate events should be suppressed during the time window. In the previous example, if a clearing event occurs within the time window and is not used to close the reported problem, subsequent failures within the time window are treated as duplicates and not reported. Figure 2-3 shows this example.

For short time windows, operators viewing the events may notice the clearing event and manually close the original problem, which resets the window. However, this method is unreliable. The clearing event may become lost within large volumes of events, particularly for long event windows. The originally reported problem stays open, and subsequent failures are not reported.



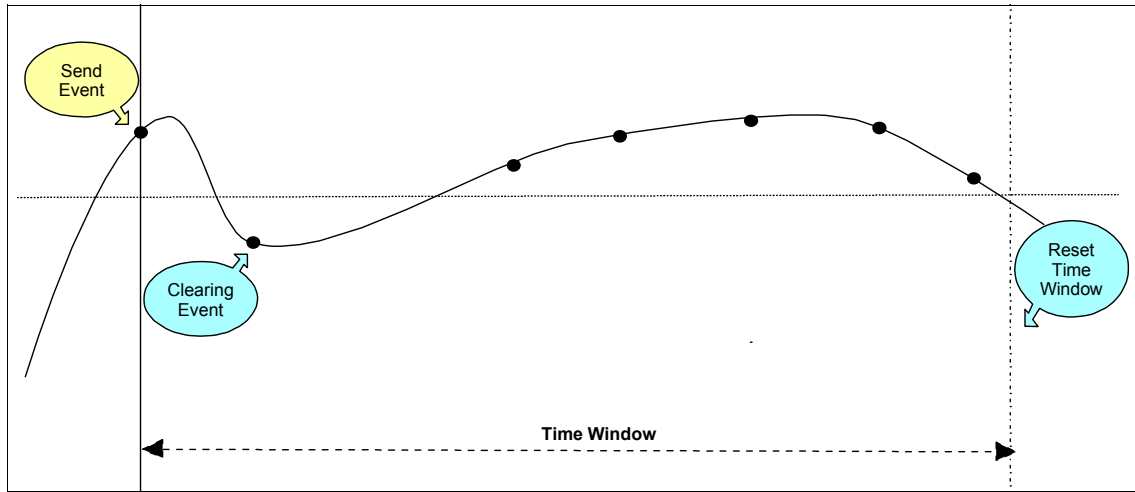


Figure 2-3 Clearing event does not reset the time window

The example in Figure 2-4 illustrates how resetting the time window and opening a new one when the problem next occurs will resolve this problem.

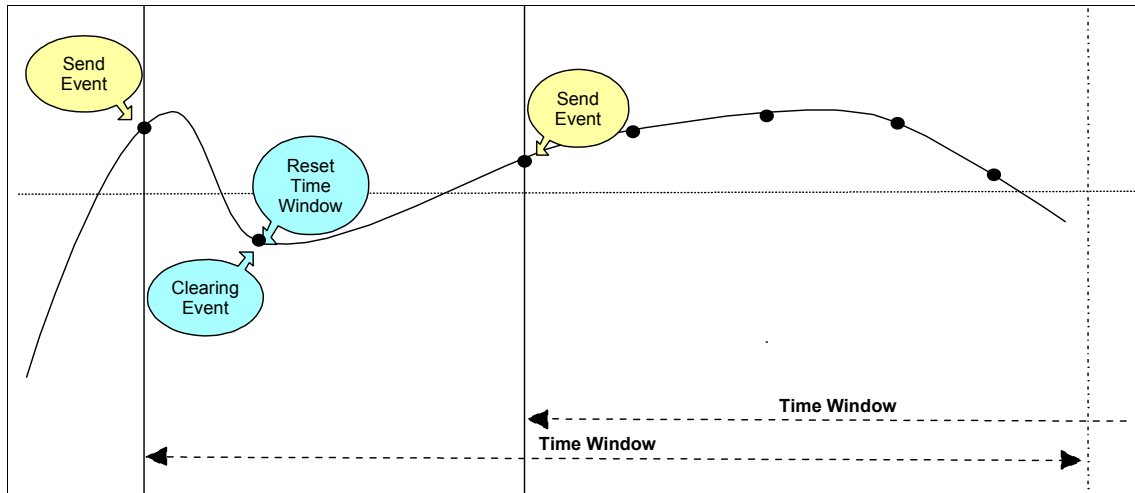


Figure 2-4 Clearing event resets time window

A good practice is to correlate the failure event with its clearing event and close the problem. This results in resetting the time window, clearing the duplicate counter, and ensuring new occurrences of the problem are reported.

### 2.4.3 Duplicate detection and throttling best practices

We make several recommendations for duplicate detection and throttling. We also provide the rationale behind these recommendations:

- ▶ Perform duplicate detection as close to the source as possible.

This practice saves cycles and system resources on event processors, and minimizes bandwidth used for sending unnecessary events.

When possible, configure the event source to detect duplicates and suppress them. If it is incapable of performing these actions or if implementing at the source causes undue, cumbersome tool configurations, use the closest event processor capable of performing the function.

- ▶ Use throttling for intermittent problems that may clear themselves automatically and do not always require action. After a problem is reported, suppress or drop duplicates.

It is frustrating for a support person to investigate a problem only to find that the problem has disappeared or requires no action. If this occurs too often, the support person loses faith in the systems management tools and begins to ignore its notifications.

- ▶ For events that indicate problems always requiring action, inform when the first event is received, and suppress or drop duplicates.

This notifies the support person most quickly and minimizes the mean-time to repair for the problem.

- ▶ Do not use duplicate events to re-inform whether the original problem has been handled. Instead, use escalation.

Using duplicate events as reminders that a problem is still open is a bad practice. The extra events clutter consoles, possibly forcing the operator to sift through many events to find the meaningful ones. If there are too many events, the console user may begin to ignore them or close them in mass. See “Unhandled events” on page 60 for a discussion about using escalation for events that have not been handled in a timely manner.

This bad practice typically arises in organizations that point fingers for unhandled problems and assign blame. Those who are responsible for resolving problems often need to justify why they miss problems and blame the tool for not informing them. The event management implementers, fearing these reproaches, configure the tools to send all occurrences of a problem rather than just one. Unfortunately, this compounds the problem because now the support person has to handle many more events and can still miss ones that require action.

Management needs to create an environment on which problem post mortems are used constructively, minimizing blame and enabling

administrators to pursue and fix the causes of missed events rather than creating event floods to avoid blame.

- Use duplicate detection for open and acknowledged events.

Duplicate detection is often implemented for open events but not acknowledged ones. These are equally as important because they indicate that the problem is being addressed but is not yet corrected. Since someone is notified of the problem, there is no reason to re-inform with subsequent events.

## 2.5 Correlation

Several sources can help to determine the relationships among events. SMEs within the organization can provide input about events in their areas of expertise. Vendors can also furnish information about the events produced by their software. Sometimes the messages manual for a product supplies event groupings by stating that a message is one of a series and listing the other messages that appear with it.

Data mining tools that employ statistical analysis, such as the proprietary software used in the IBM Data Driven Event Management Design (DDEMD) services offering, can suggest possible correlation sequences. Problem post mortems can help to determine events that frequently occur together or to validate proposed event sequences. Which sources of information an organization uses depends upon the skill level of its SMEs, the willingness or ability of its vendors to share information, and its access to statistical tools.

Often the event management implementer is left with the challenge of making sense of the information gathered and taking a best guess as to which of these potential correlation sequences to implement. Overzealousness may lead to correlating events that do not necessarily associate and dropping real problems as a result. An overly cautious approach may require more operator intervention to manually close or associate events. The best practices that follow are general guidelines to use in determining which sequences to implement.

### 2.5.1 Correlation best practices

The first step in determining what to correlate is to collect information from the sources identified previously. After the potential sequences are identified, apply these guidelines to choose which one to implement:

- Only correlate events whose relationship is clearly understood.

Correlation should be implemented only when the association between the events is known. Implementing a *best guess* can result in correlation

sequences that inadvertently drop real problems because they are erroneously thought to be symptom events. It is better to have an operator manually handle an event several times until its relationship to other events is known than to implement automatic associations that may not be valid.

As discussed in 2.2, “Policies and standards” on page 32, the event management process should be iterative and the event handling decisions should be updated as new information becomes available. Start with the sequences you know, and add to them based on the experience of your operations and support staffs.

- Automatically clear problem events when possible.

Implement this type of correlation sequence whenever feasible. This ensures that the accuracy of the list of open events from which the operations and support staffs work. It also prevents duplicate detection from flagging new occurrences of a problem as duplicates of the resolved problem.

It is usually easy to identify the clearing events associated with problem events. Monitoring agents can often be configured to generate them. The product documentation frequently describes the association between the problem and clearing events. Messages extracted from logs can be more difficult to clear. Sometimes only the problem is recorded and the recovery condition is not. In these cases, a policy is needed to ensure that operators manually close problems when they are resolved.

Implementing clearing event sequences is also easy. Many monitoring products supply the rules necessary to do this type of correlation. Ways in which the IBM Tivoli Enterprise Console product can be configured to clear events are discussed in detail in 6.3, “Correlation” on page 218.

Remember to close the clearing events as well as the problem events. This minimizes the number of events that display on operator consoles and reduces overhead at the event processor.

- Correlate events that show a worsening condition.

Sometimes when a condition intensifies, multiple events are generated to show the progression of the problem. Correlate these types of events, and keep only the first in the series. Update it with the higher severities as the other events are received. If desired, include a threshold or other appropriate information, such as the time stamp from those events, and then drop them.

This processing ensures that the original, reported event is available for event synchronization. When the problem is closed in the trouble-ticketing system or in another event processor, this event is then correctly closed. If another event is kept, the synchronization process does not know to look for it, and the problem may remain open indefinitely.

Learn how IBM Tivoli Enterprise Console can correlate and escalate problems in 6.3, “Correlation” on page 218, and 6.5, “Escalation” on page 262.

- Report all events requiring action, not just primary events.

Part of understanding the relationship among events is knowing how action taken to resolve the problem referenced by one event affects the conditions reported by related events. This information must be obtained before determining whether a secondary event requires action.

Sometimes resolving the primary problem automatically fixes the symptom problem. In this case, no further action is required. Other times, the symptom condition does not clear automatically when the primary problem is resolved, necessitating action for the secondary event. See “Root cause correlation” on page 11 for examples of correlation sequences in which the secondary events require action and those in which they do not.

To ensure all problems are resolved, report all events requiring action, even if they are symptom events. Some implications of this are discussed in 2.5.2, “Implementation considerations” on page 54.

- Drop secondary events that do not require action, unless they are status events required by a business impact manager.

If an event does not require action, it is not needed at the event processor. To prevent clutter on the processor’s console, drop the event. Some implications of this are discussed in 2.5.2, “Implementation considerations” on page 54.

The only exception to this rule is listed in the following practice for users of business impact software.

- Forward all status events to business impact software, even those that do not require action.

Business impact managers are used to show the effects of system resource failures upon business functions. The proper functioning of business impact software relies upon it accurately reflecting the status of each of the system resources that constitute its business views.

When a resource known by the business impact manager changes status, its new state needs to be reflected in both tool’s database and business views. The effect the status change has on other resources is calculated, and the affected resources and views are modified to reflect their new states.

Therefore, if using business impact software, forward all status events to it.

## 2.5.2 Implementation considerations

While product-specific implementation is discussed in Chapter 6, “Event management products and best practices” on page 173, some general guidelines apply regardless of the tools that are used:

- ▶ If no clearing event is sent to report a resolved condition, require operators to close the appropriate event.

Leaving open problems in an event processor can lead to incorrect error reporting. Namely, the duplicate detection process may think there is an outstanding problem and discard an event that reports a new problem. Also, some data warehousing tools only load closed events into their databases. Keeping problems open prevents them from being recorded in the warehouse, and subsequently being included in trending and service level agreement reports.

To prevent this from happening, implement clearing events. However, sometimes this is not possible. For these events, the operator should manually close the event through the trouble-ticketing system or event processor’s console.

This policy should be documented in the event management process guidelines, and compliance with it should be enforced and tracked.

- ▶ Link symptom events requiring action to the problem events upon which they depend.

If a symptom event requires action to resolve the problem it reports, the action most likely cannot be executed until the primary problem is resolved. For example, a process that depends upon free space in a file system cannot be restarted until that space is available.

To show dependency between events, link them in the event processor or trouble-ticketing systems. The events may be linked by updating fields in each to show its relationship to the other. Another approach is to copy the relevant information from the symptom event into the primary and then drop the secondary.

The support person assigned to the problem can read the additional text to determine what actions may be required to handle the symptoms. Then they can either perform those action if appropriate or requeue the event or events to a different group once the primary problem is resolved.

- ▶ Repeat lower level correlations at higher levels.

Correlation may fail for a couple of reasons:

- The events may arrive too far apart. When it receives an event, the event processor often sets a timer to determine how long to wait before reporting the event as a problem. If an associated event or perhaps even its root

cause event is received after the timer expires, no correlation occurs and both events are treated as primary problems that require action.

In a multi-tiered environment, the events may be forwarded to a higher level event processor. They may arrive at the higher level closer together. Defining the same correlation sequence at that processor allows for the chance that they arrive within the correlation window. The events can possibly be correlated and the appropriate event can be reported.

- Memory constraints in the event processor may prevent the events from being correlating. If the processor relies on the presence of the events in cache to associate them, correlation fails if one of the events is dropped from cache due to memory shortages.

Again, the higher level processor may not have the same constraints and may be able to perform the correlation.

- Allow sufficient time to receive events before you correlate them.

Setting timers in an event processor is an art. Waiting too little for events can result in missing the correlation between events and reporting them all as problems, even the symptom events not requiring action. Lengthier correlation windows eliminate this problem, but may introduce others. If the timer is set too long, there may be a delay in reporting the problem, resulting in a longer mean-time to repair. Also, events that are caused by different error conditions may be erroneously correlated.

Observe the rate at which associated events are generated by sources and received by event processors to choose meaningful timer values. This information can be obtained from logs at the event sources and processors, and from problem post mortems.

- Correlate events from clusters, and escalate problems as more machines in the cluster experience them.

Clusters are groups of two or more systems typically used for load balancing or failover. If one machine in the cluster reports an event, there may not be a problem. For example, in a failover environment, only one machine in the cluster may need to run an application at a time. In this case, if the monitoring agents detect the application is not running on one server, this is a normal state and does not need to be reported. The problem exists when the application is not running on any clustered system. This concept also applies to grid computing.

In a load-balancing cluster, the situation is slightly different. If one system experiences a problem, it should be addressed. However, it is less critical than if every system in the cluster has the same problem. Use differing severities to reflect the business impact of these two distinct conditions.

Implement cross-host correlations to handle the unique event reporting requirements for clusters and to ensure the proper error conditions are detected and reported.

- Perform topology-based correlation for network events using an SNMP manager.

Since the SNMP manager knows the network topology, it is capable of performing topology-based correlation. Many SNMP managers provide the ability to correlate network events out-of-the-box.

Performing topology-based correlation between network and server events requires supplying the network layout information to the event processor at which these events converge, typically not an SNMP manager. While it is possible to provide the topology to other types of event processors, the procedure is often complex and difficult to implement. Most networks are dynamic, implying frequent updates to the topology information supplied to the processors. This quickly becomes impractical, particularly in large networks.

If the SNMP manager can detect the status of non-networking resources, such as services, it can be used to perform topology-based correlation for events concerning those resources. You can find a description of how NetView implements this type of correlation in 6.3.1, “Correlation with NetView and IBM Tivoli Switch Analyzer” on page 218.

## 2.6 Notification

Notification is a key step in the event management process. It is useless to detect an error condition unless action is taken to correct it. While automation is used increasingly to recover from problems, there are still many situations that require the intervention of an administrator to resolve. Notification is the means by which the appropriate person is informed to take action.

### 2.6.1 How to notify

This section discusses the methods of notification from your event processing tool and the advantages and drawbacks of each. Figure 2-5 shows an overview of the types of notification. Depending on the structure of your business, you will handle your notifications in different ways.



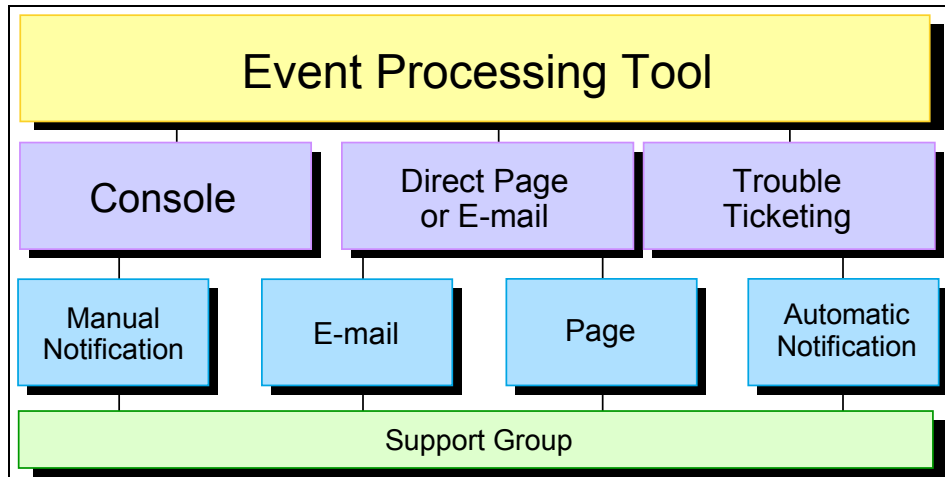


Figure 2-5 Types of notifications

With event processing tools, there are typically three ways to implement notifications:

- Console viewing by operators

Operators watch the console looking for events that require action. When they see an event, they respond by taking action themselves or manually inform another person. Having operators view the console and then manually notify support teams gives you the advantage of having someone to verify events manually when they happen and give the notification to the right person.

The disadvantages include human error, for example missing an event on the console. The disadvantages also involve the costs of keeping and training a person to watch the console.

- Direct paging and e-mailing from the event processing tool

Directly paging from the event processor, through scripts or executables that are triggered by criteria of an event, gives you the functionality of not having an operator continuously watch the console. However, it is difficult to maintain the proper lists of which groups to notify for which problems. This information, already kept in the trouble-ticketing system, needs to be duplicated in the event processor in different places such as rule bases or custom scripts. It is also difficult to track the notifications or ensure they are received.

- Integration with a trouble-ticketing system for automatic notifications

The main advantage of integrating with a trouble-ticketing system is that you tie in with tools and processes that already exist within your organization at the help desk, operations, or command center. It is much easier to track on-call lists and the right support groups for notifications. Timetables for

notifications are also easier to create and maintain within trouble-ticketing systems.

It is also easier to assign each trouble ticket to the proper groups based on information within each event. For example, if you integrate your trouble-ticketing system with your asset management system, you can automatically assign tickets to the proper support group based on the hostname slot from your event.

## 2.6.2 Notification best practices

This section discusses the best ways to design and implement problem notification:

- ▶ When possible, handle all notifications through integration with a problem tracking tool.

Many trouble-ticketing systems provide all three notification methods: a console for viewing open problems and e-mail and paging capabilities. Performing these functions from a single place simplifies system setup and maintenance. See 2.6.1, “How to notify” on page 56, for more details.

- ▶ Notify someone about all problem or actionable events

Before you start thinking about sending notifications for your event processing tool, review the events you are receiving and classify them into two separate groups:

- **Informational events:** Events that do not show a system as being down or having a problem such as clearing events
- **Problem or actionable events:** Events that indicate a system or application is unavailable or has a problem such as process or service down events

Usually it is a good idea to notify only on events that require a individual or support group to take action, specifically problem events. You do not want to notify someone about informational events, especially if it is after hours.

**Note:** While it is not a common practice to notify about clearing events, support groups may want to know when their system is back up. If this is the case, write your notification rules or scripts so that the notification comes from the original down event being closed and not the clearing event.

Next go through all of your problem events and decide which group or individual receives the notification. It is a good idea to set up a database or spreadsheet if you are not using a trouble-ticketing system.

- ▶ Consider the severity of the problem, the time of day, and critical level of the failing system when determining what notification action to take. Page for higher severity problems and critical resources, and notify others by e-mail.

After you decide which types of events require notification, go through each type and determine the severity at which you want to notify. Two automated methods are used for notification (console viewing is not automated):

- **Paging:** Sending text or numeric messages to a paging device or cell phone
- **E-mail:** Sending messages through the organization's e-mail system

When trying to determine the severity of the event, keep in mind the importance of the host from which the event is coming. Try to relate the severity of the event to the importance of the failing system.

Determining the severity for the event is directly related to the notification method chosen. When you page someone, you can reach them at all hours of the day. When you e-mail, you cannot reach someone unless they are logged in and checking their mail. Therefore, it is a good idea to have the higher severity or priority problems send a page and have the lower severity problems send an e-mail.

While you determine this, keep in mind the time of day that these notifications are sent. Although it is not suggested, some trouble-ticketing systems can send different types of notifications at different times of day such as sending an e-mail during the day and paging at night. Usually it is best to keep your notifications standard, either e-mail or page based on the problem severity. This is for easier maintainability. However, you must watch for sending unwanted or false pages, especially after hours.

- ▶ Ensure that after-hours paging does not occur for non-critical problems.

After you set up your severities and methods for notifications, double check to make sure that you are not sending notifications for non-critical problems. Also, remember that the notification process is on-going and that when a mistake or false page is sent, you must take the proper steps to ensure that it does not happen again.

- ▶ Report problems with a notification system by some other means.

Obviously, if the primary notification system is experiencing problems, it cannot be relied upon to report a problem with itself. Use another event processor or backup method to report problems with the notification system.

## 2.7 Escalation

*Escalation* is the process of increasing the severity of events to correct perceived discrepancies in their importance and to ensure problems receive appropriate and timely attention. Best practices always depend on an organization's environment and policies. It is no different for escalation.

In this section, some escalation recommendations are provided for different environments. Use those which most closely reflect your organization.

Also keep in mind hardware and performance issues when creating escalation processes. The number of escalated events needs to be well defined and controlled, ensuring that perceived discrepancies are corrected while minimizing processing overhead.

### 2.7.1 Escalation best practices

As discussed in Chapter 1, "Introduction to event management" on page 1, there are several different types of escalation. Each is important to the event management process and should have at least one policy that defines the standard way to implement it within an organization. As with all standards and policies, this facilitates tool implementation and maintenance.

This section covers the best practices for the three types of escalation.

#### **Unhandled events**

Without a clear, well-defined escalation policy for unhandled events, it is possible that problems may be reported but never resolved. An assigned support person may be too busy to work on a problem or not know how to proceed. Escalation ensures that events are handled in a timely manner by creating a greater sense of urgency through raising event severities, notifying additional people, or both.

Escalating problems is often used in help desks and network operations centers. It helps management by advising of situations in which those responsible for an event do not follow the procedures or cannot complete the job in time. With this information, managers can better coordinate the work force. For example, if an operator has too much work and cannot handle an event in time, the problem is escalated to the manager, who can allocate an idle operator to help with its resolution. See the first best practice in the list that follows this discussion.

The question becomes how long to wait to see if an event is being handled. It is possible to set different durations based on event type, host priority, time of day, and other factors. However, this quickly becomes a maintenance nightmare. An easy, effective alternative is to base it on severity. See the second and third best practices in the list that follows this discussion.

Next, decide what needs to occur within the time interval. Two typical actions are acknowledging and closing the event. The person assigned to a problem can be given a limited time, based on the severity of the problem, to acknowledge the event or close it. This informs the event management system that the problem is being addressed or is successfully resolved. If the time interval expires without event acknowledgement or closure, the problem is escalated. See the fourth best practice in the list that follows this discussion.

Regardless of whether you choose to escalate based on acknowledging or closing events or both, you must define the escalation chain or hierarchy of people to inform for unhandled problems. See the fifth best practice in the list that follows this discussion.

The best practices to consider for unhandled events are:

- Increase the severity of outstanding events after an appropriate time interval.

This is an effective way to draw attention to such events, which should result in corrective action. Also, higher severities are usually defined to mean that the problems need to be resolved more quickly. Since time has already elapsed, there truly is less time to resolve the problems to meet service level agreements.

- Set consistent time intervals for all events of the same severity.

This method means that all events of severity *warning* should be escalated if they are not handled within the same time duration. Events of another severity, such as *critical*, may be, and usually are, assigned a different interval, but that interval still applies to *all* events of that severity. The exception is the severity assigned to clearing events. Since clearing events should be closed automatically, there should never be a need to escalate them.

When severities are defined properly, they represent the urgency with which an event should be handled. They already account for service-level agreements (SLAs) and operations-level agreements (OLAs). If they are developed considering the severities discussed in 2.2.2, “Defining severities” on page 34, the severity definitions already contain information about the acceptable time to resolve problems.

Moreover, it is generally easier to implement automated escalation based on severity than on a combination of other factors. When adding a new event that require action, ensure that it has the right severity. Then little or no additional configuration or coding is required to integrate it into an existing, automated escalation process.

- ▶ Set escalation intervals to shorter than acceptable resolution times.

The severity definitions tell how quickly to handle an event. Escalate before this time interval expires to allow the informed support person to take corrective action within an acceptable time frame.

Avoid waiting to escalate until after the documented resolution time has passed. This is too late because the service level is already in violation, rendering it impossible to resolve the problem within SLA guidelines.

- ▶ Escalate when an event remains unacknowledged or unresolved.

Checking for both of these conditions is the best way to ensure that SLAs are met. Set the time interval for acknowledgement to a shorter duration than for closure. That way, if the event is unacknowledged and the problem escalated, the support person notified has enough time to work on the problem to meet the SLAs.

- ▶ When escalating an unhandled event, inform both the originally assigned support person and others that the problem now has a higher severity.

The responsible person may have accidentally forgotten about the event or may have been busy with other problems. The escalation serves as a reminder of the outstanding problem. It is also a courtesy to the administrator who may be measured on problem resolution time.

Notifying others that the event has changed increases the chance that it will be handled. If the original support person, previously unable to respond to the event, is still not in a position to pursue it, someone else can take responsibility for it.

Also, if the informed parties have authority, they can more readily ensure that the problem is handled by either reprioritizing the assigned support person's tasks or delegating the problem to someone else.

Always notify the originally assigned support person when an event is escalated, because that individual is accountable. However, do not notify everyone for each escalation. Create levels of escalation, and choose to whom to notify for each escalation. For example, a manager may not care each time an event is escalated, but needs to know if a service-level violation has occurred.

## **Business impact**

This type of escalation is based on the premise that problems with a greater business impact should be handled more quickly than others. For example, suppose two servers fail. One affects only a few internal employees, while the other prevents customers from placing orders. Business impact escalation increases the severity of the second server down event to ensure it receives priority handling.

Escalating based on the criticality of resources implies knowledge of the business impact of failures. It is necessary to understand the components that comprise a business application to use this form of escalation. Often, organizations can easily determine which server provides the front end to their business functions. They may be less likely to know the back-end servers with which that system communicates for data and other processing.

When an organization determines the systems used for business applications and their relationships, it can perform a risk assessment. This term is generally used to denote the process of assigning priorities to resources based on their value to the business. Designating a system as *high risk* implies that its failure has detrimental effects on critical business functions.

- Increase severities of key events reporting problems with the greatest business impact.

Use the risk assessment value to determine what severity to assign to an event. Assign the same severity to events that reference resources of the same risk classification. For each risk classification, choose the severity defined with the most appropriate problem resolution time.

For example, routers may be classified as core and distribution. *Core routers* handle the traffic in the network backbone and are used by most business applications. *Distribution routers* connect remote locations to the backbone and serve smaller groups of users.

Core routers may be assessed as high risk, and distribution routers may be assessed as medium risk. Suppose that critical severity was defined to mean more than one user is affected, and that fatal was defined to mean that most users are affected. The proper severity for a distribution router down event is critical. For a core router down, it is fatal. Since there are probably fewer core routers, set the severity of the router down event to critical, and escalate it to fatal if it is received for a core router.

- Perform business impact escalation for key events only.

Some events by their nature are not critical and should not be treated as such, even when reporting problems with a critical resource.

Consider again the server running the customer order application. If a server down event is received for this device, it requires immediate attention and the event severity should be adjusted to reflect this. However, if a backup power supply on the server fails, it may not need to be changed immediately. Do not perform escalation for the second event, even though it applies to the *high risk* server.

- Escalate for business impact as early in the event life cycle as possible.

Ideally, an event is generated with the severity that best reflects both the nature of the problem reported and its business impact. This minimizes

overhead in the event processors that handle it. In reality, many event sources are incapable of determining business impact or are not easily configured to do so. In these cases, an event processor must perform the business impact escalation.

Escalating as early in the event life cycle as possible minimizes the need for event synchronization later. It ensures that the event severity is accurately represented to users of intermediary event processors. Also, since the change occurs before initial notification of the problem, there is no need to renotify for the severity change.

Do not duplicate business impact escalation at multiple levels of the event management hierarchy. Otherwise, events may be escalated several times, increasing their severities higher than originally intended.

Such products as IBM Tivoli Business Systems Manager provide a means of visually determining the business impact of problems. This can be used by operators who manually escalate events from a console or by support personnel to determine the business effects of a problem they are working.

### **Worsening condition**

This form of escalation differs from those previously mentioned in that it deals with multiple events. In the escalation types previously discussed, business impact and repair time trigger changing the severity of a single event. Here the driving factor is receipt of a new event indicating a worsening condition of the original problem.

- When escalating a worsening condition, keep the first event and escalate its severity, adding information to it if necessary.

There are several reasons to keep the first rather than subsequent events. A trouble ticket may have already been opened for the first event. When the ticket is closed, event synchronization procedures attempt to close the corresponding event in other event processors. If it does not exist, the synchronization fails. The problem event that still exists in the event processor remains open, leading to subsequent occurrences of the problem being discarded by duplicate detection.

Also, keeping the first event ensures that the time at which the failure first occurred is recorded and available for problem determination.

Update the original event with desired information from the new event such as the value of a monitored variable exceeding its threshold. After updating the original event, discard the others. This reduces overhead at the event processors.

- When escalating worsening conditions, inform both the originally assigned support person and one or more others of the problem's higher severity.



The same reasons apply here as for unhandled problems since the increased severity again implies less time to resolve the underlying issue (see “Unhandled events” on page 60).

In addition, if a monitored variable reported in the event is governed by SLAs, notify those responsible for the SLAs when the reported value is about to or has caused a violation.

- Do not de-escalate for lessened conditions.

Sometimes the term *de-escalation* is used to denote lowering the severity of an event. The new severity can indicate a lessened or resolved problem.

De-escalate events only when they are resolved. There are several reasons for this. For example, you do not want to inform someone late at night about a critical problem only to give them a warning. Also, a problem may oscillate between two severities. The event processors incur unnecessary overhead by repeatedly changing event severity.

Most monitoring agents do not send events to indicate a lessened severity. They normally inform as a problem increases in severity, and re-arm only when the condition is resolved.

## 2.7.2 Implementation considerations

Escalation can be performed automatically by a capable event processor or manually by console operators. Therefore, consider the first best practice in the list that follows.

Any monitoring agent or tool capable of the function can escalate a problem. The best place depends on both the tools used and the type of escalation. Consider the last two best practices in the list that follows.

For implementation, consider the following best practices:

- Automate escalation whenever possible.

When escalation is automated for unhandled problems, it occurs as soon as an acceptable, predefined time interval has expired. Similarly, a worsening condition and business impact escalation, when automated, occur immediately upon receipt of the relevant events. Operators perform escalation less precisely, only when they notice that a condition requires it.

If you do not have a well-defined escalation process or it is too complicated to escalate using your toolset, allow operators to do it. Holding them accountable for ensuring the timely handling of problems gives them incentive them to perform the required escalation.

- Use the trouble-ticketing system to escalate problems that do not receive timely action.

This type of escalation typically requires modifying an open trouble ticket and notifying support personnel. These functions are best performed in the trouble-ticketing system itself. See 2.6, “Notification” on page 56, and 2.9, “Trouble ticketing” on page 68, for details.

If trouble-ticketing software is not used, automate the escalation using a capable event processor, preferably the same one used to notify for problems.

- Perform worsening condition and business impact escalations at the first capable event processor that receives the relevant event or events.

These types of escalation are event driven, rather than timer dependent. They can be performed most quickly when handled immediately by the first receiver of the relevant events. Escalating at the lowest level of the event processor hierarchy facilitates event synchronization because it is generally easier to synchronize events upward through the hierarchy than downward. See the following section for details.

Escalating for business impact at the first capable processor ensures that the event has the correct severity when sent to subsequent event processors in the hierarchy. This minimizes the need to synchronize the events between the processors.

## 2.8 Event synchronization

Changes made to events at one event processor can be propagated to others through which the event has passed. This is known as *event synchronization*.

There are two main areas where event synchronization is key:

- Forwarding and receiving events through a hierarchy of event processors
- Integrating with a trouble-ticketing system

Any event processor or trouble-ticketing system can change an event. Depending on the event processor that performs the update, the event changes must be propagated upward through the hierarchy (see Figure 2-6). Typically, the trouble-ticketing system notifies support personnel about problems and is at the top of the hierarchy. Therefore, changes made to trouble tickets are generally propagated downward to other event processors. If any event processor modifies an event, the synchronization is upward to the trouble-ticketing system and any event processors above it in the hierarchy, and downward to lower event processors.

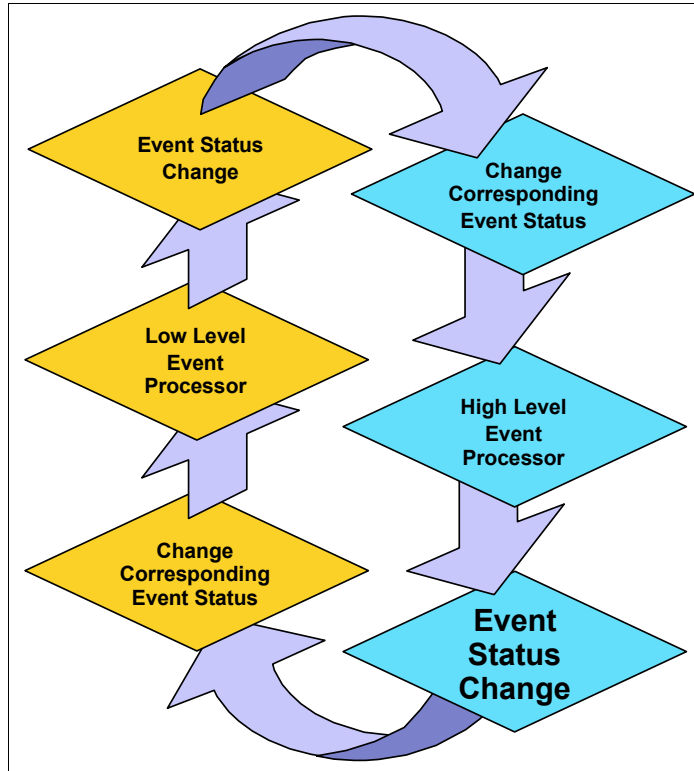


Figure 2-6 Upward and downward event synchronization

In general, it is easier to synchronize upward. Most event processors have the capability to automatically synchronize events upward, sometimes by merely reforwarding the changed event through the event processor hierarchy. Downward synchronization is more difficult and often requires custom coding.

## 2.8.1 Event synchronization best practices

When dealing with forwarding and receiving events through an event processor hierarchy, the most important aspect of the event is its status. By *status* we mean whether the event is open, closed, acknowledged, or dropped.

- Synchronize the status of events among event processors, including the trouble-ticketing system.

You want to make sure that, if an event is closed or dropped at one level of the hierarchy, it is also closed or dropped at every other level. If this is not done, you will have *orphaned events* in an open state. This can cause problems if you have any type of duplicate detection on that event.

Consider an orphaned event that was closed at a different level event processor when the problem that caused the event was resolved. The problem starts happening again, which generates another event. The event is sent to the event processor where, because of the existing orphan event, it is dropped as a duplicate.

It is important that the rules and any scripts that you set up at your event processors that deal with event forwarding and synchronization deal with the status of events.

When you deal with the integration of a trouble-ticketing system, keep in mind the status of an event. You may want to start with synchronizing the open and closed status of your trouble tickets with the open or closed status of the underlying events. Make sure that if your trouble ticket is closed, it closes the associated event and vice versa.

To take this one step further, you can send events back and forth when the event or ticket is updated. For example, if the problem ticket that was opened is acknowledged by the support group that opened it, you can have a communication sent back to the event processor changing the status of the event that caused the ticket to be generated.

- At a minimum, propagate event severity changes upward to the trouble-ticketing system and higher level event processors.

When a lower level event processor escalates an event, this information should flow upward. Notification typically occurs either at the trouble-ticketing system or a higher level event processor. As discussed in 2.7, “Escalation” on page 60, when events are escalated, someone needs to be informed. Therefore, the event processor used for notification needs to be told that the event has been escalated. Upward event synchronization performs this function.

When consoles are used at different levels of the event processor hierarchy, severity and other event changes may need to propagate downward. Suppose an organization has a central event processor at its main site that is used by its after-hours help desk. It also has distributed event processors in various time zones for use by support personnel during normal business hours. The central help desk raises the severity of an event on its console based on a user call. When the distributed support personnel assume ownership of the call in the morning, they need to know that the problem has been escalated.

## 2.9 Trouble ticketing

This section discusses integrating your event processing tool with a trouble-ticketing system. The focus is on event management, not problem

management. This section presents some best practices for problem management. They are mentioned, as necessary, as they relate to the integration of a trouble-ticketing system with an event processor.

## 2.9.1 Trouble ticketing best practices

This section covers the typical integration of a trouble-ticketing system with an event processor by discussing the process flow, which is illustrated in Figure 2-7.

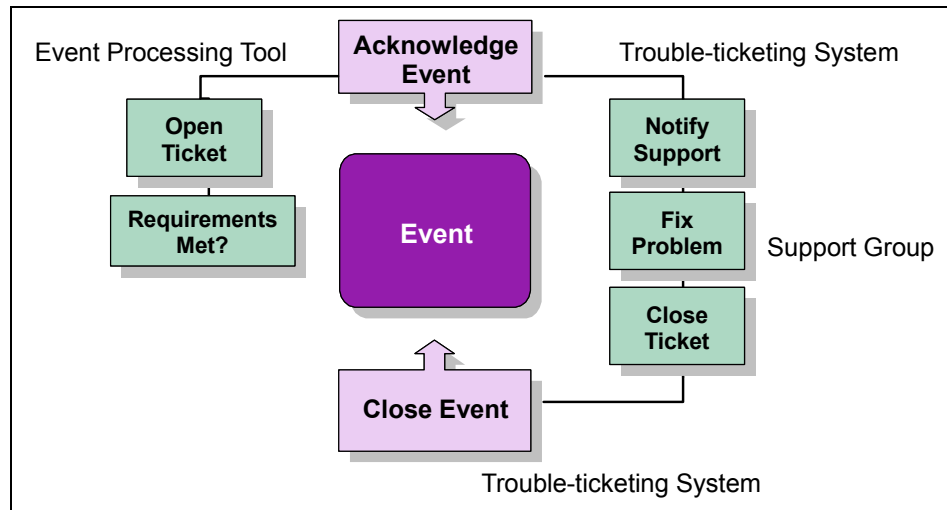


Figure 2-7 Trouble ticketing process flow

We start with an event. When this event arrives at the event processing tool, it has already gone through filtering and correlation. If it is determined through these processes that this event requires action, then the event processing tool sends this event to the trouble-ticketing system. See the first best practice in the list that follows this example.

After the event reaches the trouble-ticketing system, that system cuts a ticket. The ticket is assigned to the proper support group based on criteria within the event. That support group is then notified based on which notification was set up for this problem's priority. See the second best practice in the list that follows this example.

After the support group receives the page, it usually starts to fix the problem. During this process, it should keep the trouble ticket updated with their progress. When the problem is fixed, it should close the trouble ticket. This causes the trouble-ticketing system to trigger a communication to the event processing tool to close the event. Therefore, if the problem happens again, a new event is

generated that opens a new ticket. The support group must then determine why it did not actually resolve the problem. See the last three best practices in the list that follows this example.

In this example, you can implement the following best practices:

- Send all events to the trouble-ticketing system from the same event processor.

Most trouble-ticketing systems interface only to one event processor at a time. Also, this approach ensures that the trouble-ticketing system can initiate a close of the corresponding event when the ticket is closed.

- At the time of ticket creation, send a communication back to the event processing tool to acknowledge the event that triggered this ticket to be opened.

This usually takes place to prevent further tickets from being opened for the same problem. Duplicate detection should take place with the event as long as it is in acknowledged status. See 2.4.3, “Duplicate detection and throttling best practices” on page 50, for more details.

- If a trouble-ticketing system is in use, use it to report all problems requiring action and only those problems.

Now you can take the events from the previous section on notification that you decided were problem or actionable events. At this time, you can consider sending them to the trouble-ticketing system.

If you are not careful in choosing events that are going to open a ticket, you may start having problems in a couple different areas. If you have too many events going back and forth between your event processing tool and your trouble-ticketing system, you start to use up resources. This can happen both in the network and on the machines that are running the software.

The more events you send between your event processing tool and the trouble-ticketing system also takes a toll on your event synchronization. If you send loads of unnecessary events to open problem tickets, there is a greater chance that the acknowledging or closing events may be missed or dropped.

Another reason to choose your events carefully is to avoid mis-notifying support teams. You should only really notify critical, system down, or business impacting events that require action from support teams. If you start sending needless or unimportant pages to support groups, there is a big chance they may ignore the important ones.

You must also be careful with a reporting standpoint, which is usually carried out from the trouble-ticketing system. You do not want to have unimportant problems skew the reports.

In today's IT environment, it is essential to keep a good partnership between systems management and the various support teams. If the support teams have problems with the way tickets are opened and notifications are handled, it is hazardous to your event management system.

- Prioritize tickets based on event type, time-of-day, and criticality of source.

After you have events opening trouble tickets, consider the priority of the tickets that are opened. Figure 2-8 displays a sample event severity mapping.

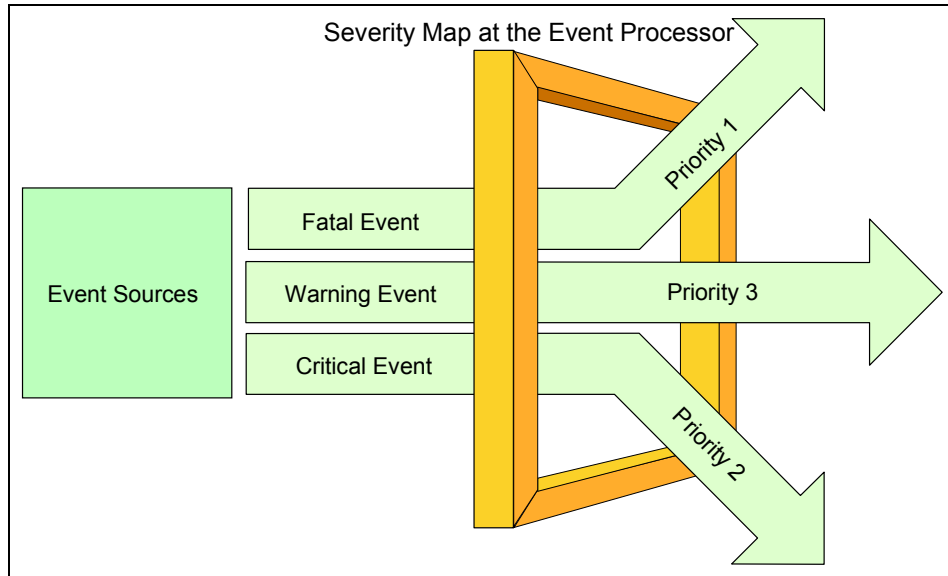


Figure 2-8 Mapping severities at the event processor

Usually it is a good idea to set the severity of your events at the source. This means that, when you send your event to the event processing tool, it should be sent with the severity that matches the priority of the ticket in the trouble-ticketing system.

Be aware of your company's current service levels when determining severity or priority. The severity that you use to send the event in should match the service levels defined in your organization for the event type, time of day, and criticality of the source. When setting up an integration with trouble ticketing, follow the processes and procedures already defined at your help desk. This makes it easier with the whole event management setup to tie into systems that are already in place.

- Implement on-call lists.

There are two ways to notify support groups from your trouble-ticketing system:

- **On-call lists:** Notify only the person on call. Maintain lists within the trouble-ticketing system of the current on-call personnel.
- **Group paging:** Page the whole support group. The person who is on call takes the call and the rest ignore the notification.

Usually it is wiser to go with the on-call list notification. This is because you send only one notification for each problem. It is less likely that notifications are dropped or ignored by support personnel who believe someone else is handling the problem. Escalation policies implemented in the trouble-ticketing system ensure that the problem is handled in a timely manner. See “Unhandled events” on page 60 for more information.

On-call lists are generally held within the trouble-ticketing system. This makes it easier to maintain because it is closer to the group listings. The lists usually go by a weekly basis depending on your organization. Keep in mind that it is important to keep these lists up to date.

## 2.10 Maintenance mode

When a system is in maintenance mode, its normal processing is disrupted by administrative functions such as applying fixes, reconfiguring components, or upgrading software. During that time, it is highly likely that one or more of the system’s resources is unavailable. If any of these resources is monitored for availability, events may be generated to report the down condition.

The resource may be down intentionally because it is being maintained, with the expectation that it will soon be restored to service. In this case, there is no need to alert anyone about its condition. However, its unavailability may be completely unrelated to the system maintenance being performed, requiring that someone be informed. The difficulty is in differentiating between the two situations.

Ideally, whenever system maintenance is performed, the administrator knows which resources are impacted and temporarily stops monitoring only those resources for the duration of the maintenance. In reality, this is nearly impossible to implement. Often, it is unclear as to exactly which resources are affected by the maintenance. Even if they are explicitly identified, it may be awkward to shut down monitoring for only those resources. Maintenance mode often entails rebooting a system one or more times. The monitoring agents will most likely restart automatically and need to be stopped again. Also, errors are often reported by other systems that normally access the system being maintained. It is impractical to assume that the appropriate monitors are shut down on other systems as well.

Although the ideal is impossible to attain, a mechanism must be in place that accounts for events from systems in maintenance mode to ensure that



extraneous events are not reported and real problems are not lost. A good approach is to inform the appropriate event processors that a system is in maintenance mode. Then have them take special action on the events from or about that system.

### 2.10.1 Maintenance status notification

An event processor needs to be informed when a system enters maintenance mode so it can take special action for events from the system. It must also know when to resume normal processing for those events. Therefore, a method is required to place a system into and remove it from maintenance mode.

- Only those processors that normally receive events concerning a system should be notified about its maintenance status.

If an event processor handles events from or about a system, it should be informed that the system is entering or leaving maintenance mode. This ensures that it can take the appropriate actions for events it may receive, as described in 2.10.2, “Handling events from a system in maintenance mode” on page 74.

Limiting the notification to only relevant event processors prevents the others from using cycles to check events for reference to the machine in maintenance mode.

- Use events to notify an event processor that a system is entering or leaving maintenance mode.

The event processors already have the ability to act upon events. The events from or about a machine in maintenance mode can be easily correlated with the maintenance notification event, and appropriate action taken.

Using events rather than some other means of notification eliminates the need for the event processor to access outside sources to determine which machines are in maintenance mode. If external files or queues are used to store the maintenance information, additional code may be required for the event processor to access that data.

- Automate the generation of the events as much as possible.

In sophisticated environments, the organization may use the change management system to automatically generate the maintenance notification events based upon its schedule. The shutdown and startup scripts or command files used during maintenance may also be modified to send the notifications.

The most common practice is to have the administrator send the events. While this method relies on the administrator’s memory, it allows for emergency changes that may not be included in the change management

system. It also allows for maintenance that does not require specific shutdown scripts to execute.

The administrator is provided with a desktop icon or script to run that automatically produces the required event. The change management procedures are updated to include generating the maintenance mode notification events as a required step in maintaining a system.

- Report extended maintenance mode situations as problems.

Experience has shown that administrators are more likely to notify when a system is entering maintenance mode than when it is leaving. They want to ensure that they are not informed of bogus problems about a machine that they are servicing. However, it is human nature to want to complete the maintenance as quickly as possible, particularly when it is performed after hours. As a result, many support people neglect to inform when a system is again operational. A method is needed to handle this situation.

First, the event processor needs to be given a time estimate for the maintenance window. This information can be sent in the maintenance mode notification event, stored on external media, or hardcoded into the event processor's rules. While any of these approaches work, sending the information in the event allows the greatest flexibility with the least effort. The administrator can easily differentiate between such maintenance (for example, parameter reconfiguration) and lengthier changes (such as software upgrades and database reorganizations). Modifying files and rules is more complex and more prone to error.

At the start of maintenance, the event processor can set a timer. After the elapsed time exceeds the estimate, the processor can generate an event to inform the administrator that the machine is in maintenance mode longer than expected, or it can merely resume normal event processing for the system. Administrators generally prefer to be notified. This prevents a potential flood of problem events, should the system still be legitimately in maintenance mode.

## 2.10.2 Handling events from a system in maintenance mode

When an event processor knows that a system is in maintenance mode, it can take appropriate action for events received from or about that system. The best practices to use for handling those events depends upon the organization and its policies for systems maintenance.

- In environments where the administrator maintaining the box has total control over the machine, *host-based maintenance* may be appropriate.

Giving an administrator total control over a machine in maintenance mode implies that it is acceptable to affect any system resource during the maintenance window. This approach also assumes that the administrator is

fully responsible for restoring all processes when the maintenance is complete. Therefore, events received from or about the box during this time do not require action and may be discarded.

We refer to the processing of dropping the events for systems in maintenance mode as *host-based maintenance*. This is a relatively simple method of handling the events and is easy to implement. However, it relies on the administrator to properly restore all functions on the machine. A condition may arise such as a filesystem filling that an administrator does not normally notice during maintenance mode. These problems may go unreported in host-based maintenance.

- Where host-based maintenance is not appropriate, cache events and report them after maintenance mode is terminated.

Caching events received from or about a system in maintenance mode ensures that real problems unrelated to the maintenance are not lost. It also preserves the correlation relationship among events. This solution should be favored in organizations where production applications may continue to run on a system that is undergoing maintenance for other processes. It may also be used to validate that the system is completely restored afterwards.

When the event processor receives events for the system undergoing maintenance, it caches them. It can also apply correlation rules to them to drop extraneous events and to clear problems. When the machine comes out of maintenance mode, the event processor waits a short time to receive and process clearing events for the system resources affected by the maintenance. It then reports any outstanding problems.

Events for which no clearing event is available are always reported using this method, even if the problem they reference no longer exists. Whenever possible, configure monitoring agents to send clearing events. This minimizes the number of these events that are inadvertently reported.

### 2.10.3 Prolonged maintenance mode

Sometimes the resolution to a problem is known but cannot be immediately implemented. For example, a short-on-storage condition may arise because a machine does not have adequate memory. Adding memory to the system is planned, but the hardware is scheduled to ship in a few days.

In these situations, it is undesirable to report the error every time it occurs. The solution is known, but cannot yet be implemented. There are several ways to handle this situation. The first is to reconfigure the monitoring agent so it does not report the error. This effectively stops the event from flowing. However, it relies upon the memory of an administrator to restart monitoring after the fix is implemented. In the case of hardware, the solution may not be implemented for

several weeks. It is highly likely that the support person will forget to re-enable the monitor at that time.

A second approach allows the event to remain open until it is resolved and to discard duplicates in the meantime. This method also has some problems. The problem may occur intermittently and be cleared automatically by an event. Leaving the event open requires a reconfiguration of the event processing rules, which has the same drawbacks as reconfiguring a monitor. Also, some event processors perform duplicate detection only on events in cache, which is cleared when the processor is recycled.

To address the shortcomings of the other two solutions, we propose that you temporarily ignore events whose resolution is known, but cannot yet be implemented.

To implement this solution, the event processor needs to be told which event to ignore, from which host, and for how long. This information may need to be stored in an external file or queue so it can be accessed by the event processor upon restart. If the event processor supports this, it may be loaded into cache at startup for more efficient runtime access.

Note that normal maintenance mode event processing does not require storing the list of nodes being maintained on external media. Maintenance windows are expected to be relatively short. They may be scheduled so that they do not occur when the event processor is scheduled for reboot.

When the event processor receives an event, it checks to see if the event should be temporarily ignored and suppresses it if appropriate. If the event is not reported, a support person does not waste time pursuing a known problem.

There are several advantages of this solution. The monitors and event processing rules can remain intact. This implies that an administrator does not need to remember to restore monitoring of the system resource. As soon as the estimated time has passed, the resource is monitored again. During this prolonged maintenance mode, the system is still monitored for other conditions that are unrelated to the known, but not yet fixable problem. Finally, if desired, the event can be correlated before it is ignored. This may prevent the investigation of symptom events for which no action can be taken.

## **2.10.4 Network topology considerations**

When a network device is placed in maintenance mode, a potentially large number of systems can be affected. If the component is a single point of failure, any network traffic that traverses a path containing it may be disrupted. In this

case, it is necessary to know the network topology before determining whether an event is the result of the maintenance.

In this case, we propose the *best practice* to use the topology-based correlation capabilities of SNMP-based managers to handle events resulting from network maintenance.

While it is possible to provide network topology information to other types of event processors, the procedure is often complex and difficult to implement. Most networks are dynamic, implying frequent updates to the topology information supplied to the processors. This quickly becomes impractical, particularly in large networks.

Allowing the SNMP-based manager to correlate the events means that only the root cause event is presented to the other event processors. This event references the network component being maintained. Since the other event processors were informed the network device is under maintenance, they can handle the event as described in 2.10.2, “Handling events from a system in maintenance mode” on page 74.

When a network component is maintained, some events caused by the maintenance may be reported. For example, a Web server that is trying to access its database server across the network path may report a communication failure. Since neither the Web server nor the database server is in maintenance mode, the event is reported.

While it is possible, theoretically, to handle these cases, it is usually not worth the effort involved. The communication between any two devices may be affected by a large number of networking components. Moreover, in today’s grid and on-demand environments, it is not unusual for an application to run on different hosts when required. Not knowing on which server the application runs at any given time makes it difficult to determine which applications are affected by a network failure.

Consider redundant network paths to minimize communication outages due to both network component failures and maintenance.

## 2.11 Automation

There are several ways to plan for and implement automation. Some organizations choose to implement correlation first and then to analyze their common problems, identifying events that may be used as triggers for automated action. Others decide automated actions at the same time as filtering,

correlation, and notification. Still others use problem post mortem investigations to uncover automation opportunities.

The approach that an organization chooses depends upon the current monitoring environment, event management policies, and the employees' skill levels. If little or no monitoring is in place, a company may decide to analyze all possible events from a monitoring tool before implementing it, making filtering, correlation, notification, and automation decisions concurrently. In environments with robust monitoring already in place, automation may be added to it. Where staffs are highly skilled and can provide insight into which events should trigger automation, the decisions can be made reliably before problems happen. Novice support staffs may wait until problems occur before they identify automation opportunities, working with vendors' support centers to gain insight into the reasons the problems occurred and how to prevent them.

Regardless of how an organization decides to handle planning and implementing automation, there are several best practices to observe as explained in the following sections.

### **2.11.1 Automation best practices**

The first step in implementing automation is deciding which events should trigger automated actions. Here are several guidelines to use in making this determination:

- Do not over automate.

Sometimes organizations are overzealous in their correlation and automation efforts. In a desire to improve the efficiency of their monitoring environments, they quickly set up automation for everything they can.

There are some pitfalls to this approach that arise from not understanding the ramifications of potential automated actions before implementing them. For example, a locked account resulting from a user mistyping a password several times needs to be reset, but if it was caused by hacking attempts, a security investigation may be required. Automatically resetting accounts based upon receipt of the account locked event is not a good idea.

Similarly, an event may be used to report more than one problem, necessitating different actions depending upon which problem caused the event. Perhaps actions may be required when an event references a certain host but not others.

These are a few examples of things to consider when determining whether automated action is appropriate for an event. Remember, it is better to be judicious in choosing automated actions and implement fewer than to implement automation whose consequences are unknown.

- Automate problem verification whenever possible.

Sometimes it is not possible to filter all events that do not indicate real problems. For example, as discussed in 1.3.10, “Automation” on page 19, an SNMP manager that queries a device for its status may not receive an answer back due to network congestion rather than the failure of the device. However, the manager believes the device is down and sends an event. Before assigning this event to someone, it is advantageous to determine if it is truly a problem.

SMEs who understand various event sources well may be able to identify events such as this one that may sometimes be *false alarms*. Likewise, help desk personnel learn from experience which events do not always represent problems. Use the expertise of these people within the organization to list events that require verification.

After the events are identified, work with the SMEs to determine if the problem verification procedures lend themselves to automation and automate whenever possible. This minimizes the amount of time that the support staffs spend investigating false events.

- Automate gathering diagnostic data if the data may disappear before the problem is investigated, multistep or long running procedures are required to capture it, or support staff lacks the skills to acquire it themselves.

In cases where diagnostic data may disappear before a support person has time to respond to the event (such as the list of processes running during periods of high CPU usage), automating the gathering of diagnostic data may be the only way to determine the cause of the problem. All events reporting these type of intermittent problems should have automated diagnostic data collection associated with them.

The situation is less clear for events whose diagnostic information remains available in the system until requested by an operator. In these cases, automate the diagnostic data gathering if multiple steps are required to obtain the data. This way, the user does not have to remember the steps, and user errors, such as typos, are eliminated.

Also, if the diagnostic gathering procedures take a long time to run, automating the data collection ensures that the data is available to the support person sooner, reducing the mean-time to repair for the problem.

Automating diagnostic data gathering in circumstances where the user can type a single command to produce the data may not make sense, since the user has to run at least one command to view the collected data as well. In this case, unless the command takes a while to run or the support staff is not highly skilled, do not automate the data collection. This saves on cycles in the event processors handling the event.

- ▶ Automate recovery only for real problems that always require the same sequence of actions.

Obviously, if an event does not represent a real problem, it does not require a recovery action, automated or otherwise.

For real problems, be sure that the same sequence of actions is always required to resolve the problem before automating. This sequence can contain conditional logic, as long as all the possible conditions and the actions to take for them are known. Also, ensure that the steps can be automated. A sequence of actions that requires operator interaction with a graphical user interface (GUI), for example, may not be automatable.

See the first best practice, “Do not over automate”, in this list for additional considerations in determining whether to automate the recovery action.

- ▶ Consider cross-platform correlation sequences for possible automation opportunities.

Cross-platform correlation refers to the process of associating events about different system resources. These event sequences are often excellent sources for automated recovery actions.

Often, cross-platform correlation sequences result in symptom events that require action. This is because the support person handling the first resource type does not usually have administrative responsibility for the second. Also, many systems are not sophisticated enough to recognize the system resources affected by a failure and to automatically recover them when the failure is resolved.

In “Cross-platform correlation” on page 13, we provide an example of a process that dies as a result of a full file system. The corresponding events can be associated, and the process can be restarted automatically when the filesystem problem clears.

## 2.11.2 Automation implementation considerations

We discuss several types of automation that can be executed upon receipt of an event. You must perform these in a specific order to ensure that the desired results are achieved.

- ▶ Automate as close to the source as possible.

There are several factors that determine where to automate.

- The selected tool must be capable of performing the automation. For example, when implementing automated action for a network device using SNMP commands, the tool used to perform the automation must be capable of communicating via SNMP.



- The automation should be easily maintainable. Sometimes it is possible to perform the action from more than one tool. If it is difficult to implement and maintain the automation using the tool closest to the source, use the next closest one instead.
- Performance considerations may dictate which tool to use for automation. If a tool requires many processing cycles or uses much bandwidth to automate a particular function, it should be overlooked in favor of one that performs better.
- If correlation is required before executing an automated action, the closest point from which automation can be triggered is the event processor at which the events in the associated events converge.
- Check to see if a device is in maintenance mode before performing any automated actions.

As discussed in 2.10, “Maintenance mode” on page 72, if a device is in maintenance mode, any problems reported for that device are suspect. No automated recovery actions should be taken for events received about devices in maintenance mode. Diagnostic automation can be performed, if desired. This ensures that the diagnostic data is available should the problem still exist after the device comes out of maintenance mode.

- Perform *problem verification* automation first.  
If this type of automation is done, it should always precede all other types of event processing such as correlation, recovery and diagnostic automation, notification, and trouble ticketing. None of these actions is necessary if there is not really a problem.
- Perform automated diagnostic data collection after verifying that the problem really exists and prior to attempting recovery.  
Obviously, if there is not really a problem, no diagnostic data is required. However, you must perform this step before you perform the recovery actions, since the diagnostic data sometimes disappears after the problem is resolved. An example of this is logs that are overwritten whenever an application starts.
- For real problems for which recovery action is desired, try the recovery action prior to reporting the problem. If it fails, report the problem. If it succeeds, report the situation in an *incident report*.

For the reasons stated earlier, this type of automation should succeed both problem verification and diagnostic data gathering automation. However, running recovery actions should be performed prior to reporting problems. This ensures that only events requiring operator intervention are reported through notification and trouble ticketing. If the recovery action succeeds, the problem should not be reported to prevent unnecessary events from cluttering the operator console.

Sometimes it may be useful to know a problem occurred, even if it recovered automatically. Consider, for example, an automated action to remove core files from a full UNIX file system. When the file system fills, an event is sent, which triggers the removal of core files from the file system, freeing adequate space. Since the file system now has an acceptable amount of freespace in it, the problem is closed and not reported. An application may be producing core files repeatedly and filling the file system. It is useful to know about this condition to identify and resolve the application problem.

Opening incident reports for conditions that are automatically recovered is the preferred method to track them. Incident reports make the information available to a support person when they choose to examine it. Reviewing the incident reports highlights *flapping* or *fluttering* error conditions, such as the one described earlier, and may lead to their resolution.

## 2.12 Best practices flowchart

In this chapter, we discuss the best practices for the various types of processing to perform for events. The purpose of this section is to recommend the order in which to perform the processing. The flowchart in Figure 2-9 shows a recommended sequence for handling events.

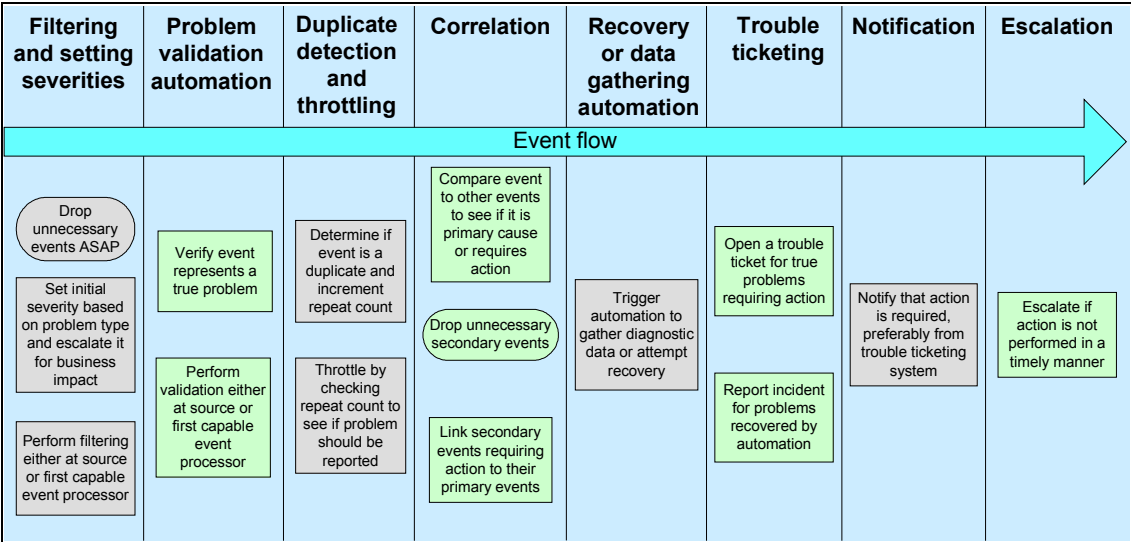


Figure 2-9 Event processing flowchart

**Note:** The exact event processing sequence that you implement depends upon the capabilities of the various monitoring agents and event processors in your environment. This flowchart was developed with the Tivoli product suite in mind.

In general, if best practices dictate performing the event processing as close to the source as possible and a monitoring agent is capable, consider configuring the agent to perform the function. For example, IBM Tivoli Monitoring can be configured to perform such tasks as filtering unnecessary events, throttling based on number of times a condition occurs, correlating multiple error conditions before reporting, and executing automation.

For actions best performed at a central site, such as trouble ticketing and notification, rely on the appropriate event processors to perform those functions.

Perform filtering and forwarding first. If an event is not necessary, suppress it in the source or as close to it as possible. If it is necessary or you are not sure, set severity based on the problem type and potential business impact, if known. Handling filtering first prevents handling unnecessary events, saves network bandwidth, and minimizes processing cycles required at event processors higher in the event management hierarchy.

At this point, the event represents a potential problem. If possible, verify the condition through automated means. Ideally, perform problem validation as close to the source as possible. Again, this practice saves bandwidth and processing power by suppressing unnecessary events as early as possible.

If the event survives filtering and problem validation, it likely is a real problem. However, it may have already been reported, or it may recover itself after a short time. Perform duplicate detection and throttling to block the event from traveling further if it is already reported. Ensure that it is forwarded if it happened a predefined number of times within a given time frame. Increment a counter of occurrences of an event within a time frame. This provides data that the support person can use to determine the extent of the problem and the load the event processor is handling.

Next, compare the problem event to other received events and classify them as a primary or secondary event. If it is a secondary event that requires action, link it to the primary and forward it. If it does not require action, drop it to prevent further processing load on this and other event processors. If necessary, delay the comparison long enough to allow other related primary events to be received.

At this point, the event represents a problem requiring action. Before informing a support person, perform automation to gather diagnostic data or attempt

recovery, if desired. Use incident reports to record success recovery and trouble tickets for unsuccessful.

The appropriate event processor or trouble-ticketing system can then inform the support person there is a problem requiring action. If the problem is not acknowledged or resolved within a preset time interval, escalate the problem by raising its severity or notifying additional support personnel.



## Overview of IBM Tivoli Enterprise Console

This chapter provides an overview of the IBM Tivoli Enterprise Console product and its main components. For detailed information about how IBM Tivoli Enterprise Console is used for event management, see Chapter 6, “Event management products and best practices” on page 173.

To learn about the architecture behind the IBM Tivoli Enterprise Console, see *IBM Tivoli Enterprise Console User's Guide, Version 3.9, SC32-1235*.

## 3.1 The highlights of IBM Tivoli Enterprise Console

The IBM Tivoli Enterprise Console product is a powerful, rules-based event management application. It integrates network, systems, database, and application management. And it provides sophisticated, automated problem diagnosis and resolution to improve system performance and reduce support costs. The product focuses on time to value and ease-of-use with out-of-the-box best practices to simplify and accelerate deployment.

The highlights of IBM Tivoli Enterprise Console include:

- ▶ Provides a centralized, global view of your computing enterprise
- ▶ Filters, correlates, and automatically responds to common management events in many tiers so events can be treated as near the source as possible
- ▶ Implements best-practices event management out-of-the-box
- ▶ Extends end-to-end management and diagnosis of your IT environment, through the integrated network management and auto-discovery functions of NetView
- ▶ Acts as a central collection point for alarms and events from a variety of sources, including those from other Tivoli software applications, Tivoli partner applications, custom applications, network management platforms, and relational database systems

The IBM Tivoli Enterprise Console helps to effectively process the high volume of events in an IT environment by:

- ▶ Prioritizing events by their level of importance
- ▶ Filtering redundant or low-priority events
- ▶ Correlating events with other events from different sources
- ▶ Determining who should view and process specific events
- ▶ Initiating automatic actions, when appropriate, such as escalation, notification, and the opening of trouble tickets
- ▶ Identifying hosts and automatically grouping events from the hosts that are in maintenance mode in a predefined event group

IBM Tivoli Enterprise Console also includes IBM Tivoli Risk Manager (limited license). It provides monitoring and management of firewalls and intrusion detection systems. And it includes IBM Tivoli Comprehensive Network Address Translator to enable integrated management of overlapping IP domains.

Refer to the following product documentation to gain more detailed information regarding IBM Tivoli Enterprise Console:

- ▶ *IBM Tivoli Enterprise Console Command and Task Reference, Version 3.9, SC32-1232*
- ▶ *IBM Tivoli Enterprise Console Installation Guide, Version 3.9, SC32-1233*
- ▶ *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9, SC32-1234*
- ▶ *IBM Tivoli Enterprise Console User's Guide, Version 3.9, SC32-1235*
- ▶ *IBM Tivoli Enterprise Console Release Notes, Version 3.9, SC32-1238*
- ▶ *IBM Tivoli Enterprise Console Event Integration Facility Reference, Version 3.9, SC32-1241*
- ▶ *IBM Tivoli Enterprise Console Adapters Guide, Version 3.9, SC32-1242*
- ▶ *IBM Tivoli Enterprise Console Rule Set Reference, Version 3.9, SC32-1282*

You can find this documentation on the following Web site:

<http://www.ibm.com/software/tivoli/library>

## 3.2 Understanding the IBM Tivoli Enterprise Console data flow

This section explains IBM Tivoli Enterprise Console as a process, with inputs, processing, and outputs. Viewed this way, IBM Tivoli Enterprise Console is a powerful application composed of integrated tools to collect event input and translate it into useful event management information and automated problem diagnosis and resolution output.

Figure 3-1 shows how each component of IBM Tivoli Enterprise Console fits into the process.

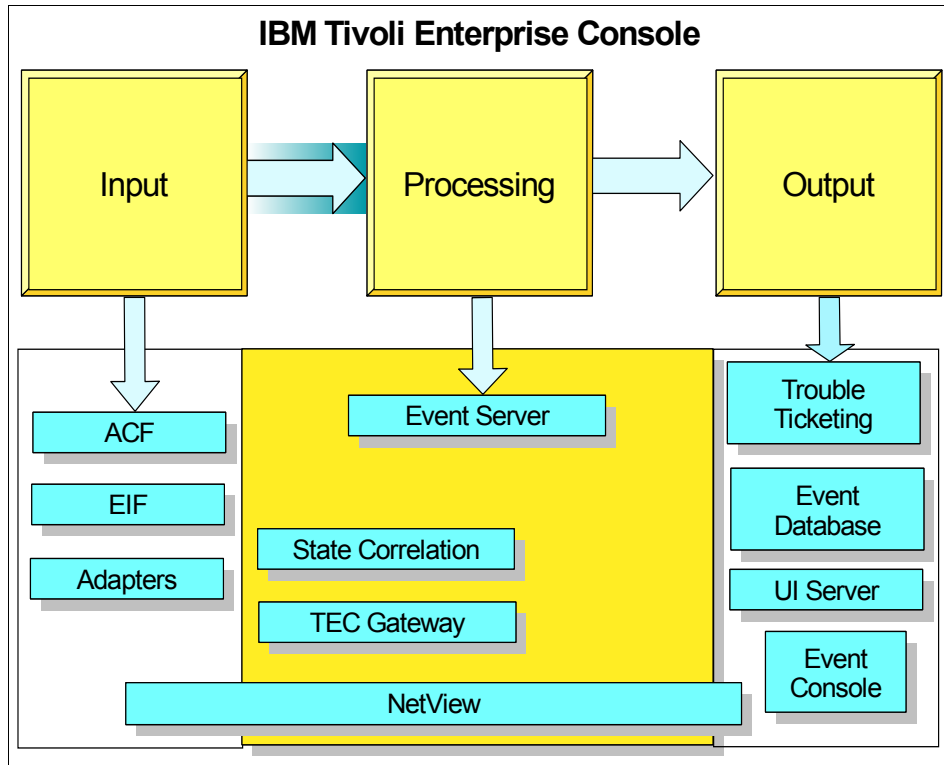


Figure 3-1 IBM Tivoli Enterprise Console process and components

### 3.2.1 IBM Tivoli Enterprise Console input

During event collection, IBM Tivoli Enterprise Console's function is to detect all variations in the source that can possibly represent a cause or effect of a problem or incident, format that information in a standard way, and ensure that these variations should be processed by IBM Tivoli Enterprise Console.

IBM Tivoli Enterprise Console can collect events from a variety of event sources, including internal events, events from IBM Tivoli Enterprise Console adapters, and from integration with other Tivoli or non-Tivoli applications. Events can be buffered at the source, gateway, and server. This helps to prevent problems in the network or with unavailable services resulting in loss of event information.

The Adapter Configuration Facility (ACF) is a powerful tool for centralized configuration of a great number of source clients. It uses some of the capabilities provided by Tivoli Management Framework to create profiles with adapter configurations and distribute them to groups of endpoints. ACF enhances IBM Tivoli Enterprise Console's ability to perform rapid deployment and maintenance.



### 3.2.2 IBM Tivoli Enterprise Console processing

Processing a huge number of events for problem diagnosis and resolution is the main role of IBM Tivoli Enterprise Console. Event processing is done in many instances, in different places, in the following order in which they occur:

1. The initial processing is done in the event source by the adapter. It filters events directly in the source before they are sent to the IBM Tivoli Enterprise Console gateway.
2. In the gateway, state correlations are responsible for summarizing events to the IBM Tivoli Enterprise Console server.
3. The server then correlates the various events from different sources into a few root cause and service impact events. It performs automated actions to resolve the root cause problem.
4. The event console is used for output and final processing. Operators use it to view events and sometimes change them manually based on their own experience or organizational procedures.

IBM Tivoli Enterprise Console processing can be divided by the meaning of its objectives, in two parts: problem diagnosis and resolution attempts. Most of the work is done during problem diagnosis when the events are filtered, correlated, and actions are taken to find the root cause of the problem.

After finding the root cause, actions are taken to solve the problem. If the problem remains, another problem diagnosis and resolution attempt can be done.

Figure 3-2 shows the relationship between these parts and their input and output.

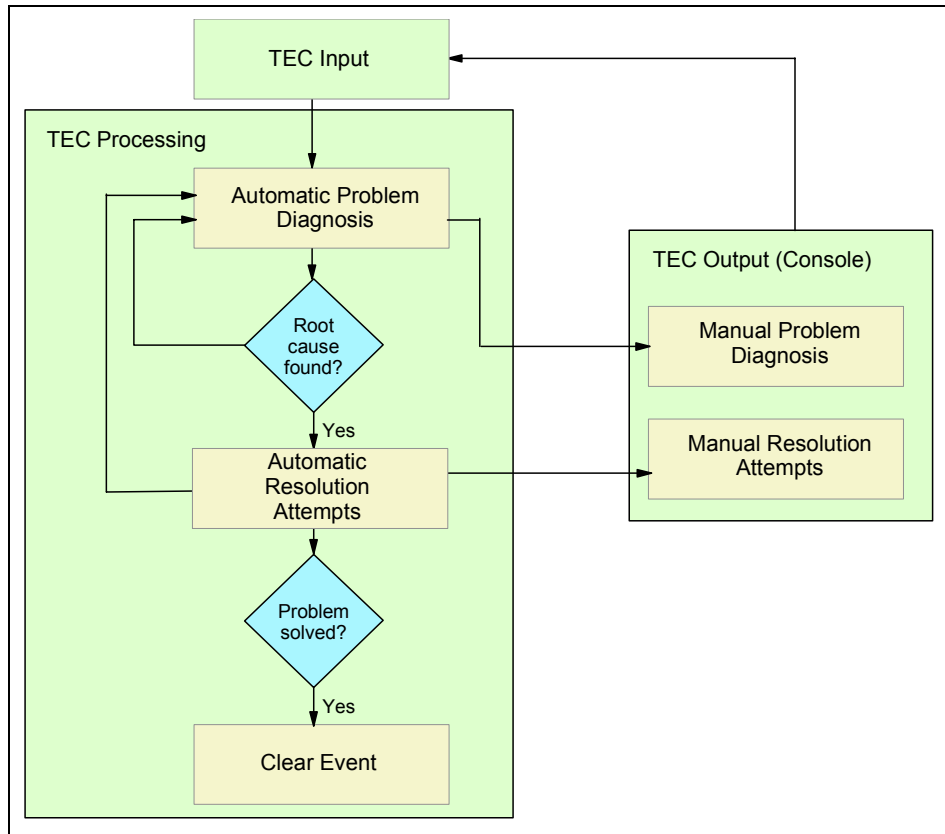


Figure 3-2 Processing fluxogram

### 3.2.3 IBM Tivoli Enterprise Console output

The purpose of the output is to provide detailed information to the right people when they need it. IBM Tivoli Enterprise Console can send output to consoles and databases.

The IBM Tivoli Enterprise Console is used as the first output, giving Tivoli Administrators the ability to view and interact with problems. This complements the automatic processing with manual problem diagnosis and resolution attempts.

Event databases are used as output as well to record event information for future processing and reporting. IBM Tivoli Enterprise Console can be integrated with other applications, such as trouble ticketing, customer relationship management (CRM), and billing systems.

## 3.3 IBM Tivoli Enterprise Console components

This section provides an overview of the IBM Tivoli Enterprise Console components. They are described in the order in which an event flows from its source to the operator.

### 3.3.1 Adapter Configuration Facility

The ACF provides the ability to configure and distribute Tivoli adapters from a central site, using a graphical user interface (GUI). The ACF allows you to create profiles for adapters, configure adapter configuration options, and distribution options. The adapters can then be distributed to the profile's subscribers using menu options or drag-and-drop functionality. This feature allows changes to be made in a central location and then distributed to the remote endpoints or managed nodes.

### 3.3.2 Event adapter

An adapter is a process that monitors resources so that they can be managed. These monitored resources are called *sources*. A source is an application (for example, a database) or system resource (for example, an Network File System (NFS) server). When an adapter detects an event generated from a source (generally called a *raw event*), it formats the event and sends it to the event server. The event server then further processes the event.

Adapters can monitor sources in the following ways:

- ▶ An adapter can receive events from any source that actively produces them. For example, Simple Network Management Protocol (SNMP) adapters can receive traps sent by the SNMP.
- ▶ An adapter can check an ASCII log file for raw events at configured intervals if the source updates a log file with messages.

An event adapter can buffer events. When event buffering is enabled for an event adapter and the event source cannot connect to the event server, events are buffered until a connection can be established to the event server. When a connection is established, the event adapter sends the buffered events to the event server.

You can specify one or more secondary event servers for an event adapter. A secondary event server is a backup event server that receives events when the IBM Tivoli Enterprise Console gateway cannot contact the adapter-specified event server. You can specify one or more secondary event servers in the IBM Tivoli Enterprise Console gateway configuration file.

### 3.3.3 IBM Tivoli Enterprise Console gateway

The IBM Tivoli Enterprise Console gateway receives events from TME® and non-TME adapters and forwards them to an event server. By default, the IBM Tivoli Enterprise Console gateway uses a connection-oriented service to the event server. A *connection-oriented service* is one that establishes a connection when the IBM Tivoli Enterprise Console gateway is started, and the connection is maintained for all events.

The IBM Tivoli Enterprise Console gateway provides the following benefits:

- ▶ Greater scalability, which allows you to manage sources with less software running on the endpoints
- ▶ Improved performance of the event server

The IBM Tivoli Enterprise Console gateway bundles events before sending them to the event server. This reduces the amount of communication tasks that the event server or the Tivoli server must perform.

- ▶ Simple deployment of adapters and updates to adapters using profiles in the Adapter Configuration Facility
- ▶ New to IBM Tivoli Enterprise Console 3.9, the IBM Tivoli Enterprise Console gateway

This gateway can provide event correlation and filtering closer to the sources. This reduces the number of events sent to the event server and improves network performance by decreasing the amount of network traffic.

**Note:** With the introduction to the state correlation gateway, IBM Tivoli Enterprise Console is moving away from Tivoli Availability Intermediate Manager. This product's purpose was to offload processing from the IBM Tivoli Enterprise Console Server by executing event correlation itself, and it was a separate install IBM Tivoli Enterprise Console. The new IBM Tivoli Enterprise Console gateway state correlation feature is built directly into IBM Tivoli Enterprise Console 3.9. You should begin merging your existing AIM environment to the state correlation engine.

### 3.3.4 IBM Tivoli NetView

IBM Tivoli NetView is now included with the IBM Tivoli Enterprise Console product set. The Tivoli NetView component provides the network management function for the IBM Tivoli Enterprise Console product. The Tivoli NetView component monitors the status of network devices and automatically filters and forwards network-related events to the IBM Tivoli Enterprise Console product.

Tivoli NetView is a different product that is integrated into IBM Tivoli Enterprise Console. For more information about Tivoli NetView, see Chapter 4, “Overview of IBM Tivoli NetView” on page 101.

### 3.3.5 Event server

The *event server* is a central server that handles all events in the distributed system. It uses a relational database management system (RDBMS) (event database) to store events. Then it evaluates these events against a set of classes and rules to determine if it should receive the event first and then respond to or modify the event automatically. Only one event server can be in each Tivoli management region.

### 3.3.6 Event database

IBM Tivoli Enterprise Console uses an external RDBMS to store the large amount of event data that is received. The RDBMS Interface Module (RIM) component of the Tivoli Management Framework is used to access the event database.

For additional information about the event database, see the *IBM Tivoli Enterprise Console Installation Guide, Version 3.9, SC32-1233*.

### 3.3.7 User interface server

The user interface server is a process, which runs on a managed node in a Tivoli Management Region (TMR). It is responsible for providing a communication channel between event consoles and the event server. It handles transaction locking and prevents multiple consoles from responding to the same event. It also handles automatically updating event status on all event consoles.

### 3.3.8 Event console

Event consoles provide a GUI that allows the IT staff to view and respond to dispatched events. A senior administrator configures multiple event consoles based on the responsibilities of the IT staff. Users can have independent or shared views of events.

New to IBM Tivoli Enterprise Console 3.9, there are two versions of the event console as discussed in the following sections.

#### **Java event console**

The Java™ version of the event console has existed since IBM Tivoli Enterprise Console 3.7. It can be installed on a managed node, endpoint, or a non-Tivoli

host. The Java event console provides a set of features needed by Tivoli Administrators to perform configuration tasks, start Tivoli NetView functions, run local automated tasks, and manage events.

### **Web event console**

The Web version of the event console is new with IBM Tivoli Enterprise Console 3.9. You can use it only to manage events from a Web browser. The Java console is still necessary to perform any other task available from the event console.

## **3.4 Terms and definitions**

The following section describes the major terms used with IBM Tivoli Enterprise Console. It also presents some new features in IBM Tivoli Enterprise Console 3.9.

### **3.4.1 Event**

The central unit of information is the *event*. An event is any significant change in the state of a system resource or application. Events can be generated for problems and for successful completions of tasks. Events provide many different types of information, such as a host being down, someone unsuccessfully trying to log in to a host as an administrator, or a hard drive being nearly full. Events can also be generated to clear other events.

In IBM Tivoli Enterprise Console, an event is an object that has been created based on data that is obtained from a source that is monitored by an event adapter. Each event is identified by a class name, which the event adapter defines.

Examples of class names include Su\_Success, Su\_Failure, No\_Permission, and Printer\_Toner\_Low. Class names are used to label events, but each event contains additional information that helps to define and locate a potential problem. Sample event information is provided as an example to manage requests for additional event information that an operator might need. Each event class has a template, which can be modified to include additional information about an event and the action required to resolve the problem. This facilitates the creation of a comprehensive online system of event information and troubleshooting.

### **3.4.2 Event classes**

Adapters separate information into *event classes*. They format the information into messages that represent specific instances of event classes. Then, they

send the information to the event server. The event server processes the information, along with information received from other adapters.

**Note:** Event classes are classifications of events. Do not confuse them with Tivoli objects.

Event classes can be divided into subclasses to facilitate a further breakdown of information so that more detailed rules can be applied to the event information. Essentially, event classes are an agreement between the adapter and the event server about what information the adapter sends to the event server.

Event classes are defined in event class definition files. The base event class definition file (root.baroc) is located in the \$BINDIR/TME/TEC/default\_rb/TEC\_CLASSES directory. Additionally, other event classes are subclasses of the base event class. For event classes, you can create Basic Recorder of Objects in C (BAROC) files and specify the event server to load those files.

You can have multiple BAROC files on an event server. When you develop a new adapter, in the BAROC files, define the types of events that the adapter can send to the event server.

See the *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9*, SC32-1234, for a detailed discussion about BAROC files.

### 3.4.3 Rules

A *rule* consists of a set of expressions used to determine if an event meets the rule conditions. A rule also includes a set of actions that are taken when an event meets the specified rule conditions.

A rule can specify, but is not limited to, the following actions:

- ▶ **Duplicate detection:** Automatically discard duplicate events within a specific time interval.
- ▶ **Thresholding:** Accumulate events until a certain number are received within a time interval. Then issue one representative event.
- ▶ **Escalation:** Increase the severity of an event or perform an action if a certain number of events are received within a time interval.
- ▶ **Correlation:** Based on the relationships between events and system problems, emphasize the most important event relating to a problem (typically the root cause of the problem) and reduce the significance of those events that are merely effects that are corrected when the root cause is corrected. For example, if an uninterruptible power supply shuts down a system

gracefully during a power failure, IBM Tivoli Enterprise Console can indicate that the root cause of the shutdown is the power failure before personnel are dispatched to repair the system.

There are five types of rules as discussed in the following sections.

### **Plain rule**

A plain rule is used with incoming new events, or with previously received events to be re-analyzed. Re-analysis of a previously received event is called a *redo request*.

Plain rules allow you the flexibility to use any predicate or Prolog feature in its actions.

### **Change rule**

A change rule is used with previously received events that have a request to change their information. A request to change an event's information is called a *change request*. For change requests, the change rules are checked before the change is actually made. This timing lets you develop rules to take action depending on the old value of an attribute, the new value of the attribute, and the origin of the change request. Change requests can be generated by:

- ▶ An event console, for example, an administrator changes the status of an event from OPEN to CLOSED
- ▶ Calling certain predicates within rules, for example, the `place_change_request` predicate
- ▶ Receiving an event from an adapter with a value of CLOSED for the status attribute

Change rules allow you the flexibility to use any predicate or Prolog feature in its actions. They can only specify plain actions. Redo actions and reception actions are considered errors when they are specified in change rules.

### **Timer rule**

The timer rule is used when a previously set timer on an event expires. Timers can be set on an event with the `set_timer` predicate in a rule. Sometimes you may want to wait for a period of time so that related events come in to help identify the root cause of a problem. Or perhaps you want to wait to ensure the event condition lasts long enough to be a problem where action is needed. With timer rules, you have the flexibility to use any predicate or Prolog feature in its actions.

Timer rules can only specify plain actions. Redo actions and reception actions are considered errors when they are specified in timer rules.



## Simple rule

Use simple rules with incoming new events or a redo request. A simple rule is not as flexible as a plain rule. For example, it contains predefined conditions and actions, and you cannot use a predicate or any Prolog feature in its actions. A simple rule does not do any correlation with other events in the event cache, except for dropping duplicate events.

## Correlation rule

Use correlation rules with incoming new events or a redo request. A correlation rule lets you establish a causal relationship between two event classes. One event either causes the other to be generated or causes the other to be closed.

With a correlation rule, you can propagate the value of the status attribute from a cause event to an effect event. For example, when closing a cause event, a linked effect event can be automatically closed. Correlation rules are called *compound rules* in the rule builder dialogs.

### 3.4.4 Rule bases

A *rule base* is a collection of event class definitions, rules that apply to those event classes, and predicates that are used by rules. A rule base on the IBM Tivoli Enterprise Console event server is the master container from which rule base targets are created. *Rule base targets* are the actual rule bases used by rule engines to process events.

Depending on the context of discussion, the term rule base and rule base target may be used interchangeably. You may have only one active rule engine managing events in a given IBM Tivoli Enterprise Console event server. If you have multiple layers of IBM Tivoli Enterprise Console event servers, which inherently means multiple TMRs, you may have multiple rule engines.

When there is more than one rule engine managing events in the environment, the rule bases used by the rule engines in that environment are referred to as *distributed rule bases*. In a distributed rule base environment, event classes and rules must be synchronized among all the rule engines. To keep these synchronized, all rule base development should be done from a central location and distributed out to all IBM Tivoli Enterprise Console event servers.

With the release of IBM Tivoli Enterprise Console 3.7 and later, we recommend that you use the **wrb** command for rule base manipulation procedures. This command provides more flexibility and function than the current rule builder available on the Tivoli desktop. Do not modify any files used internally by an event server to manage rule bases with a text editor. Use the **wrb** command or the rule builder to manipulate rule bases.

The rule compiler provided with release IBM Tivoli Enterprise Console 3.7 and later must be used to compile rule bases. Although an older rule base (pre-release 3.7) may appear to compile successfully with the newer compiler, the proper objects for a distributed rule base environment are not created. You must upgrade older rule bases before you use them in an IBM Tivoli Enterprise Console 3.7 or higher rule base environment.

### 3.4.5 Rule sets and rule packs

*Rule sets* are files that contain rules. Typically, related rules are contained within a rule set. Rule sets can be imported into a rule base target using the **wrb** command.

When a rule base is compiled, rule sets are replicated to rule base targets that specify which rule sets to import. When a rule base is used by a rule engine, generally the rules are processed in the order defined within a rule set and the order of how the rule sets were imported into the rule base target. You can alter regular rule processing order by using certain predicates within the rules.

**Note:** The order of rule sets defined for a rule base target is important, since it affects rule engine performance. Placement of the rule sets determine evaluation order by the rule engine. A starter set of rule sets is provided by Tivoli with the default rule base.

Another way to import rule sets into a rule base target is with rule packs. *Rule packs* are a convenient way to package a group of rule sets to be imported into a rule base target in a single operation. Rule packs are used to combine a group of rule sets that are used in multiple rule base targets. When a rule base is compiled, the rule base targets receive rule pack contents, which are rule sets.

Before you can import rule sets and rule packs into rule base targets, you must import them into the rule base on the IBM Tivoli Enterprise Console event server. Again, the rule base on the IBM Tivoli Enterprise Console event server is the master container for all of the objects that comprise rule base targets. There are multiple ways to import rule sets and rule packs into a rule base, using various options of the **wrb** command.

For example, you can use the following options:

- ▶ **cprb** option with the **–rulesets** and **–rulepacks** suboptions  
This copies an existing rule base into another rule base and copies the rule sets and rule packs from the source rule base.
- ▶ **crtrp** option with the **–import** suboption  
This creates a rule pack and imports rule sets into it.

- ▶ `imprprule` option  
This imports rule sets into an existing rule pack.
- ▶ `imprbrule` option  
This imports rule sets into a rule base.
- ▶ `-lsrbpack` option with the `-detailed` suboption  
Use this to list the rule packs in a rule base.
- ▶ `-lsrbrule` option  
Use this option to list the rule sets in a rule base.

You can also import rule sets into a rule base using the GUI Rule Builder.

### 3.4.6 State correlation

Chapter 8, “Writing rules for the state correlation engine”, in the *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9*, SC32-1234, is a great resource to learn about the details of the state correlation engine.

The *state correlation engine* is a correlation service that runs on an IBM Tivoli Enterprise Console gateway or adapter. It assists the IBM Tivoli Enterprise Console event server by running a correlation closer to the source of the event and reducing traffic by discarding, or consolidating events, that match the correlation rules.

The state correlation rules used are defined via Extensible Markup Language (XML) syntax, and not Prolog, unlike the event server rules engine. The correlation rules are defined from a Document Type Definition (DTD) file, named `tecscce.dtd` on the gateway or adapter. The default XML file is located in `$BINDIR/TME/TEC/default_sm/tecroot.xml`. There are sample XML rule files on the IBM Tivoli Enterprise Console Non-TME Installation CD in the directory `/EIFSDK/samples/state_correlation`.

**Note:** IBM Tivoli Enterprise Console administrators and rule writers will find that the new state correlation engine uses rules defined in an XML syntax. This is quite a change from prolog-based rules for the IBM Tivoli Enterprise Console event server. This should represent a streamlined approach to define rules for state correlation engines, using the latest technologies.

Machines in your environment that are used for state correlation (IBM Tivoli Enterprise Console gateways and adapters) are also known as *state machines*. Most state correlation rules are defined using state machines. A state machine gathers and summarizes information about a particular set of related events. It is

composed of states, transitions, summaries, and other characteristics, such as expiration timers and control flags.

There are five state-based rule types that are all based on state machines: duplicate, threshold, pass-through, resetOnMatch, and collector. Each state machine looks for a trigger event to start it. There is one stateless rule type—match.

A state-based rule operates on a sequence of events arriving at the state correlation engine, while a stateless rule operates on a single, current event. A rule can contain the following elements:

- ▶ Predicates for matching events relevant to that rule
- ▶ Actions that run after the rule triggers
- ▶ Attributes, such as a threshold limit

There are six rule types as explained in the following list. Each rule analyzes incoming events based on predicates that you specify. Depending on this analysis, each received event is either discarded by the rule or forwarded to the actions specified in the rule. If no action is specified, the events are forwarded to the gateway.

- ▶ **Duplicate:** The first event is forwarded, and all duplicate events are discarded. All the events that match the rule are forwarded.
- ▶ **Match:** All of the events that match the rule are forwarded.
- ▶ **Collector:** Events are collected for the time specified in the rule and then all events are sent individually to the server.
- ▶ **Threshold:** A single event is sent to the server when the threshold is reached.
- ▶ **Pass-through:** The events are sent to the server only if a specific set of subsequent events arrives during a time window.
- ▶ **Reset on match:** The events are sent to the server only if a specific set of subsequent events does not arrive during a time window.

**Note:** An event is analyzed according to the time it arrives at the state correlation engine, not at the time it originates. In some circumstances, this can affect the behavior of time-based rules, such as thresholding rules. For example, if the connection between an event source and the gateway is lost, events are cached at the adapter until the connection becomes available again. When the connection is restored, all of the cached events are sent at once, which may trigger a threshold rule unexpectedly.

See Chapter 6, “Event management products and best practices” on page 173, for examples of some of these rule types.



## Overview of IBM Tivoli NetView

This chapter discusses the purpose and capabilities of IBM Tivoli NetView distributed, a typical intermediate manager for IP networks. It also discusses the ability for NetView to provide information for other parts of a system's management environment such as the IBM Tivoli Enterprise Console or IBM Tivoli Business Systems Manager. The actual version of NetView, which is covered in this book, is Version 7.1, Release 7.1.4.

Although NetView runs under various UNIX platforms and under Microsoft Windows, we refer to the UNIX types of NetView. You can find a more detailed discussion about the various NetView distributions in Chapter 2, "Event management categories and best practices" on page 25.

The actual NetView documentation is distributed with NetView. You can find it in the Portable Document Format (PDF) under `/usr/OV/books/C/pdf` and in Hypertext Markup Language (HTML) format under `/usr/OV/books/C/html`. Updated versions of the documentation are located on the IBM Tivoli Web site at:

<http://www.ibm.com/software/tivoli/library>

**Tip:** Although the documentation is available on the NetView media, you need to install it in a separate installation step. To move the documentation into the mentioned directories, enter the following command:

```
nvinstal -K books
```

On UNIX and Linux platforms, man pages exist for most of the processes and main applications provided by NetView. Simply type:

```
man app_name
```

Here *app\_name* is the name of the process, configuration file, or application for which you are searching.

## 4.1 IBM Tivoli NetView (Integrated TCP/IP Services)

This section gives a brief overview of the capabilities and features of IBM Tivoli NetView (also known as *Integrated TCP/IP Services*) distributed. It also provides a short list of related documentation.

For more than a decade, IBM Tivoli NetView for distributed systems has been the major application from IBM to monitor and manage IP network resources in a distributed environment. It provides the capabilities shown in Figure 4-1, which we discuss briefly in this section.

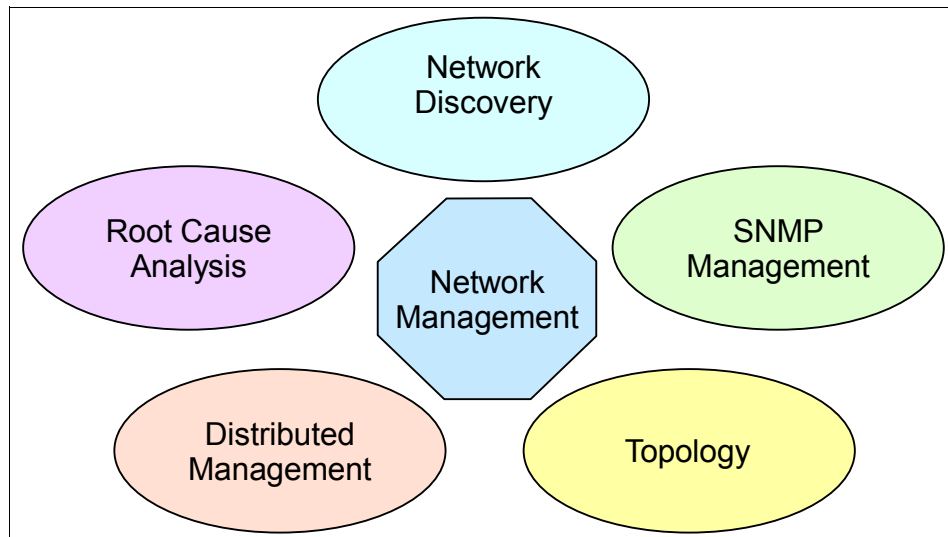


Figure 4-1 NetView's main capabilities

In addition, NetView can provide information to other management applications such as IBM Tivoli Enterprise Console.

NetView is distributed as part of the IBM Tivoli Enterprise Console under the component name *TEC Integrated TCP/IP Services* as well as the stand-alone product *IBM Tivoli NetView distributed*. Because the functions and capabilities of both distributions are mostly identical, we use the short name *NetView* throughout this document.

As a typical IP manager, NetView performs several tasks that are necessary to retrieve information from network resources and to manage them accordingly. These NetView capabilities include:

- ▶ **Network discovery:** NetView discovers the local IP network automatically and identifies the type and capabilities of the discovered resources. Discovery can be limited to a part of the network such as the local backbone or even the local subnet only. Through further configuration, you can include or exclude particular parts of the network into the discovery process. Also, you can configure NetView to limit discovery to a certain number of net devices such as servers and connection elements (router/switches). NetView then displays only these devices along with the network segments in which they reside.
- ▶ **Simple Network Management Protocol (SNMP) management:** NetView proactively manages network resources through the SNMP protocol as defined in RFC 1157. NetView covers both areas in SNMP management:
  - Retrieval of information by polling SNMP Management Information Base (MIB) entries from discrete network objects
  - Processing of asynchronous events in the network delivered as SNMP traps to the management system.
- ▶ **Topology display:** NetView show the network topology including network segments and their connecting routers, as well as other connecting resources such as hubs, switching devices, terminal servers, etc. Remember, NetView is considered an IP manager. It discovers your layer 3 (IP topology).
- ▶ **Distributed management:** This function accesses NetView management systems from various locations inside the network domain and across firewalls with the help of Java-based consoles.
- ▶ **Root cause analysis:** NetView automatically analyzes and detects the root cause of a network problem. This eliminates the generation of a high number of unwanted events and clearly points to the real cause of the problem.

Starting with NetView Version 7.1, NetView has become more tightly integrated with other management applications such as Tivoli Enterprise Console and IBM Tivoli Business Systems Manager. This extends its management flexibility with

the ability to provide both network management and, to a given extent, systems management and business systems management.

NetView 7.1.4 introduces more integration facilities to other Tivoli components such as the Tivoli Data Warehouse. We discuss these new features in 4.4, “Changes in NetView 7.1.3 and 7.1.4” on page 124.

In addition to its main tasks, NetView can act as an integration platform for a large number of third-party applications. An example is the popular suite of management applications available to manage Cisco devices.

In addition to its ability to manage IP networks, NetView can analyze and display ISO Layer 2 information, such as Switch Port assignments, status, and more, through its Web console. In conjunction with the IBM Tivoli Switch Analyzer companion product, NetView extends its root cause analysis capabilities down to Layer 2 in switched network environments.

For more information about IBM Tivoli NetView, refer to the following publications:

- ▶ *Tivoli NetView for UNIX User's Guide for Beginners, Version 7.1*, SC31-8891
- ▶ *Tivoli NetView for UNIX Administrator's Guide, Version 7.1*, SC31-8892
- ▶ *Tivoli NetView for UNIX Administrator's Reference, Version 7.1*, SC31-8893
- ▶ *Tivoli NetView Web Console User's Guide, Version 7.1*, SC31-8900

You can find publications about NetView, application programming interface (API) and programmer reference publications, on the Web at:

<http://www.ibm.com/software/sysmgmt/products/support/>

The IBM Redbook *Tivoli NetView 6.01 and Friends*, SG24-6019, covers most of the NetView Version 6 architecture and background information. For examples about how to interface NetView with other systems management applications or to integrate NetView into those applications, consult these IBM Redbooks:

- ▶ *Tivoli Web Solutions: Managing Web Services and Beyond*, SG24-6049
- ▶ *Tivoli Business Systems Manager Version 2.1: End-to-end Business Impact Management*, SG24-6610

## 4.2 NetView visualization components

This section discusses the various subcomponents of NetView and its main features. For the user, NetView consists of two parts:

- ▶ A GUI: Displays the actual state of the managed network
- ▶ An event console: Displays events either sent to NetView in the form of SNMP traps by managed objects or generated by NetView itself to signal changes in the status of the NetView managed objects.



You display the graphical environment by using the X Window-based native End User Interface (EUI).

In case access from a remote location is required or for information purposes, a Java-based console provides access to NetView topology views and to an event console. The Java console can be configured to show a subset of the actual data for a given user group or a specific part of the network

In addition to these directly visible components, the NetView product uses several additional subcomponents that carry out essential tasks to effectively manage complex networks.

### 4.2.1 The NetView EUI

The NetView EUI provides a graphical representation of the discovered network and the actual state of the managed elements. The EUI for the UNIX and Linux versions of NetView are based on X Window and should execute under most UNIX window managers. The EUI for NetView for Windows is based on the native Windows graphic environment. To differentiate between the X Window-based EUI and the NetView Web console, the X Window EUI is also called *native NetView console*.

Figure 4-2 shows a typical native NetView console. The native console consists of the following parts, which are launched when NetView is started:

- ▶ **Map view:** Represents the main EUI window. It displays the NetView topology information called *maps*. It also allows you to perform various activities and further configuration of NetView via a menu bar implementing menu tree that you can extend and customize.
- ▶ **Event console:** Displays, by default, all events generated by NetView itself, as well as the SNMP traps being sent to NetView by network elements. To receive traps from other network elements, you must configure the actual network device to forward traps to the NetView server.
- ▶ **Tool window:** Gives you several shortcuts to often used functions and a navigation window, which you can use to quickly select a submap in the NetView topology.

**Important:** In case you access NetView via a Windows workstation using a local X server, such as Exceed, an additional empty window named *dummy shell* is displayed. Do *not* close this window simply because it seems to be of no particular use. If you do, it closes all other NetView windows that you may have opened in your session.

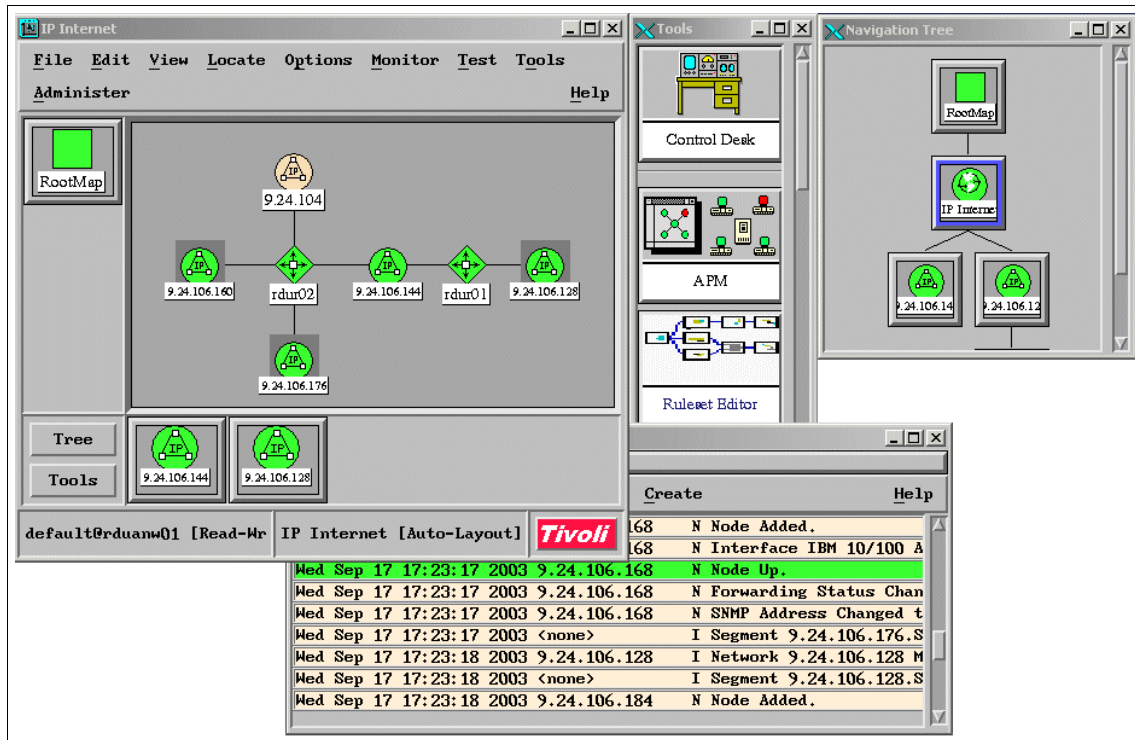


Figure 4-2 Typical NetView EUI components

NetView's menu system contains entries to access various configuration steps from within the NetView EUI. It also provides menu entries to several integrated functions, such as a MIB browser, a real-time graphing tool and other useful applications. You can access and launch most of those applications from a UNIX command line, but you have to provide all necessary parameters. When launching these applications from the NetView menu, the relevant information is provided by the EUI.

## 4.2.2 NetView maps and submaps

The central work area of the NetView EUI is the map window. NetView organizes its graphical views in submaps, which are related to each other.

It starts with a *root map* as shown in Figure 4-3. From the root map, you can select various representations of NetView managed elements. Additional applications that provide a graphic representation also place their root symbols on this map. By default, the root map contains three symbols:

- ▶ **Manager submap:** Contains all NetView servers visible to the selected NetView.
- ▶ **Smartset submap:** Contains all the default and custom smartsets.
- ▶ **IP Internet map:** Contains all elements of the discovered IP (Layer 3) topology.
- ▶ Arbitrary number of other root map entries depending on the number of third-party applications you may have integrated into NetView.

Figure 4-3 shows the NetView top-level map as it should appear right after the installation of NetView. All graphical views positioned below the root map are called a *submap*.

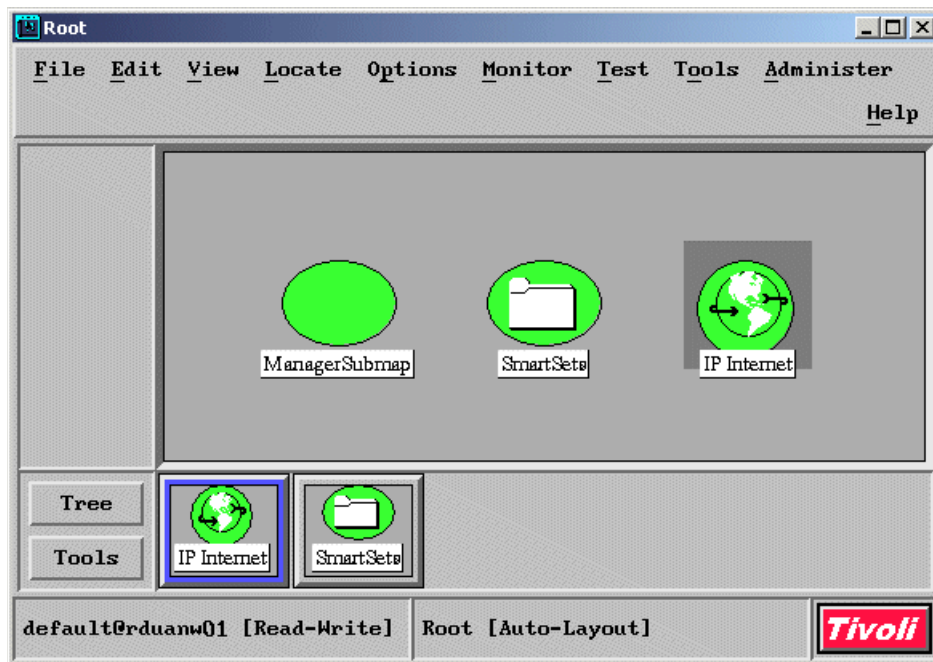


Figure 4-3 NetView root map display

If you click the **IP Internet** icon, the top-most level of the NetView topology map is displayed. Figure 4-4 shows the top level of our lab environment shortly after we set it up when we started to write this redbook. For a complete layout of the lab environment see Figure 7-1 on page 360.

The IP Internet submap represents the IP topology since NetView could discover it. On this map, only routers, location symbols, and IP segments are displayed since they make up the IP topology. In Figure 4-4, you can see two routers and several segments connected through the routers.

This submap also gives more information. If you examine the segment 192.168.111, notice that the connection line between the routers appears dotted, marking a serial connection. Also the color of that segment is magenta, unlike the other segments drawn. Magenta is the color for an administratively down segment.

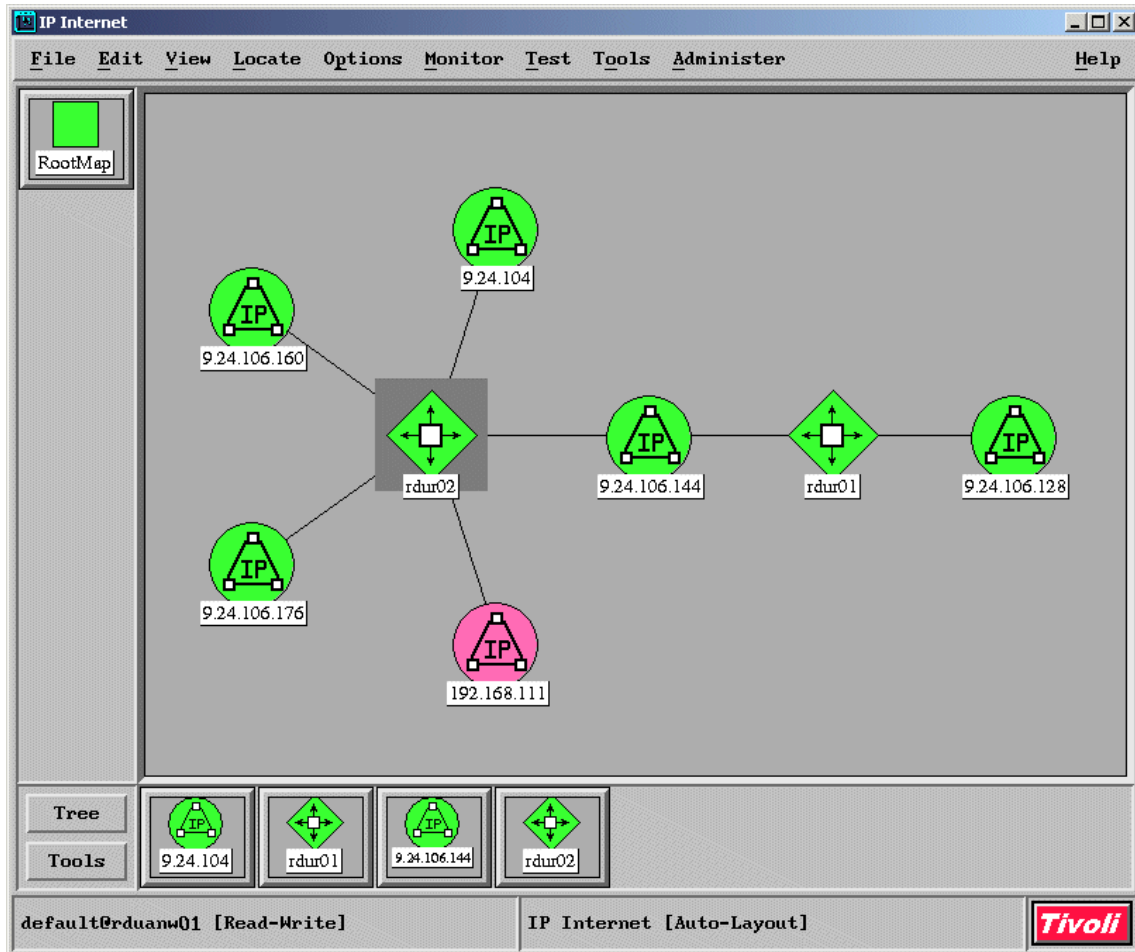


Figure 4-4 NetView top topology map

Our lab environment contains only a few elements, so that the IP Internet submap displays all the essential information without customization. As your network grows, the IP Internet will likely be crowded with elements, demanding more configuration. If the map becomes too crowded, you can move parts of the topology into location symbols. NetView connects these locations accordingly. To

demonstrate the location symbol, we moved the segments connected to router rdur02 and the router into a location called *NC\_net*. Figure 4-5 shows the result.

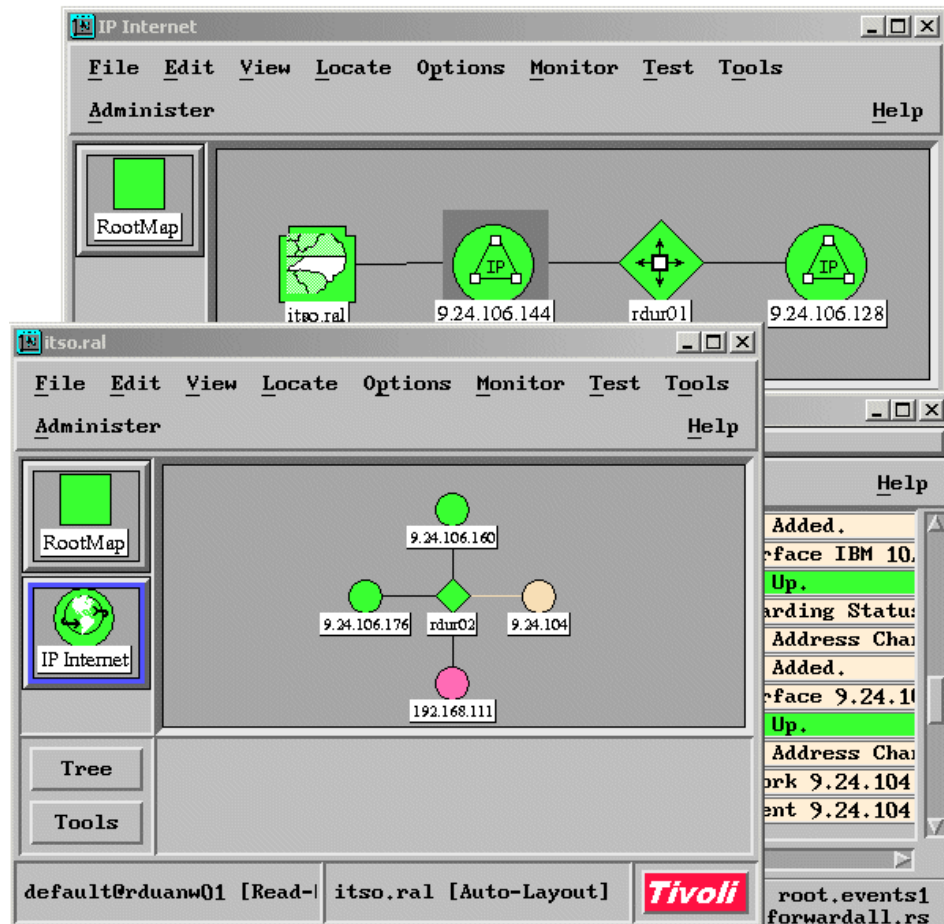


Figure 4-5 Using location symbols

You can use a location symbol to group your network. Even nested locations are supported. NetView automatically reconnects the locations to the correct connector elements. Only the IP Internet submap allows the position of location elements and assures the correct connection of elements.

All the submaps that are discussed don't accept custom elements or elements copied or moved from other submaps. NetView displays those elements but does not control or change them.

**Note:** With NetView version 6.0, a new configuration file was introduced. It allows you to set up a topology layout using location symbols. NetView uses the definitions in /usr/OV/conf/location.conf, creates the defined locations and their associated submap, and places the elements into the submaps only at discovery time. The supplied location.conf file contains comments about how to build a location layout.

The next lower level in the NetView topology is represented by the IP segment symbols. Clicking an IP segment symbol brings you to the connection submap. Figure 4-6 shows segment 9.24.106.144 of our lab environment.

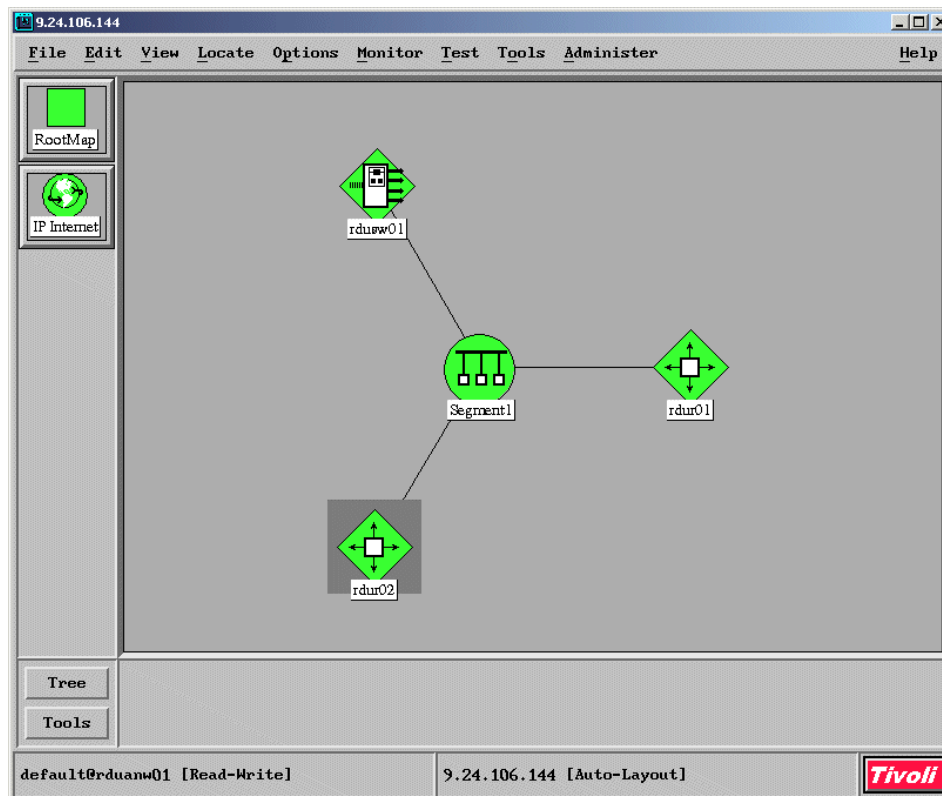


Figure 4-6 The connection submap

The connection submap contains a lot of useful information such as:

- ▶ The type of the underlying segment topology: In our case, it is an Ethernet bus segment. Other segment types displayed here are, for example, token ring or Fiber Distributed Data Interface (FDDI).

- ▶ Routers which have interfaces assigned to this IP segment: In our case, two routers are connected to the segment. You can verify this with the IP Internet submap as shown in Figure 4-4 on page 108.
- ▶ Any other element of type connector, for example Ethernet switches, terminal servers, or print servers displayed: In Figure 4-6, the Ethernet switch, which connects the NetView server to the lab environment, is correctly placed.

Double-clicking a segment symbol brings you to the segment level where all the discovered elements, which are part of the particular network segment, are placed. In addition to the connector type elements that we already know, you find all the remaining discovered elements such as printer, workstations, etc.

Figure 4-7 shows the segment map of our discussed topology. You can see one additional element, which is the NetView server residing in this segment.

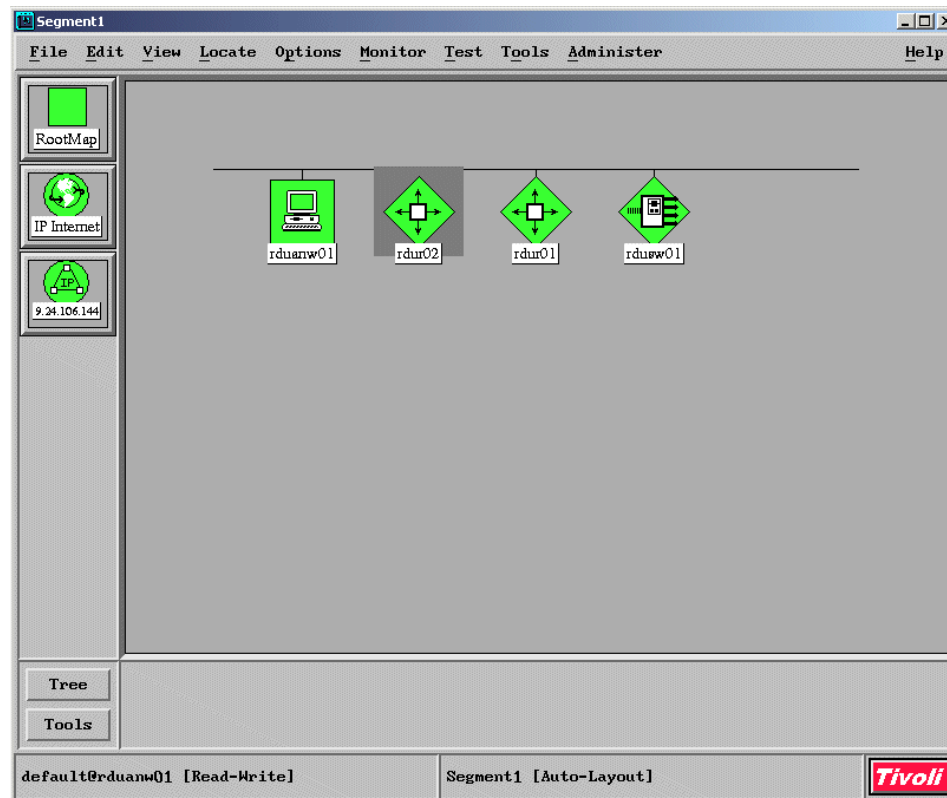


Figure 4-7 A segment submap

The segment map contains all network elements discovered and managed in the particular segment. It is the only submap where NetView places managed

elements that own a single IP interface and don't have a special topology-related meaning.

The lowest level of submaps that you will find in NetView is a representation of interfaces. You can access this submap from each higher level submap by clicking a real element or object. Figure 4-8 shows the interface submap of our router `rdur02`. It displays all the configured interfaces and their status. You can see three running, one unmanaged, and one interface, which is set to administrative down.

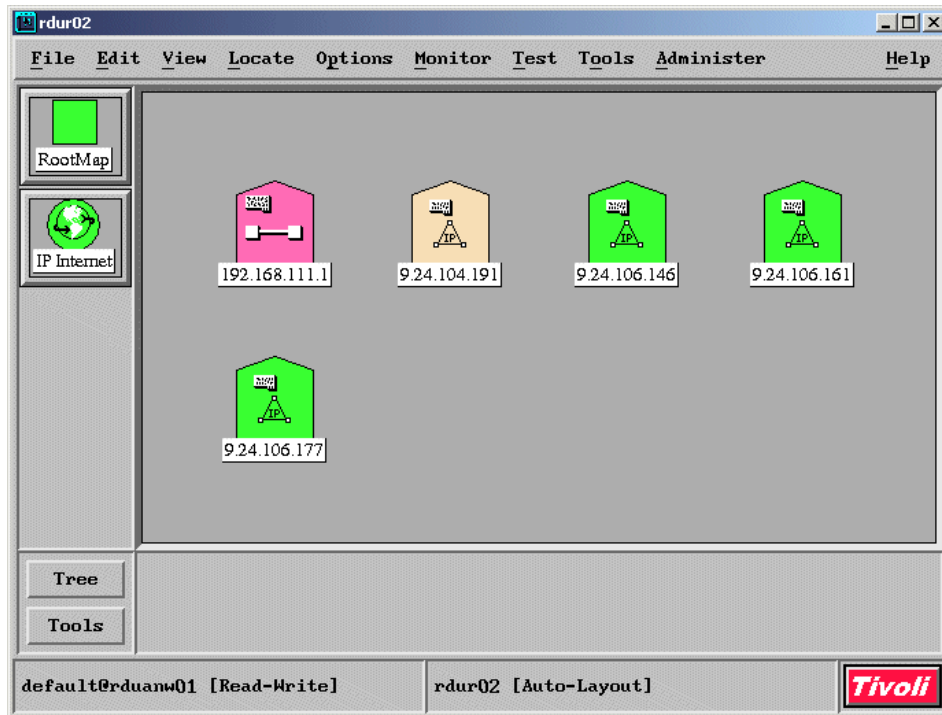


Figure 4-8 The interface submap

The change of interface status affects all other submaps. In case of a multi-interface object, such as an router, the change of an interface from up to down status changes or propagates the status up to the highest level. All upper level submaps change to yellow, which signals that at least one interface in the related submap chain is of status down.

### 4.2.3 The NetView event console

As discussed in 4.2.1, “The NetView EUI” on page 105, an event console is launched as part of the NetView EUI. However, the NetView configuration allows



you to change the appearance of the EUI. Often you find that the event console disconnected from the main NetView window. Refer to “Suggested NetView EUI configuration” on page 402, which discusses how to customize EUI behavior.

**Note:** We use the term *event* for both SNMP traps sent from remote SNMP agents to NetView and for messages that NetView generates as a result of a state change detected by polling a device. The messages triggered by a state change are in the same format as the SNMP traps received. Both types of message are an asynchronous event issued by NetView as a result of a (synchronous) polling operation and true asynchronous traps. NetView processes both types exactly the same way.

The NetView event console (Figure 4-9) displays all events arriving in the form of SNMP traps as well as those generated by NetView itself. By default, all arriving events that are not a log type are displayed only in the console.

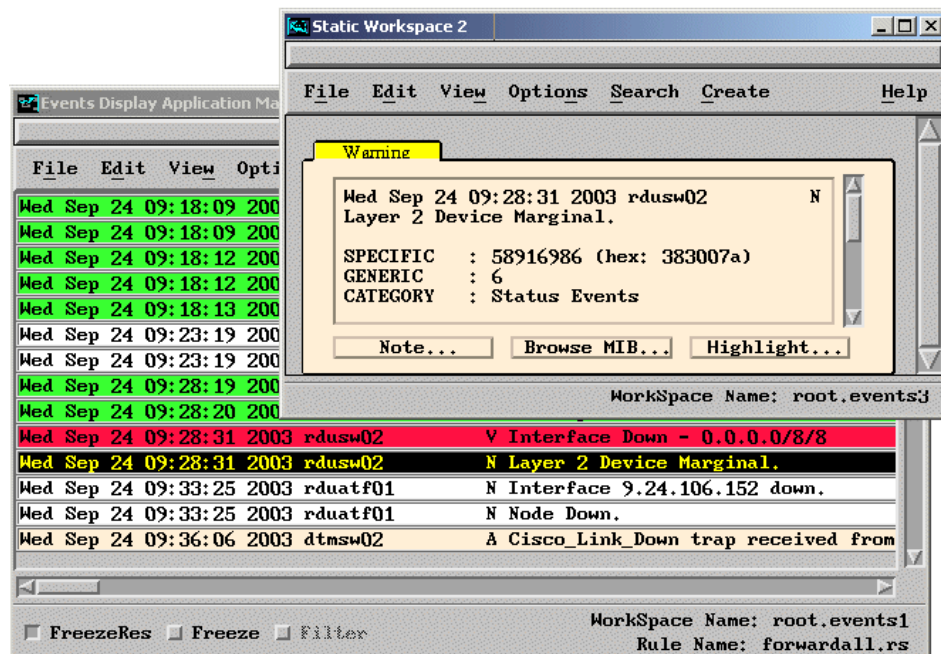


Figure 4-9 The NetView event console

The event console allows you to perform a few important tasks such as these:

- ▶ Switch the representation of events between *line mode* as shown in Figure 4-9 or *card mode* where the events are displayed in a card file manner.
- ▶ Select single events and retrieve all information for the event.

- ▶ Attach notes to specific events which are then included in reports that you can generate from a part or all events in the console.
- ▶ Create dynamic event consoles and apply filter definitions to this console to display only specific events.

You can configure the event console through a configuration file and tailor it to your needs. “Event console configuration” on page 403 lists examples of common event console configurations.

## 4.2.4 The NetView Web console

**Note:** Before you launch a Web console, you must configure at least one user using the NetView native interface. See “Web console security” on page 407.

The new Java-based NetView client is shown in Figure 4-10, enables you to access NetView from anywhere within your network. You can access it with a stand-alone application that you can download using NetView’s built-in Web server. Or you can use a Java plug-in for your Web browser. Access to NetView through the Web console is granted on a per-user basis. You can assign each user a definable role giving more or less restricted access to NetView, as well as a specific view and a limit on access to the graphical information.

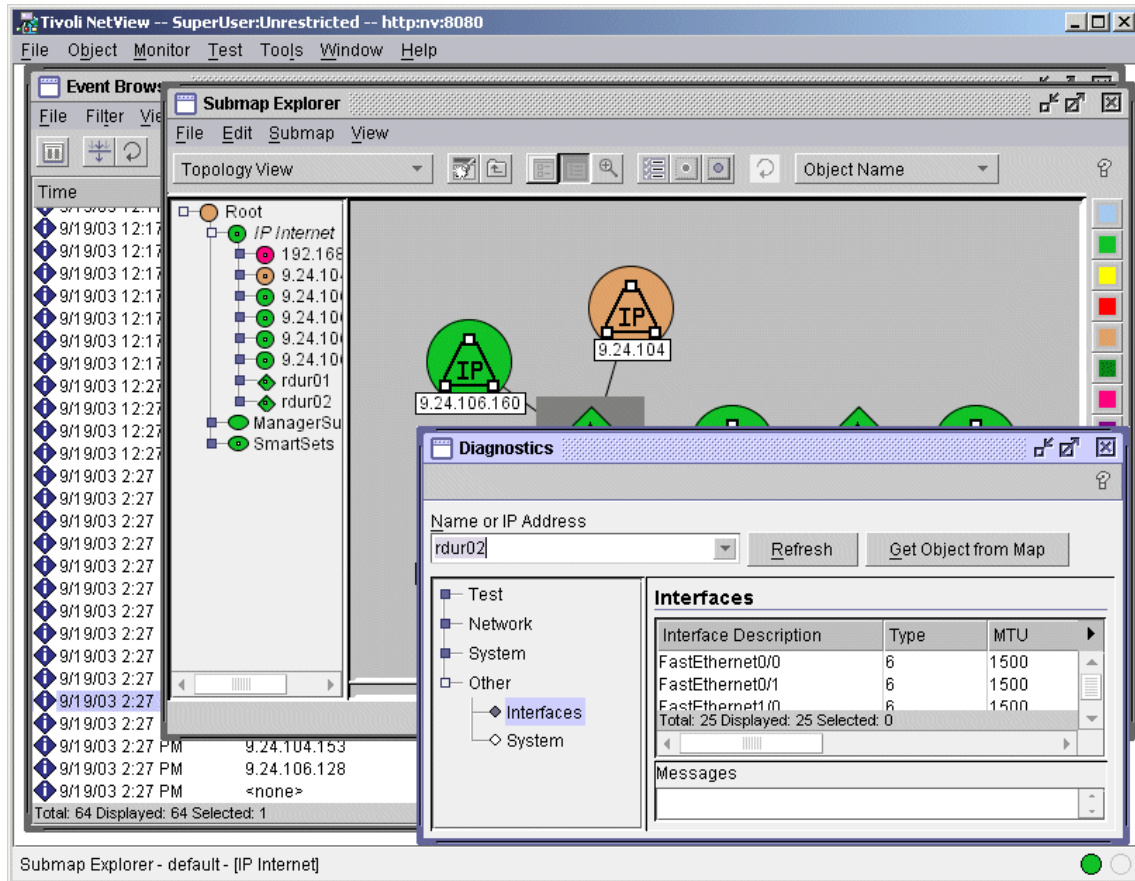


Figure 4-10 NetView Web console

The Web console provides graphical information about node status, as well as:

- ▶ An event browser that displays incoming events according to the actual user role and view
- ▶ An MIB browser that can retrieve MIB variables from all objects in your network
- ▶ A set of commonly used SNMP or IP-based diagnostics ranging from connectivity tests to limited systems management operations such as file system monitors
- ▶ Basic management capabilities, such as manage or unmanage an object or acknowledge or unacknowledge a certain status of an object

Any changed status is accordingly propagated among all working instances of the NetView console.

- ▶ Several Layer 2 switch diagnostics are not available with the NetView native console. These include:
  - Port view (Figure 4-11) which displays switch port along with its current spanning tree status, Media Access Control (MAC) address, forwarding status and the IP address connected to the port
  - A switch status view, which summarizes the spanning tree and IP status of each port
  - A MAC view, which shows the relationship between the MAC address, IP address, and host name for each port
  - A list of inactive ports

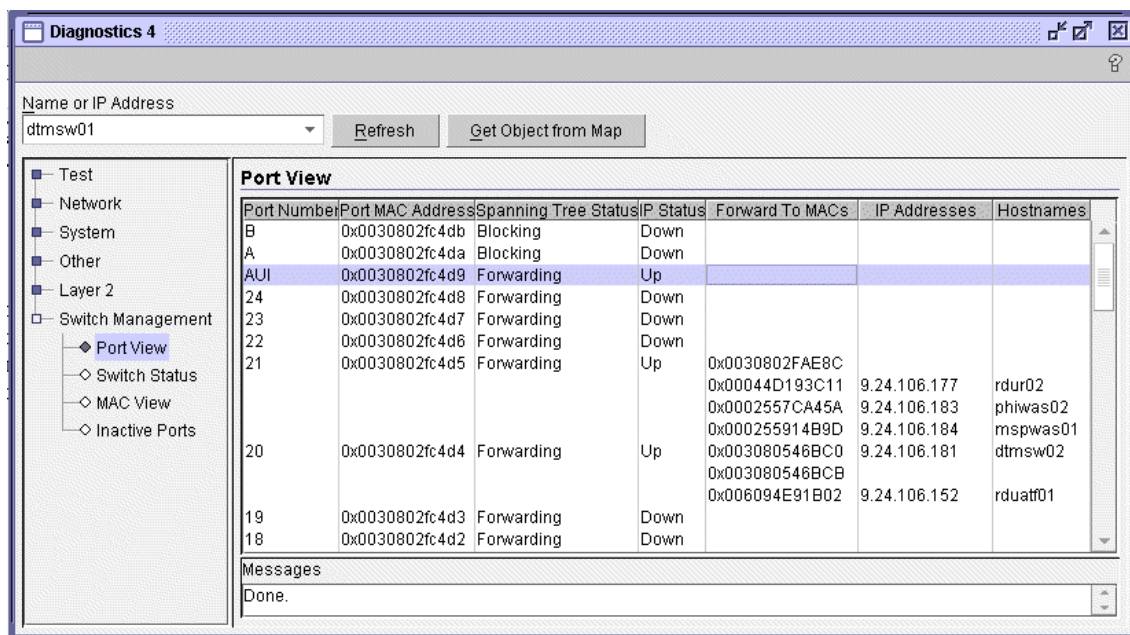


Figure 4-11 Port view of NetView switch diagnostics

- ▶ The same menu extension as for the native EUI if IBM Tivoli Switch Analyzer is installed

You can perform an impact analysis on the connector objects (router and switches) and rediscover the layer 2 topology used by the IBM Tivoli Switch Analyzer.

- ▶ Ability to extend the Web console menu and add your own applications to the menu

Results of custom menu entries can be displayed only in a Web browser.

## 4.2.5 Smartsets

Smartsets or collections were introduced in NetView version 4 and became more and more important. In general, a smartset displays objects stored in NetView's object database and groups them based on rules in a separate submap. Some NetView components create smartsets automatically, for example the servmon application. You can access the smartsets via a separate set of submaps via the NetView root map as shown in Figure 4-12.

In Figure 4-12, the smartset contains all the switches in our lab environment. The objects inside a smartset are not connected in any way. The submap simply collects NetView managed objects based on the rule that is specified.

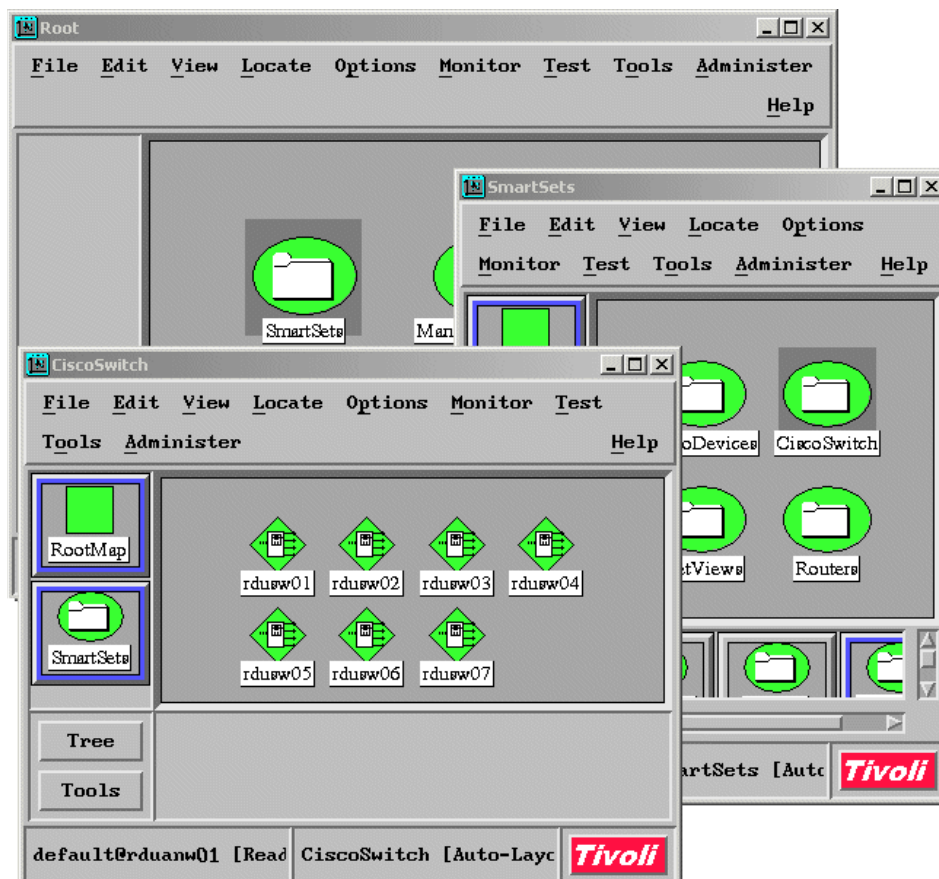


Figure 4-12 NetView smartsets

The advantages of smartsets are:

- ▶ They collect objects with identical attributes.
- ▶ They allow you to quickly check the status of a group of managed objects sharing the same attributes. For example, a smartset containing the routers in your environment can be used as an overview window to see the actual state of the routers without traversing a complex topology.
- ▶ Smartset rules allow the definition of dynamic smartsets. For example, NetView administrators often define a smartset, which contains all servers marked as down by NetView. The smartset contains only the critical servers. As soon as an object becomes available again, it is automatically removed.

You can define your own smartsets. A *smartset editor* is provided to help you with the definition. You can access the smartset editor via the native console menu. Select **Tools** → **SmartSet Editor...** from the NetView menu. This opens the SmartSet Editor window shown in Figure 4-13.

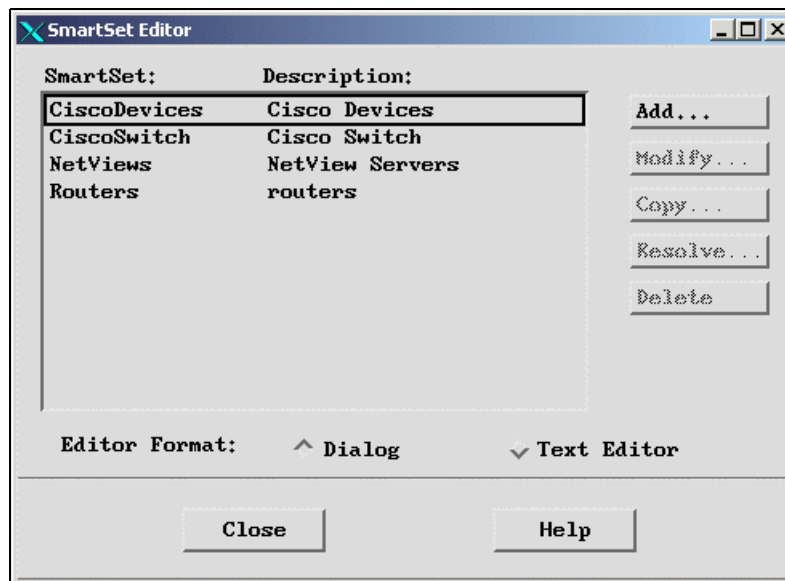


Figure 4-13 SmartSet Editor

In this window, you can add, modify, and remove smartsets. The smartset editor allows you to define simple rules using an interactive window. For more complex rules, when the rule exceeds four expressions, the smartset editor automatically displays the rule in a text only format.

To define a rule, the smartset editor offers several features:

- ▶ **Boolean operators:** Used to connect different parts of the rule in the form `expression_1 && expression 2 || expression 3`. Supported boolean operators are and (&&), or (||), and not (!).
- ▶ **Comparison operators:** Used to compare the contents of an attribute with a constant or a general expression. Supported comparison operators are: = (equal), != (not equal), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), ~ (contains), and !~ (does not contain).
- ▶ **Subnets:** Specifies all objects in a single subnet. For example `IN_SUBNET 9.24.106.128` includes the objects `rdur01`, `rdusw01`, `rduatc01`, and `rduarm01` of our lab environment on the smartset.
- ▶ **Object list:** Lets you define a static list of single objects that you can select on a submap to be included into the smartset.
- ▶ **Attribute:** Lets you specify any attribute of an object and compare the contents with a constant expression. If this expression is true, the object is included into the smartset.
- ▶ **Smartset rule:** Can be used to include or exclude the objects defined by another existent smartset rule into the actual smartset.

“A smartset example” on page 417 shows an example of a dynamic smartset.

## 4.2.6 How events are processed

This section gives an overview over the relationship between event processing daemons in NetView. For an in-depth discussion about event processing inside NetView and how the different processes are involved in event processing, see *Tivoli NetView for UNIX User's Guide for Beginners, Version 7.1*, SC31-8891.

The main entry point for any SNMP trap and any NetView generated event is the well behaved *trapd daemon*. This daemon, or process, listens on port 161, which is the well-known port to receive SNMP traps. The trapd daemon communicates with several other NetView processes to distribute the events. Figure 4-14 shows the relationship between the processes in regard to event processing.

The process follows this sequence:

1. Trapd logs all incoming traps and events to `trapd.log`. After logging, the application can directly process the event. Trapd also sends events to:
  - *pmd*: The postmaster daemon which further distributes events to registered processes
  - *nvcorrd*: The NetView correlation daemon

2. *pmd* receives events from *trapd*, applies routing information, and passes the events to the *ovesmd daemon*. This daemon forwards events after filtering to registered applications and to the log agent *ovelmd*. *ovelmd* logs all events and stores them into the *ovevent.log*. This log is the only source of event information for dynamic and historical events. Events displayed in the NetView event console are taken from this log file.
3. *Nvcorr*d correlates and compares events with rules defined with the rule editor and stored as rule sets. Depending on the decisions made in the rule sets, information is passed to the *actionsvr daemon* and, if necessary, to the *nvpagerd daemon*. Also, *nvcorr*d passes events to *nvserverd*.
4. *nvserverd* forwards events to the IBM Tivoli Enterprise Console. A NetView rule set must be assigned to *nvserverd*, which allows filtering of unnecessary events before they are forwarded to IBM Tivoli Enterprise Console.

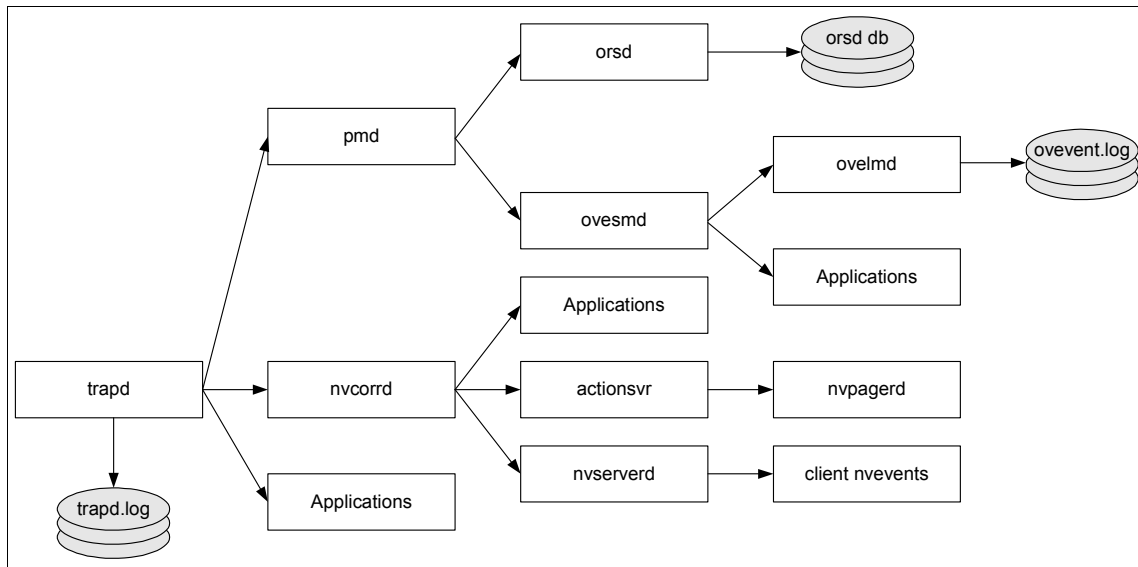


Figure 4-14 Process relationship for event and trap processing

## 4.3 Supported platforms and installation notes

This section describes the supported operating systems. It includes some installation notes that you should review prior to installing IBM Tivoli NetView.



### 4.3.1 Supported operating systems

The IBM Tivoli NetView distributed product is supported on several platforms:

- ▶ AIX®: V4.3.3 (Maintenance Level 09)
- ▶ AIX: V5.1 (Maintenance Level 01)
- ▶ AIX: V5.2
- ▶ Solaris: V2.8, V2.9 (with all Sun-required patches)
- ▶ Linux Intel®:
  - RedHat Version 7.2 (2.4.7-10 kernel)
  - RedHat Advanced Server Version 2.1
  - SuSE Version 7.2 (2.4.4-4GB kernel)
  - UnitedLinux Version 1.0
- ▶ Linux for zSeries®:
  - RedHat Version 7.2
  - SUSE LINUX Enterprise Server 7.2 for S/390® and IBM @server® zSeries (SLES 7)
  - UnitedLinux Version 1.0

NetView is not supported on 64-bit Linux for zSeries systems. The code is distributed on separate media for the UNIX, Linux, and zSeries platforms.

### 4.3.2 Java Runtime Environments

NetView requires Java Runtime Environment (JRE) 1.3.1 on the following platforms:

- ▶ AIX
- ▶ Solaris Operating Environment (hereafter referred to as Solaris)
- ▶ Linux Intel
- ▶ Linux for zSeries

### 4.3.3 AIX installation notes

NetView is supported on AIX Versions 4.3.3 maintenance Level 10 and 5.1 maintenance level 01.

In addition to the AIX operating system release and the required maintenance level, NetView depends on the number of additional system components which apply to both supported versions of the operating system:

- ▶ The X11.compat package
  - X11.compat.lib.X11R5
  - X11.compat.fnt.pc
- ▶ X11.vfb
- ▶ bos.compat.links

You must install these components before you install NetView. NetView performs a prerequisite check after the installation of the NetView catalog files, but before it starts to distribute the NetView images. In case the prerequisite check fails, you can look in /tmp/ NVS\_714\_BIN\_before.output to determine the failing packages.

After a successful installation, the NetView daemons list (see Example 4-1) should be running.

*Example 4-1 NetView processes overview*

---

```
Glasi-/usr/OV/conf >nvstat
```

NOTE: ovspmd is process 20210

DAEMON NAME	PROCESS	PARENT	DEPENDENCIES
actionsvr	22632	ovspmd	nvsecd,nvcorr
ems_log_agent	26316	ovspmd	nvsecd,ems_sieve_agent
ems_sieve_agent	25804	ovspmd	nvsecd,pmd,ovtopmd
mragentd	22486	ovspmd	nvsecd
netmon	23810	ovspmd	vsecd,ovtopmd,trapd,ovwdb
nvcol	25034	ovspmd	nvsecd,ovwdb
nvcorr	23516	ovspmd	nvsecd,trapd
nvlockd	23090	ovspmd	nvsecd
nvpagerd	19668	ovspmd	nvsecd
nvsecd	17592	ovspmd	
nvserverd	24348	ovspmd	nvsecd,nvcorr
ovactiond	24772	ovspmd	nvsecd,trapd
OVORS_M	21124	ovspmd	nvsecd,pmd
ovtopmd	25546	ovspmd	nvsecd,trapd,ovwdb
ovwdb	20504	ovspmd	nvsecd
pmd	22018	ovspmd	nvsecd
servmon	18070	ovspmd	ovwdb,trapd,nvcol
snmpCollect	26064	ovspmd	nvsecd,trapd,ovwdb,ovtopmd
snmpserver	20962	ovspmd	nvsecd
trapd	21700	ovspmd	nvsecd,pmd
trapgend	22832	ovspmd	nvsecd
webserver	23738	ovspmd	ovwdb

---

**Note:** The netviewd daemon is missing. This process provides a NetView map in case the NetView EUI is not active and must be registered to NetView. You can do this by typing:

```
/usr/OV/bin/ovaddobj /usr/OV/lrf/netviewd.lrf
```

### 4.3.4 Linux installation notes

Make sure that all required additional packages (in the following list) are installed before you begin the NetView installation:

- ▶ binutils
- ▶ inetd
- ▶ ucd-snmpd: Has different package names in various distributions
- ▶ xvfb: Named *XFree86-Xvfb* under Red Hat and *xextra* under SuSE

**Note:** If you decide to install NetView under a higher SuSE release than SuSE 7.2, be aware that the xextra package no longer exists. The frame buffer package, xvfb, which is required is still available.

In this case, you must switch off the prerequisite checking of nvinstall. To switch the prerequisite checking off, create an empty file under /tmp with the name *noNVPrereqCheck*.

- ▶ pdksh: Install from the NetView installation media, located under /RPMS
- ▶ The man pages don't integrate under SuSE. You must rename the files under /usr/OV/man. The man pages reside under various directories in /usr/OV/man, name man1 .... man 8. You need to add an extension to the man page files to reflect the subdirectory:
  - a. Change into the subdirectory, for example:

```
cd /usr/OV/man/man1
```
  - b. Rename the files

```
>for i in $(ls);do mv $i $i.1;done
```
  - c. Repeat steps a and b for all the manx directories.
  - d. Recreate the man page index database by typing:

```
mandb -c
```

## 4.4 Changes in NetView 7.1.3 and 7.1.4

This section presents an overview of the most significant enhancements introduced into NetView V7.1.3 and 7.1.4 for the scope of this book. You can find a complete list of enhancements beginning with the last major version (for example NetView V 7.1) and a list of actual fixes in the readme documentation.

### 4.4.1 New features and enhancements for Version 7.1.3

The new features and enhancements for Version 7.1.3 are outlined in the following sections.

#### **Web console enhancements: Menu integration**

The NetView EUI provides an interface to customize the NetView menu tree to integrate third-party or custom applications into NetView. These application are launched in a given context, for example based on a selected object on a NetView map.

Applications interface into NetView using application integration files (ARF) that are placed under `/usr/OV/registration/C`. All registration files in this path are read during NetView startup and linked to the appropriate parts of NetView.

For the Web console, NetView 7.1.3 introduces a similar, but limited mechanism. It is limited in that the Web console security concept does not allow a full integration into the Web console. The output of any application that you provide is always redirected into a Web browser. If a menu from the Web console is selected, a Uniform Resource Locator (URL) launch is initiated. This causes the integrated Web server to launch your application and passes the appropriate context information to the program. The context information is taken by the same mechanism as for the native menus.

Any program output is sent to a Web browser. We suggest that you format the output in the HTML format. Plain text output is also displayed as a Web page. In this case, it appears in different colors regardless of whether it is sent via the `stdout` or the `stderr` channel.

The configuration of the Web console menus take place with the help of Web Application Registration Files (WARF). The difference between ARF and WARF files is their format. While ARF files use a proprietary C style format to describe menus and their attributes, WARF files use standard Extensible Markup Language (XML) files to describe the menu behavior.

For an actual example of the Web console integration, see “Web console menu extension” on page 408.

## Secure Sockets Layer security

As an additional security measure, you can configure the Web console to use Secure Socket Layer (SSL) communication. You can configure SSL communication via the NetView server setup utility as explained here:

1. Stop all Web consoles.
2. Select **Configure** → **Configure Web Server** → **Enable Web Daemons**.
3. Set Enable Web Server daemon and Enable SSL for Secure Web Server Communications to **yes**.
4. Specify a port or keep the default port designation.
5. Start and stop the Web server daemon.

**Note:** After logging on to the Web console using port 8080 with SSL enabled, the session switches to port 8443 instead of port 8080.

## Revised Tivoli Enterprise Console Integration for Version 7.1.3

NetView Version 7.1.2 marks the beginning of a somewhat tighter integration into the IBM Tivoli Enterprise Console. When distributed with IBM Tivoli Enterprise Console, NetView becomes a subcomponent of IBM Tivoli Enterprise Console called Integrated TCP Service Component.

The integration of IBM Tivoli Enterprise Console and NetView is provided on both sides:

- ▶ An IBM Tivoli Enterprise Console rule set named `netview.rls` provides the necessary correlations for several essential network events. On the NetView side, a special NetView rule set, `tec_its.rs`, filters the relevant network events, formats them, and forwards the events to IBM Tivoli Enterprise Console.
- ▶ The process, `dispsub`, is loaded together with the NetView EUI. In case an IBM Tivoli Enterprise Console operator clicks a network event on the IBM Tivoli Enterprise Console, this action sends an SNMP trap with context information to the Integrated TCP Service Component. `Dispsub` accepts the information and launches a read-only submap on the IBM Tivoli Enterprise Console operators screen.
- ▶ Alternatively, you can use IBM Tivoli Enterprise Console to launch the Web console to give the IBM Tivoli Enterprise Console operator all the capabilities of a full blown Web console, including the Web console diagnosis.
- ▶ On the Integrated TCP Service Component side, a menu entry is added which allows a NetView operator to verify which IBM Tivoli Enterprise Console events are sent for a given node.

## 4.4.2 New features and enhancements for Version 7.1.4

This section discusses the enhancements made to NetView Version 7.1.4 since most apply to the scope of this redbook.

### Service discovery and monitoring

Up to NetView Version 7.1.3, a special application, *nvsniffer*, was used to discover TCP services running on managed nodes. After discovery of a service, *nvsniffer* could check the status of the service. Monitoring the status of services becomes more important and is part of a consistent event management.

Up to NetView Version 7.1.4, the *nvsniffer* utility was used to discover services on nodes managed by NetView. Actually, *nvsniffer* only checked the TCP port that you specified in its configuration for the presence of a listening process. If a process was detected on a well-known port *nvsniffer*, it assumed the service to be present. With the detected information about available services, *nvsniffer* created several smartset containing nodes on a per service base.

With NetView 7.1.4, the *nvsniffer* application is replaced by a new process called *service monitor*, which is implemented as the *nvservmon* daemon.

Part of managing a network requires that you know which nodes carry essential services for your enterprise. Often, the status of a node in a network management environment is determined on the status of its interface. A critical service on that node, such as a Web server, can be down without affecting the interface.

The new service monitor, *servmon*, provides a function to discover and monitor these services. It is set up via a configuration file where you specify both the discovery and the status monitoring method for a given service. The syntax of the configuration file is identical to the syntax of the *nvsniffer* configuration.

The difference between *nvsniffer* and *servmon* is that *servmon* has full integration into NetView. The server can be stopped, started, and configured via the NetView server setup. To can access the configuration page, you click **Administer** → **Server Setup** to launch the NetView server setup utility.

In general, *servmon* replaces and extends the functions of *nvsniffer*. Unlike *nvsniffer*, it can respond to topology status changes and changes in smartset membership. *servmon* detects traps received by *trapd* and messages generated by *nvcold*, enabling *servmon* to maintain internal discovery and monitoring lists.

The *servmon* daemon detects and acts on the following traps:

- ▶ Node Up
- ▶ Node Down

- ▶ Node Marginal
- ▶ Node Added
- ▶ Node Deleted
- ▶ Node Managed
- ▶ Node Unmanaged
- ▶ Interface Added
- ▶ Interface Deleted
- ▶ Service Removed (from a node)
- ▶ Service Managed
- ▶ Service Unmanaged
- ▶ Change Service Polling Intervals
- ▶ SNMP Address Changed
- ▶ Demand Poll (Forced Poll)

The servmon daemon detects and acts on the following messages from the nvcold daemon:

- ▶ Collection (SmartSet) Added
- ▶ Collection (SmartSet) Deleted
- ▶ Collection (SmartSet) Object Added
- ▶ Collection (SmartSet) Object Removed

Unlike nvsniffer, servmon acts on these events. For example, when a node down is received, this event is intercepted by servmon. Servmon checks if a service is monitored on that affected node and generates a service down event for that node as shown in Figure 4-15.

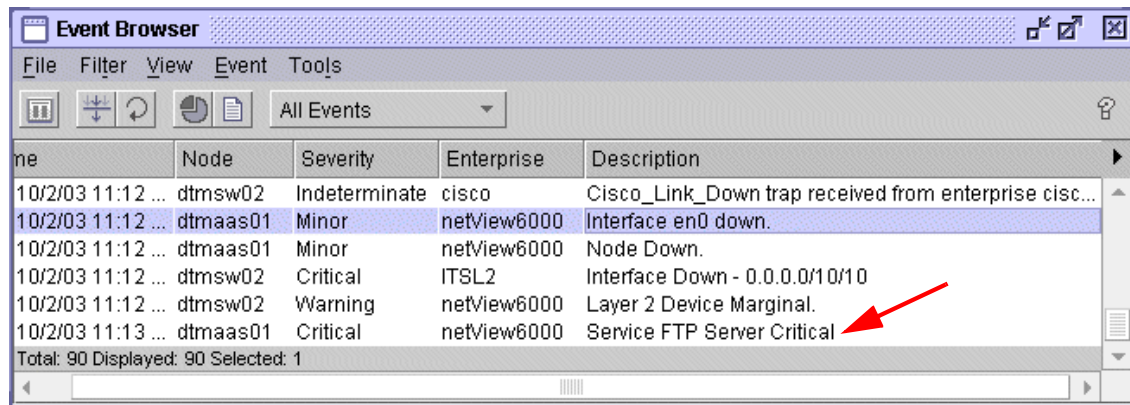


Figure 4-15 Service down event generated by servmon

Servmon also supports the new severity escalation for critical nodes. When a service running on a NetView managed object is detected, servmon adds a field

that describes this service to the NetView object database. Note the following fields:

- ▶ isITMEndpoint
- ▶ isITM\_IBM\_WebSphere\_Application\_Server
- ▶ isITM\_IBM\_WebSphere\_MQ
- ▶ isITM\_IBM\_DB2

If a service is missing on a given node for a certain amount of time (default is 7 days), the fields are removed. Then the fields are used for severity escalation, which we discuss in “Enabling Tivoli Enterprise Console event severity escalation” on page 129.

### **Revised Tivoli Enterprise Console integration for Version 7.1.4**

NetView now supports both the socket-based IBM Tivoli Enterprise Console communication as it was supported in former releases of NetView and, with NetView 7.1.4, the Tivoli communication method. You must install a Tivoli endpoint on your NetView server if you intend to use the Tivoli communication method when forwarding events to IBM Tivoli Enterprise Console. To specify the IBM Tivoli Enterprise Console communication, use:

- ▶ `nvinstal` with the `-T` flag for the Tivoli communication method or the `-t` flag for socket based IBM Tivoli Enterprise Console communication
- ▶ `nvits_config` script: The same flags apply as with the `nvinstal` utility.

From the NetView native console, click **Administer** → **Server** → **Configure** → **Configure event forwarding to Tivoli Enterprise Console**. In the window shown in Figure 4-16, complete the fields. If you select the Tivoli communication method, the window shows one or more Endpoint instance numbers. You must select one instance. Usually you see only one endpoint instance.



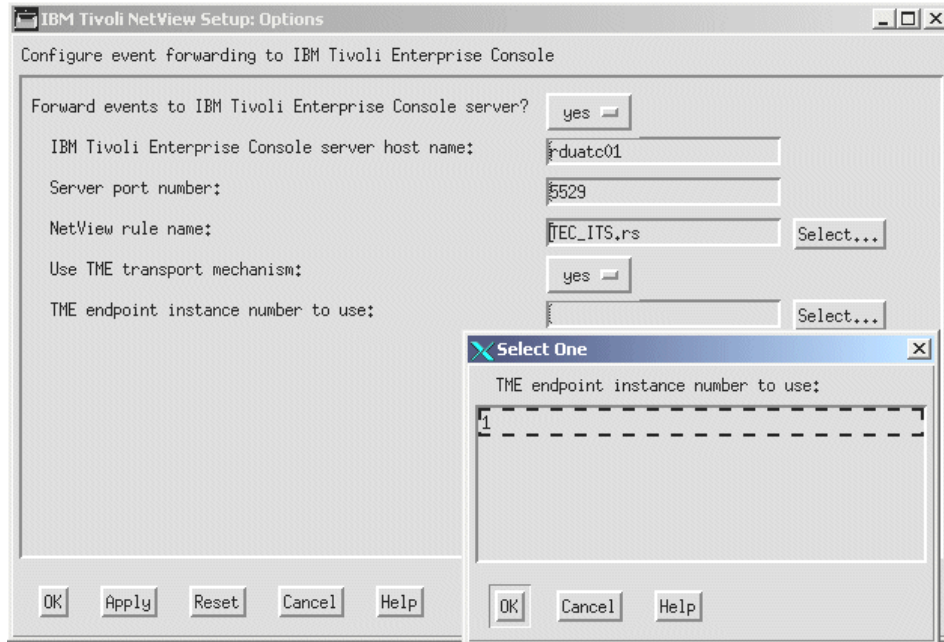


Figure 4-16 IBM Tivoli Enterprise Console configuration window

## Enabling Tivoli Enterprise Console event severity escalation

For nodes monitored by the IBM Tivoli Monitoring product, IBM Tivoli NetView can now query IBM Tivoli Monitoring servers for information that can help to escalate the severity of Tivoli Enterprise Console events for endpoint nodes that are monitored by IBM Tivoli Monitoring.

**Note:** The IBM Tivoli NetView documentation refers to an ITMQUERY function. However, the actual command is written in lowercase and can be executed via `/usr/ov/bin/itmquery`.

Using the ITMQUERY function, you can query IBM Tivoli Monitoring servers for information about IBM Tivoli Monitoring server endpoints and the services that are installed on these endpoints. ITMQUERY is also used to add and delete IBM Tivoli Monitoring servers monitoring nodes, which are detected by servmon. All configured IBM Tivoli Monitoring servers are queried for their nodes and their resource models.

This new connection between the network management instance and the monitoring instance is used by the IBM Tivoli Enterprise Console interface:

- ▶ The NetView IBM Tivoli Enterprise Console Adapter forwards two new events to IBM Tivoli Enterprise Console to allow correlation and an increase in severity for critical resources. The state correlation engine built in the NetView IBM Tivoli Enterprise Console adapter is used for this purpose.
- ▶ Upon receipt of a TEC\_ITS\_NODE\_STATUS or TEC\_ITS\_ROUTER\_STATUS event where the status is DOWN or MARGINAL and the affected node owns any "isITM\_\*" attributes, a TEC\_ITS\_NODE\_SERVICE\_IMPACT event is generated for each IBM Tivoli Monitoring service attribute found on the node. This includes the node host name and the affected service.
- ▶ Upon receipt of a TEC\_ITS\_SUBNET\_CONNECTIVITY where the status is UNREACHABLE and the affected nodes own one of the "isITM\_\*" attributes, a TEC\_ITS\_SUBNET\_SERVICE\_IMPACT event is generated for each affected IBM Tivoli Monitoring-related service.

### 4.4.3 First failure data capture

The first failure data capture (FFDC) function provides a means to collect data after the first failure of a Tivoli NetView daemon. The FFDC subsystem automatically collects and archives necessary data at the time of failure. You can use this information to analyze the problem, or you can send the data archive to IBM Customer Support.

The FFDC subsystem does not collect any data in case a daemon is stopped in a friendly manner using the **ovstop** command.

FFDC increases the tracing and logging activity and can affect your initial discovery. Because an initial discovery occurs rather seldom in a working environment, the overall impact of FFDC should be moderate.

**Note:** At the time this book was written, no particular information about the **atctl** command was available. In general, you use this command to control (start or stop) the FFDC subsystem.

To turn off the FFDC function permanently, use the following procedure:

1. Edit the `/usr/OV/conf/FFDC/autotrace.properties` file.
2. Set the following property to TRUE:

```
NV_AUTOTRACE_DISABLE=TRUE
```

Use the following procedure to turn off and on the FFDC function for the current Tivoli NetView session. The FFDC function remains off until the Tivoli NetView product is stopped and restarted.

1. Enter the following command to turn off tracing and check for residual tracing information:

```
/usr/OV/bin/atctl off NVD
```

2. Enter the following command to remove Tivoli NetView AutoTrace channel 60:

```
/usr/OV/bin/atctl remove 60
```

3. Enter the following command to remove Tivoli NetView AutoTrace channel 0:

```
/usr/OV/bin/atctl remove 0
```

4. Enter the following command to confirm that the Tivoli NetView AutoTrace function is stopped:

```
/usr/OV/bin/atctl info
```

As an alternative to these four steps, you can put everything in a script. Simply enter the following command to turn on the Tivoli NetView AutoTrace function:

```
/usr/OV/bin/atctl init /usr/OV/conf/FFDC/autotrace
```

FFDC operations are logged under `/usr/OV/log/FFDC.log`. Additional issues regarding FFDC are discussed in 4.5, “A closer look at the new functions” on page 131.

## 4.5 A closer look at the new functions

This section give a more in-depth look at some of the new feature provided with NetView 7.1.4 as they were used in our lab.

### 4.5.1 servmon daemon

The new servmon daemon provides a method for discovery and monitoring of important services. The servmon configuration file, `/usr/OV/conf/servmon.conf`, allows you to choose which services are important to your environment. The configuration file comes with the services defined in Example 4-2.

*Example 4-2 servmon configuration file excerpt*

---

```
#isITMEndpoint|||/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.DiscoveryMonitor|*|0

#isITM_IBM_WebSphere_Application_Server|WAS|||/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.DiscoveryMonitor|*|0

#isITM_IBM_WebSphere_MQ|MQ|||/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.DiscoveryMonitor|*|0
```

```
#isITM_IBM_DB2|DB2|||/usr/OV/jars/nv_itm.jar@com.tivoli.netview.itm.servmon.DiscoveryMonitor|*
|0

#isService_IBM_DB2|50000|IBM_DB2_Servers|IBM DB2 Server||*|20

#isService_IBM_WebSphere_MQ|1414|IBM_MQ_Servers|IBM MQ Server||*|20

#isRMON|1.3.6.1.2.1.16.1.1.1|RMON|RMON|/usr/OV/jars/snmpServiceTest.jar@com.tivoli.netview.sn
mpServiceTest.SnmpAttributeDiscoveryMonitor|/usr/OV/jars/snmpServiceTest.jar@com.tivoli.netview
.snmpServiceTest.SnmpAttributeDiscoveryMonitor|*|20

#isService_IBM_WebSphere_Application_Server|9090 9080 80 WAS|WebSphereServers|IBM WebSphere
Application
Server|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpDiscoveryMonit
or|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpDiscoveryMonitor|*
|20

#isService_IHS|80 IHS|IHS|IBM HTTP
Server|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpDiscoveryMonit
or|/usr/OV/jars/httpServiceTests.jar@com.tivoli.netview.httpServiceTests.HttpDiscoveryMonitor|*
|20
```

---

By default, each service that is defined is commented out. You must either remove the pound symbol (#) from the beginning of the service that you want to monitor or define a new service to monitor, using the syntax described in the servmon configuration file. Also note that the servmon daemon is enabled by default, although the default configuration does not monitor any services.

If you make customizations to the configuration file, and there are syntax or format errors in your configuration, the log file servmon.log is created under the /usr/OV/log directory. If errors are found with the configuration file, the service definition where the error or errors were found is ignored.

During our testing with servmon, we found that you need to stop and start the servmon daemon after you make any changes to the configuration file for those changes to take effect. After you define a service within the configuration file, and that service is found during the service discovery, a new smartset object is created for each service found, with the name specified in the configuration file.

Figure 4-17 shows a sample of the smartsets created using servmon. In this example, four new smartsets were created: Apache, IBM\_DB2\_Servers, IHS, and WebSphereServers.

You can view the status of services in two ways by default through the NetView console:

- Smartsets

When viewing smartsets, you can open one of the newly created servmon discovered services, and view which hosts are running that specified service. This view helps you to see the service status for all hosts for a given service.

- Hosts

When viewing the NetView map by hosts, you see the servmon discovered services under each host view that is running one or more of the defined and discovered services. This view helps you to see the status of all discovered services from a given host.

Servmon also gives you the ability to set the polling interval for service discovery and status. This polling interval is specified in the definition of the service in the configuration file. Another feature of servmon is its ability to stop monitoring down services. By default, if a service is down for seven days, servmon removes it from the list of discovered services on a per host basis. This option is also configurable from within the configuration file. You can also modify the polling interval and Service Down Delete Interval via the SNMP Configuration window.

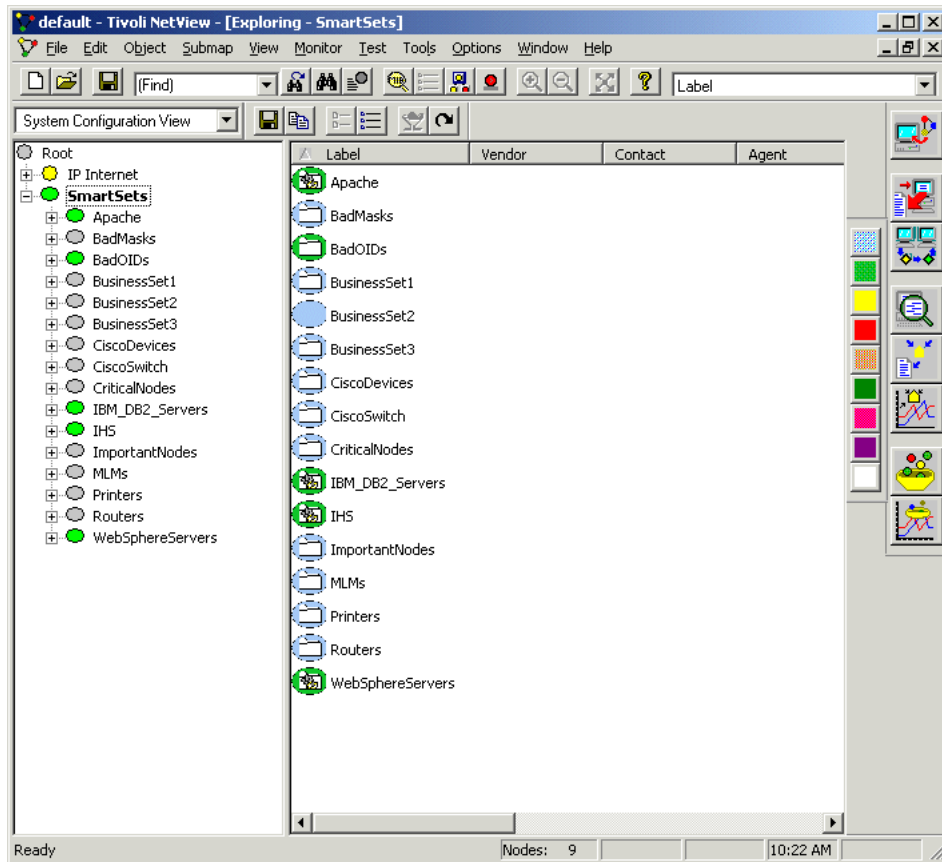


Figure 4-17 Servmon smartset after service discovery

## 4.5.2 FFDC

Example 4-3 shows the result of a short test in our lab environment. We stopped a few NetView processes to collect some FFDC data.

The FFDC subsystem writes the results of abnormal process terminations into /usr/OV/PD/FFDC. For each day, a directory is created which holds the data for that day.

#### Example 4-3 FFDC entries under /usr/OV/FFDC

```
Glasi/ >cd /usr/OV/PD/FFDC
Glasi/usr/OV/PD/FFDC >ls -l
total 24
drwxrwxrwx  2 root    system      512 Sep 22 16:07 2003.09.22
drwxrwxrwx  2 root    system      512 Sep 23 13:56 2003.09.23
drwxrwxrwx  2 root    system      512 Sep 25 08:15 2003.09.25
Glasi/usr/OV/PD/FFDC >
```

Inside the directories, you find a compressed tar file for each occurrence of an abnormal termination in the form *process\_name*FFDC\_ *num*.tar.*compressor* as in Example 4-4, where *process\_name* is the name of the crashed process, *num* is a consecutive numbering for that particular process, and *compressor* designates the compression program used. FFDC first looks for the gzip program. If it does not find the program, it uses the normal UNIX compress utility.

#### Example 4-4 FFDC per day repository

```
Glasi/usr/OV/PD/FFDC >cd 2003.09.25
Glasi/usr/OV/PD/FFDC/2003.09.25 >ls -l
total 15352
-rw-rw-rw-  1 root    system      1769363 Sep 25 08:11 netmonFFDC_0.tar.gz
-rw-rw-rw-  1 root    system      2007975 Sep 25 08:14 netmonFFDC_1.tar.gz
-rw-rw-rw-  1 root    system      1997498 Sep 25 08:15 nvserverdFFDC_0.tar.gz
-rw-rw-rw-  1 root    system      2080110 Sep 25 08:14 ovtopmdFFDC_0.tar.gz
Glasi/usr/OV/PD/FFDC/2003.09.25 >
```

The compressed file contains a tar archive, which contains all the collected data. The data differs from daemon to daemon, but in general, it contains the possible core files, important system and network information, and related log files at the time of the problem.

Example 4-5 shows the typical contents of a netmon FFDC archive.

#### Example 4-5 Typical contents of a netmon FFDC archive

```
Glasi:/usr/OV/PD/FFDC/2003.09.25 >tar -tvfnetmonFFDC_1.tar|more
drwxrwxrwx  0 0      0 Sep 25 08:14:52 2003 ./
-rw-rw-rw-  0 0    14179 Sep 25 08:11:59 2003 ./core
-rw-rw-rw-  0 0      86 Sep 25 08:14:51 2003 ./corefile.timestamp
-rw-rw-rw-  0 0  1117342 Sep 25 08:14:51 2003 ./errpt.out
-rw-----  0 0  15728896 Sep 25 08:14:50 2003 ./netmon_snap.at
-rw-rw-rw-  0 0     3479 Sep 25 08:14:51 2003 ./no_a.out
```

```

-rw-rw-rw- 0 0 14250 Sep 25 08:14:51 2003 ./ps_efl.out
-rw-rw-rw- 0 0 2410 Sep 25 08:14:51 2003 ./sysenv
-rw-rw-rw- 0 0 17766 Sep 25 08:14:52 2003 ./netview_root.log
-rw-rw-rw- 0 0 19336 Sep 25 08:14:52 2003 ./netview_daemon.log
-rw-rw-rw- 0 0 65168 Sep 25 08:14:52 2003 ./nettl.LOG00
-r--r--r-- 0 0 3473 Sep 25 08:14:52 2003 ./ovsuf
-r--r--r-- 0 0 303 Sep 25 08:14:52 2003 ./version
-rw-rw-rw- 0 0 52898 Sep 25 08:14:52 2003 ./netmon.trace
-rw-r--r-- 0 0 65919 Sep 25 08:14:52 2003 ./trapd.log
-rw-r--r-- 0 0 19293 Sep 25 08:14:52 2003 ./ipmap.log
-rw-r--r-- 0 0 2760 Sep 25 08:14:52 2003 ./netmon.conf
-rw-r--r-- 0 0 698 Sep 25 08:14:52 2003 ./communityNames.conf
Glasi:/usr/OV/PD/FFDC/2003.09.25 >

```

---

Files to be included into a FFDC archive are specified under `/usr/OV/conf/FFDC/scripts`. This directory contains a set of files and scripts that are commonly used and a set for each process being watched by FFDC as in Example 4-6.

*Example 4-6 Part of /usr/OV/conf/FFDC/scripts*

```

Glasi:/usr/OV/conf/FFDC/scripts >ls -l
total 208
-r-xr-xr-x 1 root system 12009 Aug 16 03:31 common_FFDC
-r--r--r-- 1 root system 207 Aug 16 03:31 common_FFDC.files
-r-xr-xr-x 1 root system 84 Aug 16 03:31 netmon_FFDC
-r--r--r-- 1 root system 182 Aug 16 03:31 netmon_FFDC.files
-r-xr-xr-x 1 root system 58 Aug 16 03:31 nvcolld_FFDC
-r-xr-xr-x 1 root system 13 Aug 16 03:31 nvcolld_FFDC.specific

```

---

You should not change the `common_FFDC` fileset. Perform any configuration, for example for directories, etc., in `/usr/OV/conf/FFDC/FFDC.properties`.

Each of the process specific filesets can consist of two or three files, which follow a common name convention. The files always begins with `process_name_FFDC`, where `process_name` is the name of the process:

- The script file being called by the FFDC control program is named `process_name_FFDC`. At the time this book was written, the scripts call the `common_FDDC` script with the name of the process as the only parameter.
- A file named `process_name_FFDC.files` lists, on a line-by-line base, the full path of any file to be included in addition to the information already included by the `common_FFDC` script.



- ▶ A file named *process\_name\_FFDC.specific* specifies specific processing for that particular process. For example, the results of a **netstat** command should be added to the archive.

## Integrating third-party processes

We now show an example of how to add another process into the FFDC subsystem. Processes that are being included need to fulfill these prerequisites:

- ▶ They must have an entry in the NetView ovsuf file under `/usr/OV/conf/ovsuf`.
- ▶ They must exist as a native executable. The processes started via the wrapper application `/usr/OVservice/spmsur`, in general the newer Java-based processes, did not integrate at the time this book was written.

In our example, we integrate the IBM Tivoli Switch Analyzer main process, coordinator, into the FFDC system. In addition to the default information, we include the IBM Tivoli Switch Analyzer configuration files, its log files, and its caches.

1. Make backup copies of all files that you are going to modify.
2. In case you want the change temporarily, modify the `itsl2` entry in `/usr/OV/suf`. Select the entry marked with 0 in position 1 as shown in Example 4-7. Locate the three consecutive colons at the end of the entry and replace the colons with the reference to a new FFDC script we will create. Example 4-7 shows how to make the entry.

### Example 4-7 The ITSL2 ovsuf entry

---

```
0:itsl2:/usr/OV/ITSL2/bin/coordinator:OVs_YES_START:ovtopmd,trapd,ovwdb::OVs_WELL_BEHAVED:30:/usr/OV/conf/FFDC/scripts/itsl2_FFDC:5:
```

---

3. If you want the integration to be permanent, you can make the same modification in the local registration file for the `itsl` process, `ITSL2.lrf`, located under `/usr/OV/lrf`.
4. Change to the `/usr/OV/conf/FFDC/scripts` directory. In this directory, using the editor of your choice, create two files named `itsl2_FFDC` and `itsl2_FFDC.files`. You can copy two files from the directory and modify them accordingly.
5. In the `itsl2_FFDC` file, add a line, as shown in the following example, to call the common script along with the constant `itsl2` as in Example 4-8. The `itsl2_FFDC` file is called from the FFDC subsystem in case the coordinator process ends in an unfriendly way.

```
/usr/OV/conf/FFDC/scripts/common_FFDC itsl2
```

6. Open or create itsl2\_FFDC.files and provide a list of process-specific information files. We included the IBM Tivoli Switch Analyzer configuration files and the cache files as in Example 4-8.

*Example 4-8 Contents of itsl2*

---

```
Glasi:/usr/OV/conf/FFDC/scripts >more itsl2_FFDC.files
/usr/OV/log/netmon.trace
/usr/OV/log/nettl.LOG00
/usr/OV/ITSL2/conf/cd_api.ini
/usr/OV/ITSL2/conf/coordinator.ini
/usr/OV/ITSL2/conf/correlator.ini
/usr/OV/ITSL2/conf/event_controller.ini
/usr/OV/ITSL2/conf/event_receiver.ini
/usr/OV/ITSL2/conf/ext_cmd_server.ini
/usr/OV/ITSL2/conf/ov_cmd_server.ini
/usr/OV/ITSL2/conf/ov_event_adapter.ini
/usr/OV/ITSL2/conf/poll_server.ini
/usr/OV/ITSL2/conf/topo_server.ini
/usr/OV/ITSL2/conf/cache/corr_cache
/usr/OV/ITSL2/conf/cache/corr_event_num
/usr/OV/ITSL2/conf/cache/event_cache
/usr/OV/ITSL2/conf/cache/raw_event_num
/usr/OV/ITSL2/conf/cache/topo_cache
/usr/OV/ITSL2/conf/cache/corr_cache
/usr/OV/ITSL2/conf/cache/corr_event_num
/usr/OV/ITSL2/conf/cache/event_cache
/usr/OV/ITSL2/conf/cache/raw_event_num
/usr/OV/ITSL2/conf/cache/topo_cache
Glasi:/usr/OV/conf/FFDC/scripts >
```

---

7. Test the file if you are allowed to end a process. Otherwise, trust the output shown in Example 4-9, which shows the results of a **kill -15** signal issued against the coordinator process of the IBM Tivoli Switch Analyzer.

*Example 4-9 Contents of the itsl2 FFDC archive*

---

```
Glasi:/usr/OV/PD/FFDC/2003.09.26 >ls
itsl2FFDC_0.tar.gz
Glasi:/usr/OV/PD/FFDC/2003.09.26 >
Glasi:/usr/OV/PD/FFDC/2003.09.26 >gzip -d itsl2FFDC_0.tar.gz
Glasi:/usr/OV/PD/FFDC/2003.09.26 >
Glasi:/usr/OV/PD/FFDC/2003.09.26 >tar -tvfitsl2FFDC_0.tar
drwxrwxrwx  0 0      0 Sep 26 10:27:27 2003 ./
-rw-rw-rw-  0 0      66 Sep 26 10:27:25 2003 ./corefile.timestamp
```

```

-rw-rw-rw- 0 0 1126460 Sep 26 10:27:26 2003 ./errpt.out
-rw----- 0 0 15728896 Sep 26 10:27:24 2003 ./itsl2_snap.at
-rw-rw-rw- 0 0 3479 Sep 26 10:27:26 2003 ./no_a.out
-rw-rw-rw- 0 0 10442 Sep 26 10:27:25 2003 ./ps_efl.out
-rw-rw-rw- 0 0 3020 Sep 26 10:27:26 2003 ./sysenv
-rw-rw-rw- 0 0 18462 Sep 26 10:27:26 2003 ./netview_root.log
-rw-rw-rw- 0 0 22501 Sep 26 10:27:26 2003 ./netview_daemon.log
-rw-rw-rw- 0 0 125128 Sep 26 10:27:27 2003 ./nettl.LOG00
-r--r--r-- 0 0 3732 Sep 26 10:27:27 2003 ./ovsuf
-r--r--r-- 0 0 303 Sep 26 10:27:27 2003 ./version
-rw-rw-rw- 0 0 53660 Sep 26 10:27:27 2003 ./netmon.trace
-rw-rw-rw- 0 0 181 Sep 26 10:27:27 2003 ./cd_api.ini
-rw-rw-rw- 0 0 1233 Sep 26 10:27:27 2003 ./coordinator.ini
-rw-rw-rw- 0 0 1005 Sep 26 10:27:27 2003 ./correlator.ini
-rw-rw-rw- 0 0 383 Sep 26 10:27:27 2003 ./event_controller.ini
-rw-rw-rw- 0 0 388 Sep 26 10:27:27 2003 ./event_receiver.ini
-rw-rw-rw- 0 0 404 Sep 26 10:27:27 2003 ./ext_cmd_server.ini
-rw-rw-rw- 0 0 327 Sep 26 10:27:27 2003 ./ov_cmd_server.ini
-rw-rw-rw- 0 0 335 Sep 26 10:27:27 2003 ./ov_event_adapter.ini
-rw-rw-rw- 0 0 355 Sep 26 10:27:27 2003 ./poll_server.ini
-rw-rw-rw- 0 0 433 Sep 26 10:27:27 2003 ./topo_server.ini
Glas:/usr/OV/PD/FFDC/2003.09.26 >

```

---

Integration of all other well behaved non-Java processes can be done in a similar manner. As mentioned, if you want to include files in the archive, which require some processing to produce it, you must specify the additional *process\_name*.specific file.





## Overview of IBM Tivoli Switch Analyzer

This chapter explains the seven layers of the Open Systems Interconnection (OSI) model and the features and functions of IBM Tivoli Switch Analyzer that provide network management capabilities at layer 2 of the model. It also describes the integration of IBM Tivoli Switch Analyzer and IBM Tivoli NetView, particularly in terms of their abilities to correlate layer 2 and layer 3 events to find the root cause of network problems.

## 5.1 The need for layer 2 network management

The need for layer 2 network management cannot be understood without first defining what is meant by the various network layers. This section describes the model used to define the network layers. It also explains why layer 3 management is insufficient to determine the root cause of network errors.

### 5.1.1 Open Systems Interconnection model

In 1984, the International Organization for Standardization (ISO) published the Open Systems Interconnection Basic Reference Model (ISO 7498) to provide an open standard for the communication between devices across a network. The OSI model, as it is usually called, defines seven sets of requirements called *layers* that govern all aspects of communication from the wire across which the signal travels (layer 1) to the application itself (layer 7). Figure 5-1 shows the layers of the model.

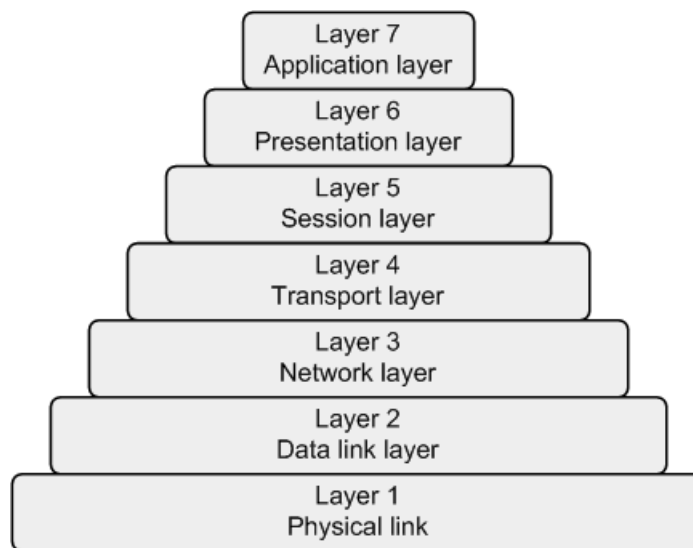


Figure 5-1 OSI model

Each layer of the OSI reference model provides a set of functions to the layer above it and, in turn, relies on the functions provided by the layer below it. The seven layers are in order:

1. **Physical layer:** This layer is responsible for physically transmitting data bits over the communication link. It provides the mechanical, electrical, functional, and procedural standards to access the wire or physical medium.

2. **Data link layer:** This layer provides the functions and protocols to transfer data between network entities and to detect (and possibly correct) errors that may occur in the physical layer. It transforms data between bit streams and frames. Switches and intelligent hubs operate at layer 2.
3. **Network layer:** This layer provides the means to establish connections between networks by determining the proper addressing and routing to use for the transmission. It translates between the network names used at the higher layers and Media Access Control (MAC) addresses used by the data link layer. It also provides routing information. Routers, and sometimes switches, operate at layer 3.
4. **Transport layer:** This layer provides transparent and reliable end-to-end data transfer. It does this by breaking large transmissions into consistently sized packets and providing flow control and error checking for those packets.
5. **Session layer:** This layer provides mechanisms to organize and structure interaction between applications, devices, or both. It handles the identification of the parties exchanging information and the rules they use for communication such as who speaks first and for how long.
6. **Presentation layer:** This layer is concerned with the representation of user or system data. This includes necessary formatting (for example, a printer control character) and code translation (for example, ASCII to EBCDIC) to present the data to the user interface.
7. **Application layer:** The user interface resides at the application layer. This layer gives the user access to all the lower OSI functions. An example of an application that provides a user interface is a Web browser.

Although messages can only pass vertically through the stack from layer to layer, from a logical point of view, each layer communicates directly with its peer layer on other nodes. For example, if the network layer adds routing information to a transmission, the device that receives the transmission must be able to decode the routing information. This holds true for every layer of the model.

An advantage of this layered approach is flexibility. Changes can be made to how products implement a single layer without affecting the remaining layers. In principle, as long as standard interfaces to the adjacent layers are adhered to, an implementation can still work.

### 5.1.2 Why layer 3 network management is not always sufficient

As discussed in Chapter 4, “Overview of IBM Tivoli NetView” on page 101, the NetView Integrated TCP/IP Services Component V7.1.4 product performs management at the third layer of the OSI Model. Its topology is based upon the routing information that is defined at that level and reported by the various devices in the network. During the discovery process, devices report their IP

routing information. NetView uses the information to build a map showing the layer 3 connectivity of devices. If a device has a direct route to a subnet through one of its interfaces, it is attached to that subnet on the map.

This type of management is useful in determining whether information can be routed between any two devices using IP. However, if IP routing is not possible, layer 3 management may not provide sufficient information to determine the cause of the failure. This is because the IP traffic is sent through the data link layer before it is passed across the network link to another device. A layer 2 failure may be the actual cause of the inaccessibility of a layer 3 device.

Consider the example of a layer 3 router that is attached to a layer 2 switch. If the switch experiences a failure of the port to which the router is attached, the router is inaccessible using IP, even though it may be running. In this case, the root cause problem is a broken switch port, not a failed router. However, sometimes the router is not accessible using IP because it really failed. In this case, the router failure is the real problem.

To determine root cause failures in these types of scenarios, it is necessary to know the layer 2 connectivity of the devices being managed. Both layer 2 and layer 3 failure conditions must be detected and correlated together to determine which failures to address. The purpose of IBM Tivoli Switch Analyzer V1.2.1 is to provide the layer 2 information necessary to make this determination.

## 5.2 Features of IBM Tivoli Switch Analyzer V1.2.1

The IBM Tivoli Switch Analyzer extends the ability of the IBM Tivoli NetView product to perform root cause analysis of layer 2 problems. Working with the IBM Tivoli Switch Analyzer, NetView can isolate problems to non-IP ports on managed layer 2 devices. Using IBM Tivoli Switch Analyzer, NetView users can view the port-level discovery of switches and view the impact of taking down a switch or port. Because the IBM Tivoli Switch Analyzer is tightly integrated with NetView, it is easy to use and maintain, and NetView operators can learn how to use it quickly.

### 5.2.1 Daemons and processes

The IBM Tivoli Switch Analyzer daemon, `itsl2`, is implemented as a well-behaved NetView daemon. Because it is a well-behaved daemon, it is started and stopped using the **ovstart** and **ovstop** commands.

When the daemon is started, seven other processes start. Table 5-1 lists the processes and their functions.



Table 5-1 *itsl2 daemon processes and their functions*

Process	Function
coordinator	<ul style="list-style-type: none"><li>▶ Starts and stops all IBM Tivoli Switch Analyzer components</li><li>▶ Routes messages between the components</li></ul>
correlator	<ul style="list-style-type: none"><li>▶ Receives raw events</li><li>▶ Correlates raw events to identify the root cause of a problem</li><li>▶ Processes polling requests</li><li>▶ Processes topology requests for layer 2 discovery</li></ul>
event_controller	Manages the event database
event_receiver	<ul style="list-style-type: none"><li>▶ Forwards events from the NetView event adapter to IBM Tivoli Switch Analyzer</li><li>▶ Starts and stops the OV and NV event adapters</li></ul>
ext_cmd_server	Provides the interface to send events to the NetView product
ov_cmd_server	Provides the interface to the NetView topology database
ov_event_adapter	<ul style="list-style-type: none"><li>▶ Receives raw events from the NetView trapd daemon</li><li>▶ Filters events and forwards them to the NetView event receiver</li></ul>

The status of these processes can be displayed by issuing this command from the /usr/OV/ITSL2/etc directory:

```
./ITSL2 status
```

You should see output similar to what is shown in Example 5-1.

Example 5-1 *Status of IBM Tivoli Switch Analyzer processes*

```
root:/usr/OV/ITSL2/etc >ITSL2 status
```

```
coordinator...      running
correlator...       running
event_controller... running
event_receiver...   running
ov_event_adapter... running
ov_cmd_server...    running
ext_cmd_server...   running
```

Each process has its own initialization (.ini) file in the /usr/OV/ITSL2/conf directory that contains parameters that control the execution of the process. These files define the name and size of the log file, to which the process writes

its messages, and ports used for communications with other processes. We discuss the flow of information among these processes later.

## 5.2.2 Discovery

Before layer 2 devices can be managed, they need to be discovered. For the purposes of this chapter, the term *discovery* is used in two different ways. NetView discovers layer 3 IP devices. IBM Tivoli Switch Analyzer discovers the port information for switches and how they interconnect with the rest of the network topology. Both of these discoveries need to happen before IBM Tivoli Switch Analyzer manages a layer 2 device.

After NetView discovers the switch from an IP layer 3 perspective, IBM Tivoli Switch Analyzer attempts to discover the addresses of devices attached to the switch's ports. If the device meets the requirements defined in Table 5-2, the discovery is successful, and the switch port information is used to complete IBM Tivoli Switch Analyzer's topology and correlate events from various network devices.

### Supported layer 2 devices

For a layer 2 device to be discovered using IBM Tivoli Switch Analyzer, the requirements listed in Table 5-2 must be met.

Table 5-2 Requirements for a layer 2 device to be discovered

Requirement	Explanation
NetView discovered itself using Simple Network Management Protocol (SNMP).	IBM Tivoli Switch Analyzer builds the layer 2 topology based on each discovered port's position (upstream or downstream) relative to the NetView machine. Therefore, it needs to know about the NetView box.
NetView discovered the device and is managing it.	IBM Tivoli Switch Analyzer only queries devices already discovered by NetView. The <code>ov_cmd_server</code> obtains this information by querying NetView's topology database. All devices downstream from an unmanaged device are ignored by IBM Tivoli Switch Analyzer.
The device's object identifier (OID) is defined in NetView and flagged as a bridge or hub.	IBM Tivoli Switch Analyzer discovers devices with OIDs flagged as bridges or hubs, but not routers in NetView. Use the following command to determine which devices NetView identified as switches:  <code>ovtopodump -X</code>
Device supports RFC 1493 bridge Management Information Bases (MIB).	IBM Tivoli Switch Analyzer queries the MIB for information about which devices are attached to the various ports. This information is supplied by the forwarding table that contains MAC addresses. Issue the following command to see if the MIB is supported:  <code>snmpwalk -c community string switch IP address dot1dBridge</code>

Requirement	Explanation
NetView has the correct community name defined for the device.	IBM Tivoli Switch Analyzer issues SNMP queries to the device using the community string defined in NetView. If this is incorrect, discovery will fail.
Non-Cisco device cannot require community string indexing (CSI) to access virtual local area network (VLAN) information.	IBM Tivoli Switch Analyzer 1.2.1 uses CSI to support Cisco switches with multiple VLANs. Other vendor switches are currently limited to discovery of VLAN1.
The network on which the switch resides is fully connected back to the NetView server.	Layer 3 connectivity is required from NetView to any managed switch. All routers and switches in this path must be SNMP enabled and the server must be configured with their community strings.

You can use the IBM Global Services *ITSATool* tool to determine IBM Tivoli Switch Analyzer's ability to discover ports for the various switches in an organizations environment. Contact your IBM Global Services representative for more information.

## The discovery process

At the start of the discovery process, IBM Tivoli Switch Analyzer downloads the complete NetView layer 3 network topology. Then it attempts to add the layer 2 information to it using the following procedure.

The IBM Tivoli Switch Analyzer installation process reads NetView's `/usr/OV/conf/oid_to_type` file. Then it extracts the object IDs for which the B, H, or BH flags are set. Next, it puts these OIDs in IBM Tivoli Switch Analyzer's `/usr/OV/ITSL2/conf/files/l2_oids.cfg` file. It ignores those entries for which the G flag is set. Table 5-3 lists the meanings of these flags in the `oid_to_type` file.

Table 5-3 Flag definitions for `oid_to_type` file

Flag	Description
B	Treat the object as a bridge or simple repeater. A symbol for objects with this OID appear in the Segment and Network views. The symbol can be used to connect segments.
G	Treat the device topologically as a gateway (router). A symbol for this object appears in the Segment, Network, and Internet views. The symbol can be used to connect networks.
H	Treat the object as a multi-port repeater or hub. A symbol for objects with this OID appear in the Segment and Network views. The symbol can be used to connect segments. This symbol can also appear at the center (hub) of Star segments.

IBM Tivoli Switch Analyzer reads the `l2_oids.cfg` file created by the installation process and attempts to discover the switch information from any devices NetView already discovered that have these OIDs.

**Notes:**

- ▶ If a new OID is added to NetView's `oid_to_type` file after IBM Tivoli Switch Analyzer is installed, the `l2_oids.cfg` file can be updated by executing the `/usr/OV/bin/importNvOids` command. The `itsl2` daemon must be recycled for the changes to take effect.
- ▶ To see the devices NetView discovered as switches, run the following command:  

```
ovtopodump -X
```

  
See “NetView layer 2 topology report” on page 149 for more information about this command.

IBM Tivoli Switch Analyzer uses the SNMP community name defined in the NetView SNMP Configuration panel to perform SNMP queries against the interfaces. It uses `dot1dBridge` branches of the standard MIB-II tree for the devices it is trying to discover. In particular, it sends a query using `oid.1.3.6.1.2.1.17.1.1` (`dot1dBridge.dot1dBase.dot1dBaseBridgeAddress`). The result returned should be a hex address.

If the SNMP agent on the switch does not provide a response to this MIB object, then discovery fails. Problems can be due to incorrect community names defined in NetView, incompletely discovered layer 3 path to the device, or switch configuration problems. See “Discovery problems” on page 158 for additional information about the causes of discovery failures.

For Cisco switches with VLANs defined, IBM Tivoli Switch Analyzer uses the community string index to query the device's Cisco-specific MIBs for information it can use to build its topology.

IBM Tivoli Switch Analyzer loads the switch topology it discovers into cache and writes it to the `/usr/OV/ITSL2/cache/topo_cache` file. Cache is updated every 15 minutes by default, as defined in the `topo_cache_freq` parameter of the `/usr/OV/ITSL2/conf/correlator.ini` file with information about new or successfully rediscovered switches. See “Rediscovery” on page 154 for more information.

**Discovery status**

IBM Tivoli Switch Analyzer attempts to discover the switch information for all NetView objects whose OIDs are defined as bridges or hubs, but not routers. There are a few methods to determine the success of the discovery process.

### ***The topo\_server log***

Errors encountered during discovery are written to the /usr/OV/ITSL2/log/topo\_server.log file. You can check this file to see which switches failed discovery and the reasons for the failure. Resolve these errors and attempt rediscovery for the switches. You can find more information about some of the errors that may appear in this file in “Discovery problems” on page 158.

### ***NetView layer 2 topology report***

You can run the following command either before or after installing IBM Tivoli Switch Analyzer. Its purposes is to show the devices that NetView believes are switches and their layer 2 status. To run the report, type:

```
/usr/OV/bin/ovtopodump -X
```

This command displays the following information about each node as shown in Figure 5-2:

- ▶ **OVwDb object ID:** A number assigned to the object in the NetView database
- ▶ **Node name:** The host name of the layer 2 device as discovered by NetView
- ▶ **IP status:** Indicates whether the IP address is reachable for the device
- ▶ **SNMP address:** The IP address to which SNMP commands are sent
- ▶ **sysObjectID:** The SNMP system object ID as reported to NetView during the discovery process
- ▶ **Layer2Status field value:** Reports up, down, and marginal status for layer 2 devices

This field is set and maintained by the Tivoli Switch Analyzer program for layer 2. The value of the field is initially set to Unset. When a problem occurs, the field is set to either Marginal or Down. While the problem is resolved, the Layer2Status field is updated to Marginal or Up. If a switch does not ever experience a problem, this field remains as Unset.

- ▶ **Layer 2 OID?:** Indicates whether the SNMP sysObjectId is missing from the /usr/OV/ITSL2/conf/files/l2\_oids.cfg file

Prior to installing IBM Tivoli Switch Analyzer, this field is No. After IBM Tivoli Switch Analyzer is installed, if this field is set to No, run the importNvOids command to synchronize the layer 2 object IDs between NetView and IBM Tivoli Switch Analyzer.

```

root:/ >ovtopodump -X
Node ID  Object      IP Status  L2 Status  IP Address      SNMP OID          Layer 2 OID?
OID?
  541    rdusw03    Up        Unset     9.24.106.163    1.3.6.1.4.1.9.5.18 Yes
  543    rdusw04    Up        Unset     9.24.106.164    1.3.6.1.4.1.9.5.31 Yes
  549    rdusw05    Up        Unset     9.24.106.179    1.3.6.1.4.1.9.5.18 Yes
  551    rdusw06    Up        Unset     9.24.106.180    1.3.6.1.4.1.9.5.18 Yes
  555    rdusw07    Up        Unset     9.24.106.181    1.3.6.1.4.1.9.5.31 Yes
  568    rdusw01    Up        Unset     9.24.106.131    1.3.6.1.4.1.9.5.18 Yes
  605    rdusw02    Up        Unset     9.24.106.147    1.3.6.1.4.1.9.5.31 Yes

```

Figure 5-2 Sample output from the ovtodump -X command

In addition to the information provided in the output, successful execution of this command implies:

- ▶ The machine on which Tivoli NetView is installed has been discovered.
- ▶ The capability field isConnector is set to True and the isRouter field is set to False for the devices listed.

**Layer 2 discovery reports**

Another way to determine discovery status is to run the IBM Tivoli Switch Analyzer supplied discovery reports. Two reports are provided:

- ▶ **Layer 2:** Lists each discovered switch and the discovered devices that are connected to each port on the switch. The report can be run for all discovered devices or for a specific layer 2 device.
- ▶ **Summary:** Lists the total number of switches discovered and their names.

Table 5-4 shows the commands to execute on the UNIX and Windows platforms to generate the reports.

Table 5-4 Commands to generate reports on UNIX and Windows

Platform	Layer 2 report command	Summary report command
UNIX	/usr/OV/ITSL2/bin/ITSL2_reports -r layer2 [-s device_name]	/usr/OV/ITSL2/bin/ITSL2_reports -r layer2
Windows	\usr\ov\itsl2\bin\itsl2_reports.bat -r layer2 [-s device_name]	\usr\ov\itsl2\bin\itsl2_reports.bat -r summary

The reports are generated from the information in IBM Tivoli Switch Analyzer’s cache. Each time the itsl2 daemon is restarted, IBM Tivoli Switch Analyzer rebuilds cache. It first waits the amount of time indicated by the topo\_cache\_freq period in the correlator.ini file (default is 15 minutes). Then it performs discovery

and updates its cache. Since the itsl2 daemon is typically started when NetView starts, this delay allows NetView to synchronize before IBM Tivoli Switch Analyzer begins the layer 2 discovery.

Running these reports before the daemon can discover the switches results in the message Layer 2 topology data is currently not available.

### ***Summary report***

The summary report (Figure 5-3) shows the switches that were discovered and their IP addresses. It was generated using the following command:

```
/usr/OV/ITSL2/bin/ITSL2_reports -summary
```

```
=====
                        Layer 2 Summary Report
=====

-----
Discovery has been completed for the following nodes:
-----

dtmsw01/9.24.106.180
dtmsw02/9.24.106.181
mspsw01/9.24.106.163
phisw01/9.24.106.179
rdusw01/9.24.106.131
rdusw02/9.24.106.147
sapsw01/9.24.106.164

=====
```

*Figure 5-3 Summary report*

### ***Layer 2 report***

The layer 2 report provides totals of the number of switches in the various stages of discovery: successful, in progress, and discovered with errors. It then shows the machine names and IP addresses of every device attached to the discovered switches.

In the following example, the layer 2 report was produced using the command:

```
/usr/OV/ITSL2/bin/ITSL2_reports -r layer2
```

The first switch listed is dtmsw01, which has IP address 9.24.106.80. Attached to it are:

- ▶ dtmwas01 with IP address 9.24.106.185 on port 7
- ▶ dtmsw02 with IP address 9.24.106.180 on port 20
- ▶ phisw01 with IP address 9.24.106.179 on port 21

Similarly, the devices attached to other switches are listed, along with their ports and IP addresses. Sometimes a port entry shows multiple addresses next to it. The first address represents the address through which NetView discovered the device. The second address is the one connected to a port.

In cases where a router is attached to a layer 2 device, these addresses may be different. See port 7 under rdusw01 in the example in Figure 5-4 through Figure 5-6 on page 154. The router, rdur01, was discovered in NetView through port 9.24.106.145, but it has multiple ports. The router port with address 9.24.106.130 is attached to switch port 7.

**Note:** You can generate the report for a single switch by using the command:

```
ITSL2_reports -r layer2 device-name
```

Or you can select the switch from the NetView topology map, and click **Monitor** → **Layer 2** → **Discovery**.

```
=====
                        Layer 2 Discovery Report
=====

-----
Number of layer 2 nodes discovered (no errors)      : 7
Number of layer 2 nodes with discovery in progress: 0
Number of layer 2 nodes discovered (with errors)   : 0

To view the summary report, run "ITSL2_reports -r summary"
-----

dtmsw01/9.24.106.180
  [20/20/0.0.0.0] ==>
    [dtmsw02/9.24.106.181 - 12/12/0.0.0.0]
  [21/21/0.0.0.0] ==>
    [phisw01/9.24.106.179 - 12/12/0.0.0.0]

dtmsw02/9.24.106.181
  [12/12/0.0.0.0] ==>
    [dtmsw01/9.24.106.180 - 20/20/0.0.0.0]

mbspw01/9.24.106.163
  [2 /2/0.0.0.0] ==>
    [rdur02/9.24.106.146 - FastEthernet0/1/2/9.24.106.161]
  [5 /5/0.0.0.0] ==>
    [phiwas01/9.24.106.167 - IBM 10/100 EtherJet PCI Adapter/0/9.24.106.167]
```

Figure 5-4 IBM Tivoli Switch Analyzer layer 2 report (Part 1 of 3)



```

[7 /7/0.0.0.0] ==>
    [sapsw01/9.24.106.164 - 7 /7/0.0.0.0]

phisw01/9.24.106.179
[2 /2/0.0.0.0] ==>
    [mspwas01/9.24.106.184 - AMD PCNET Family Ethernet Adapter/0/9.24.106.184]
[3 /3/0.0.0.0] ==>
    [phiwas02/9.24.106.183 - AMD PCNET Family Ethernet Adapter/0/9.24.106.183]
[5 /5/0.0.0.0] ==>
    [dtmaas01/9.24.106.186 - en0; Product: PCI Ethernet Adapter (23100020)
        Man/1/9.24.106.186] *
    [dtmwas01/9.24.106.185 - IBM 10/100 EtherJet PCI Adapter/0/9.24.106.185] *
    [rdur02/9.24.106.146 - FastEthernet1/0/3/9.24.106.177]
[12/12/0.0.0.0] ==>
    [dtmsw01/9.24.106.180 - 21/21/0.0.0.0]

rdusw01/9.24.106.131
[2 /2/0.0.0.0] ==>
    [rduarm01/9.24.106.136 - /0/9.24.106.136]
[3 /3/0.0.0.0] ==>
    [rduatc02/9.24.106.135 - en0; Product: PCI Ethernet Adapter (23100020)
        Ma/1/9.24.106.135]
[7 /7/0.0.0.0] ==>
    [rdur01/9.24.106.145 - FastEthernet0/0/1/9.24.106.130]

rdusw02/9.24.106.147
[2 /2/0.0.0.0] ==>
    [rdur01/9.24.106.145 - FastEthernet0/1/2/9.24.106.145]
[3 /3/0.0.0.0] ==>
    [rduatc01/9.24.106.153 - en0; Product: PCI Ethernet Adapter (23100020)
        Ma/1/9.24.106.153]
[5 /5/0.0.0.0] ==>
    [rdur02/9.24.106.146 - FastEthernet0/0/1/9.24.106.146]
[6 /6/0.0.0.0] ==>
    [rduanw01/9.24.106.154 - en0; Product: PCI Ethernet Adapter (23100020)
        Ma/1/9.24.106.154]
[9 /9/0.0.0.0] ==>
    [rduwsw01/9.24.106.151 - AMD PCNET Family Ethernet Adapter/0/9.24.106.151]
[10/10/0.0.0.0] ==>
    [rduatf01/9.24.106.152 - en0; Product: PCI Ethernet Adapter (23100020)
        Ma/1/9.24.106.152]

```

Figure 5-5 IBM Tivoli Switch Analyzer layer 2 report (Part 2 of 3)

```
sapsw01/9.24.106.164
  [2 /2/0.0.0.0] ==>
    [sapwas02/9.24.106.168 - IBM 10/100 EtherJet PCI Adapter/0/9.24.106.168]
  [7 /7/0.0.0.0] ==>
    [mbspw01/9.24.106.163 - 7 /7/0.0.0.0]
=====
```

Figure 5-6 IBM Tivoli Switch Analyzer layer 2 report (Part 3 of 3)

## Rediscovery

There are several ways in which IBM Tivoli Switch Analyzer rediscovers switches.

### ***Automatic rediscovery***

The itsl2 daemon performs a periodic discovery poll by default every 24 hours. The discovery poll forces a rediscovery of each switch it knows. This captures spanning tree and port connectivity changes for the switch. The user can modify the setting for the discovery interval value via the `discovery_interval` field (measured in minutes) in the `/usr/OV/ITSL2/conf/topo_server.ini` file.

### ***Retry attempts for failed discovery***

IBM Tivoli Switch Analyzer attempts to rediscover the devices that have OIDs listed in the `l2_oids.cfg` file and for which initial discovery failed. There are a couple of parameters that govern this rediscovery. These are defined in the `/usr/OV/ITSL2/conf/topo_server.ini` file. The parameters are:

- ▶ **`l2_retry_interval`**: The `l2_retry_interval` is the frequency at which failed layer 2 requests are retried. The default value is 900, which 15 minutes.
- ▶ **`l2_retry_cnt`**: The `l2_retry_cnt` is the maximum number of times a layer 2 request is retried. The default value is 5.

Using the default values, this means that IBM Tivoli Switch Analyzer attempts to discover a switch upon startup of the itsl2 daemon. If the discovery of the switch fails, IBM Tivoli Switch Analyzer waits 15 minutes and then tries to rediscover the switch, repeating this process up to 5 times. After the sixth attempt at discovery, if IBM Tivoli Switch Analyzer still has not discovered the switch, it stops trying to discover that particular switch. The operator must then manually rediscover the switch to reset the `l2_retry_cnt` to 0.

### ***Operator-initiated rediscovery***

To initiate a rediscovery, the user can restart the itsl2 daemon using the **`ovstop itsl2`** and **`ovstart itsl2`** commands. The daemon should be restarted if several switches must be rediscovered.

An operator can also initiate discovery at any time using the NetView Web console or native console. Use this method if there is only a small number of switches requiring rediscovery. Also use this method if the user has taken action to resolve the discovery problems recorded for the switches in the `/usr/OV/ITSL2/log/topo_server.log` file.

As shown in Figure 5-7, the user can select each switch that requires rediscovery in the applicable NetView submaps and click **Monitor** → **Layer 2** → **Rediscovery**. This triggers a rediscovery of only the selected switch, eliminating the overhead of performing a complete rediscovery. Note that only one switch can be selected at a time when initiating rediscovery through the NetView console.

If the rediscovery request succeeds for a switch, whether through automatic or operator-initiated discovery, its `l2_retry_cnt` is reset to 0 for that switch. Future layer 2 requests that fail for the switch are again retried up to the limit specified limit by the `l2_retry_cnt` parameter in `/usr/OV/ITSL2/conf/topo_server.ini`.

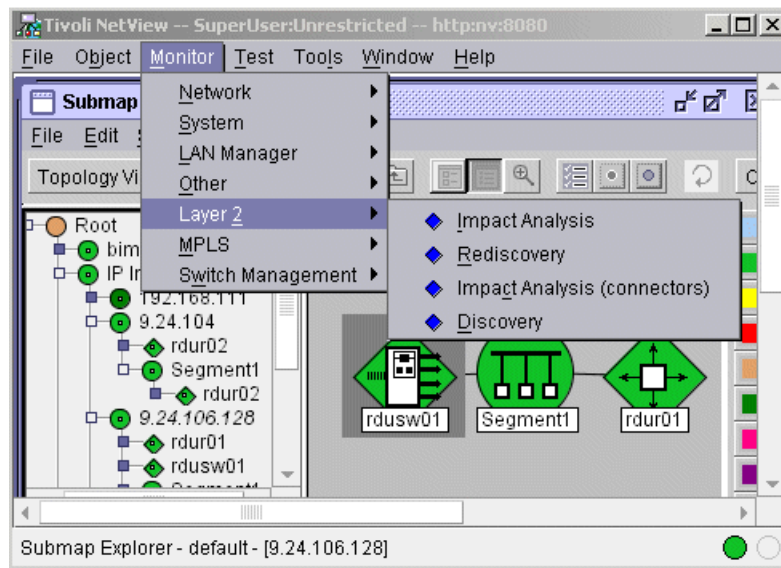


Figure 5-7 NetView console showing rediscovery menu option

## New switch discovery

When NetView discovers a new device, it issues a *Node added* trap to indicate that it added that node to its layer 3 topology. This trap is sent to IBM Tivoli Switch Analyzer.

If the new device's OID is contained in the `/usr/OV/ITSL2/conf/files/l2_oids.cfg` file, IBM Tivoli Switch Analyzer identifies it as a switch and performs the

necessary SNMP queries to discover it. The newly discovered switch is added to IBM Tivoli Switch Analyzer's cache during the next cache refresh, as identified by the `topo_cache_freq` parameter of the `correlator.ini` file.

### 5.2.3 Layer 2 status

IBM Tivoli Switch Analyzer provides the ability to check the layer 2 status of devices. This section describes what is involved with determining the layer 2 status and how to view the layer 2 status.

#### Determining status

IBM Tivoli Switch Analyzer is responsible for determining the status of layer 2 devices and passing this information to NetView for reflection in the topology map. It does this by processing Interface Down traps that it receives from NetView.

When IBM Tivoli Switch Analyzer receives a down trap, it first checks the device to verify that it is still down. This prevents IBM Tivoli Switch Analyzer from performing its analysis for down traps that are merely the result of missed responses to status queries.

Next, IBM Tivoli Switch Analyzer looks in its topology cache to determine the layer 2 device upstream (closer to the NetView machine) from the failing device. It polls it as well as peer devices for status. It continues in this manner until it finds the farthest upstream down device or interface.

After determining the farthest upstream failing device, IBM Tivoli Switch Analyzer uses correlation logic to determine the root cause of the original problem. It sends a root cause trap to NetView, and, if applicable, updates the `Layer2Status` field in NetView's topology database for the appropriate switch or switches.

#### ***Layer2Status field***

NetView comes with a topology database field called *Layer2Status* for tracking the layer 2 status of switches. This field is set and maintained by the Tivoli Switch Analyzer program for the layer 2 devices it recognizes.

The value of the field is initially set to Unset. When a problem occurs, the field is set to either Marginal or Down. When the problem is resolved, the `Layer2Status` field is updated to Marginal or Up. If a switch has not experienced a problem, its `Layer2Status` is Unset.

#### Displaying status

The Tivoli NetView program provides several methods to display this field:

- Tivoli NetView Web console

From within the Tivoli NetView Web console, select the device on the map, and select **Object Properties** → **Other** to display fields that provide the IP Status, Layer2Status, and status for each interface.

- Command line utilities

These two commands display the Layer2Status field for layer 2 devices:

```
/usr/OV/bin/ovobjprint -s selectionname
```

```
/usr/OV/bin/ovtopodump -l selectionname
```

*selectionname* is either the fully qualified IP name or the IP address if it cannot be resolved.

- Submap Explorer

You can use the System Configuration view to display the Layer2Status field and the IP Status field. If the status changes, this view is not dynamic and must be refreshed to show the change.

## 5.2.4 Integration into NetView's topology map

In addition to trap notification for layer 2 problems, the map symbol status for layer 2 devices is updated. Layer 2 connector devices are displayed in the Network and Segment submaps. Without Tivoli Switch Analyzer installed, the status of NetView devices is determined from the status of their interfaces. This means that the status of layer 2 device symbols depends on the status of a single management IP interface.

**Note:** On Windows, the status of services also contributes to the status of the layer 2 device symbol. When the Tivoli Switch Analyzer program is installed, the status of only the layer 2 device symbols is determined by either their IP status or layer 2 status, whichever is more severe. For example, if the management interface is Up, but the layer 2 status indicates that one or more ports have failed, then the status of the layer 2 device symbol is Marginal.

### *Tivoli NetView Web console*

You can use the Tivoli NetView Web console to display the status of a layer 2 device. Select the device on the map, and select **Object Properties**. Then on the Object Properties window (Figure 5-8), select **Other** in the left pane to display fields that provide the IP Status, Layer2Status, and status for each interface. On Windows platforms, it also displays all of the service objects.

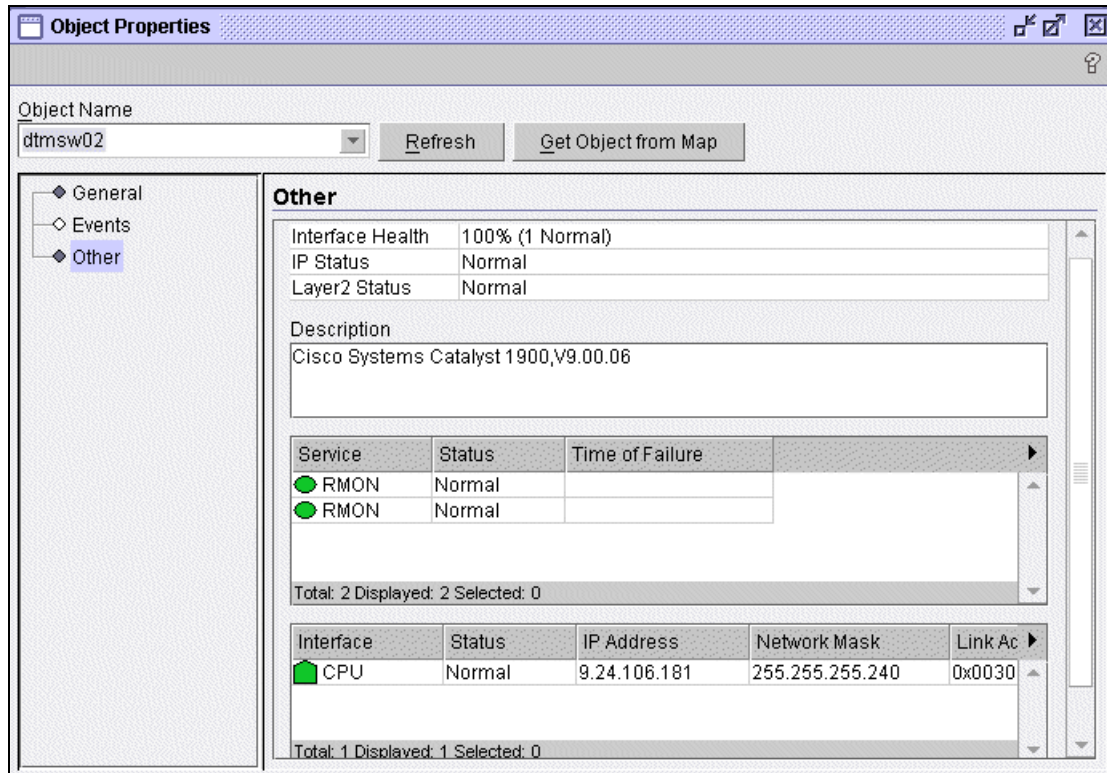


Figure 5-8 The Layer2 Status Dialog of the Web Console

## Discovery problems

IBM Tivoli Switch Analyzer does send an SNMP query to the switch that it is trying to discover. For this reason, it must have the correct community name. IBM Tivoli Switch Analyzer uses the community names as defined in netmon. If this community is wrong or not defined for a switch, you see the series of messages in the /usr/OV/ITSL2/log/topo\_server.log file (Example 5-2).

### Example 5-2 Messages in the /usr/OV/ITSL2/topo\_server.log file

```
"... start discovery process for node [10.20.30.40|<community name>]
... L2 ERROR for node [10.20.30.40]: SNMP request timed out
[.1.3.6.1.2.1.17.1.1.0]
... end discovery process for node [10.20.30.40]: 1
... L2 ERROR for request [n]: unable to discover layer 2 interfaces"
```

## 5.2.5 Traps

The Tivoli Switch Analyzer program issues ITSL2 Enterprise traps to notify the user of the correlated root cause and subsequent update events. These traps are displayed and handled by NetView.

When a trap is received, NetView issues IBM Tivoli NetView Enterprise layer 2 Status traps if it determines that the status of the device has changed as a result of this trap. These traps are used to update the status of the device on the NetView topology map.

There are three traps: Up, Marginal, and Down. You can use the NetView trap customization utility to define a notification method such as e-mail, pagers, and so on.

### Event browser

Traps can be displayed using native event display and the Web console event browser. The traps identify the root cause device in the hostname field.

The type of event and the failing ports are identified in the description field. The Source field for these traps contains the letter V for vendor.

### Forwarding traps to the IBM Tivoli Event Console

If the Tivoli NetView program is configured to forward traps to the IBM Tivoli Event Console, all IBM Tivoli Switch Analyzer traps and the NetView layer 2 Status traps are forwarded as TEC\_ITS events by default.

NetView layer 2 Status traps are mapped to TEC\_ITS\_L2\_STATUS events with a value of Up, Down, or Marginal. IBM Tivoli Switch Analyzer traps are mapped to TEC\_ITS\_SA\_STATUS events. The status is defined in the sastatus slot. See the “Synchronization and Correlation with Tivoli NetView” chapter in the IBM Tivoli Event Console Rule Builder’s Guide, Version 3.9 for more information about the NetView rules.

Figure 5-9 shows the traps added during the installation of IBM Tivoli Switch Analyzer. This information is placed into /usr/OV/tmp/itsl2/Install\_stdout.log on UNIX and in \usr\OV\tmp\itsl2\Install\_stdout.log on Windows.

```
Successful Completion
Done.

Adding Switch Analyzer traps...
Trap ITSL2_IF_DOWN has been added.
Trap ITSL2_NODE_DOWN has been added.
Trap ITSL2_NODE_MARGINAL has been added.
Trap ITSL2_UPD_IF_UP has been added.
Trap ITSL2_UPD_IF_UNMAN has been added.
Trap ITSL2_UPD_IF_DEL has been added.
Trap ITSL2_UPD_NODE_UP has been added.
Trap ITSL2_UPD_NODE_UNMAN has been added
Trap ITSL2_UPD_NODE_RESOLVED has been ad
Trap ITSL2_UPD_NODE_DEL has been added.
Trap ITSL2_UPD_IF_DOWN has been added.
Trap ITSL2_UPD_NODE_DOWN has been added.
Trap ITSL2_UPD_NODE_MARGINAL has been ad
Done.

Switch Analyzer Installation Complete.
```

*Figure 5-9 Traps added during the installation of IBM Tivoli Switch Analyzer*

## 5.2.6 Root cause analysis using IBM Tivoli Switch Analyzer and NetView

This section describes the layer 3 and layer 2 root cause analysis provided when using IBM Tivoli Switch Analyzer and NetView.

### NetView layer 3 router fault isolation

The NetView Router Fault Isolation (RFI) function identifies a network outage problem at the IP layer. If the problem is with a router, NetView issues a Router Status trap and calculates the impact.

Subnets and routers in the impacted partition are set to the Unreachable status. However, if the problem is with a layer 2 device, such as a switch, NetView identifies the nearest impacted router as the root cause. It cannot detect port problems in the layer 2 switch. This problem is solved by the IBM Tivoli Switch Analyzer product.

### IBM Tivoli Switch Analyzer layer 2 root cause analysis

IBM Tivoli Switch Analyzer program extends the ability of NetView to identify problems at the port level of layer 2 devices. It reports the root cause whether it is



at the layer 2 or the layer 3 level, either confirming the RFI result or identifying the failing layer 2 device.

The IBM Tivoli Switch Analyzer program is triggered by an Interface Down trap from NetView. It then determines the root cause of the problem and performs the following actions:

1. Issues a trap that identifies the root cause device whether it is a router, switch, or end node.
2. Updates the object database Layer2Status field if it is a switch or router. The IBM Tivoli Switch Analyzer program continues to monitor and update the root cause device and associated devices until the problem is resolved.

If the situation changes, the IBM Tivoli Switch Analyzer program performs these actions:

- ▶ Issues an update trap for the root cause device.
- ▶ Updates the object database Layer2Status field.
- ▶ Identifies any new downstream problems as they become known.

## 5.2.7 Real-life example

To demonstrate the ITSL2 capabilities, we used the lab environment shown in Figure 7-1 on page 360. We produced a few controlled interruptions to show the work of NetView and IBM Tivoli Switch Analyzer. The focus was to see how NetView and, to a lesser degree, IBM Tivoli Enterprise Console act on various forms of outages. Where applicable, we show the resulting IBM Tivoli Enterprise Console events with the supplied TEC\_ITS.rs NetView rules in effect.

Actually, we couldn't produce real problems by manipulating the hardware itself. However, pulling a connection had the same effect of triggering a root cause analysis.

Two of the three tests were conducted in segment 9.24.106.176 of our lab environment. This segment contains three Cisco 1900 switch devices connected together in a daisy chain. Each switch has one or two of our lab resources connected to an arbitrary port as shown in Figure 5-10.

To collect the results in a reasonable time, we changed the default polling time of NetView (under **Options** → **SNMP Configuration**) to one minute and the interface\_timeout in /usr/OV/ITSL2/conf/correlator.ini to the value 20 meaning 20 seconds as shown in Example 5-3.

The interface\_timeout value specifies the time between when an interface down event is received and the time where IBM Tivoli Switch Analyzer starts searching for a root cause. A timeout of 20 seconds is appropriate for our small lab network

with sufficient bandwidth. For more complex environment, increase this value to a value up to the default of 300 seconds or five minutes, depending on the average propagation time in your network.

*Example 5-3 Changed interface\_timeout in /usr/OV/ITSL2/conf/correlator.ini*

---

```
[Correlation]
topo_cache=./cache/topo_cache
topo_cache_freq=900
corr_cache=./cache/corr_cache
corr_cache_freq=120
#interface_timeout=300
interface_timeout=20
corr_timeout=10
interface_bounce_count=3
interface_bounce_interval=3600
down_event_severity=5
up_event_severity=4
```

---

### **Case 1: Removing a node**

The first test is simple. We disconnected the node DTMWAS01 from its switch port. This simulated a broken cable as marked in Figure 5-10. Since the node is no longer available, this causes a port down condition at the switch.

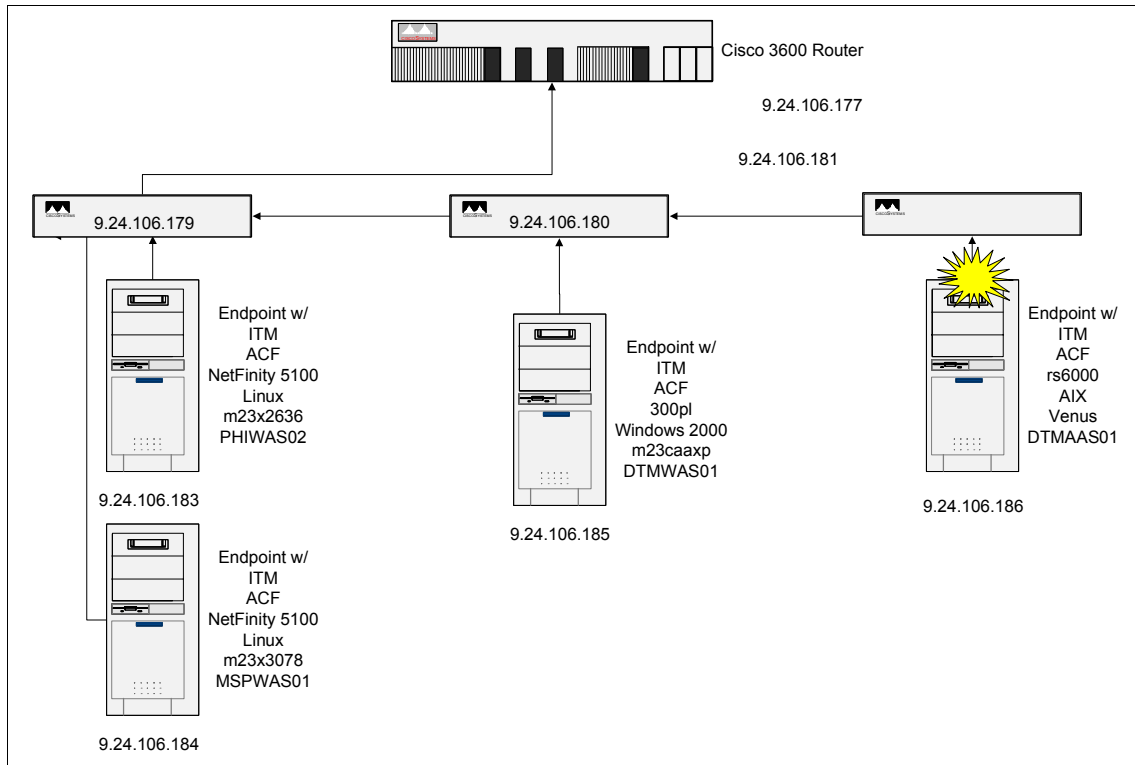


Figure 5-10 The first interruption

Removing an object managed by NetView and displayed in any submap causes at least a Node down event being created by NetView because the next status poll is unanswered. As a result, the node should be set to critical (red) in the submap. Figure 5-11 shows the results of the action.

As expected, NetView marks the node itself critical. If you look at the upper part of the event console, where all incoming events are displayed, you can see the related events. The second and third event from the top are the events that caused NetView to mark the node as critical. The first event is actually a trap issued by the switch itself, which in this case can be the indication of the root cause. The Cisco\_Link\_down trap shown as the first event passes the failing port as the only argument.

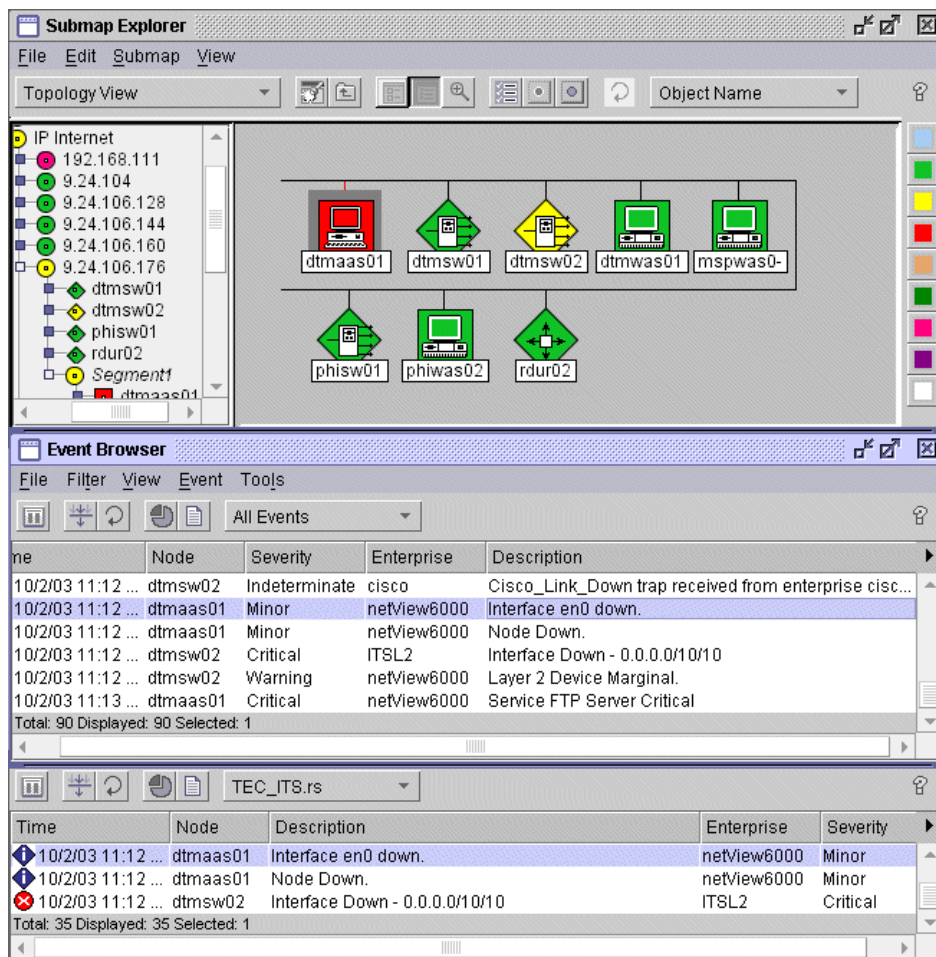


Figure 5-11 Results of a broken cable

Back to the topology window. You can see the switch dtmsw02 being marked as marginal. Normally, this is impossible. NetView allows only nodes that have multiple Layer3 interfaces being marked as *marginal*. A switch appears in NetView as an object represented only by its management interface.

If you look at the event console again, you can see a new event (fourth from top) issued by ITSL2. It clearly states the port where the node was connected and is now failing. This event causes event 5 to be generated by NetView again giving the information of a Layer 2 device marginal, which matches with the topology.

The last event that you can see is a nice side effect. That is NetView's new servmon, in our lab environment, monitors the File Transfer Protocol (FTP)

service provided by several nodes in the lab. Because the node is no longer available, it correctly signals an FTP service as *critical*.

Now let's look at the second event console, the lower part of Figure 5-11. The second console shows the events after the event stream passed the TEC\_ITS rule on NetView, which normally provides a filter for the network events forwarded to IBM Tivoli Enterprise Console. In this console, only three events are displayed. The filter removed redundant events leaving the relevant information.

Last, IBM Tivoli Enterprise Console works on the events sent to the enterprise console (Figure 5-12). It correlates the interface down and node down event detected for node dtmaas01 leaving the information about the failing server, and in the second event (the root cause of that problem), the failing switch port.

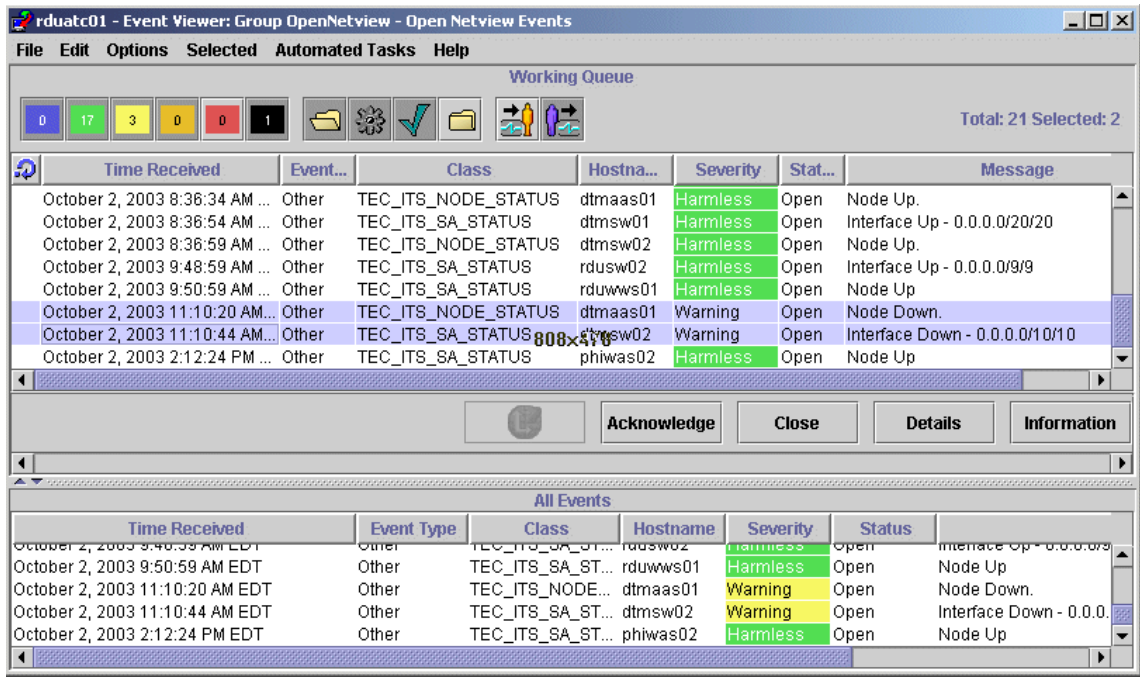


Figure 5-12 Incident as it appears in the IBM Tivoli Enterprise Console

Finally, we reconnected the node to the switch (Figure 5-13). As expected, the clearing events relevant to the problem were issued by NetView and IBM Tivoli Switch Analyzer. IBM Tivoli Enterprise Console correctly shows only the clearing events for the two violation events caused by the removal of the node. You can now use the clearing events inside IBM Tivoli Enterprise Console to perform the necessary cleanup.

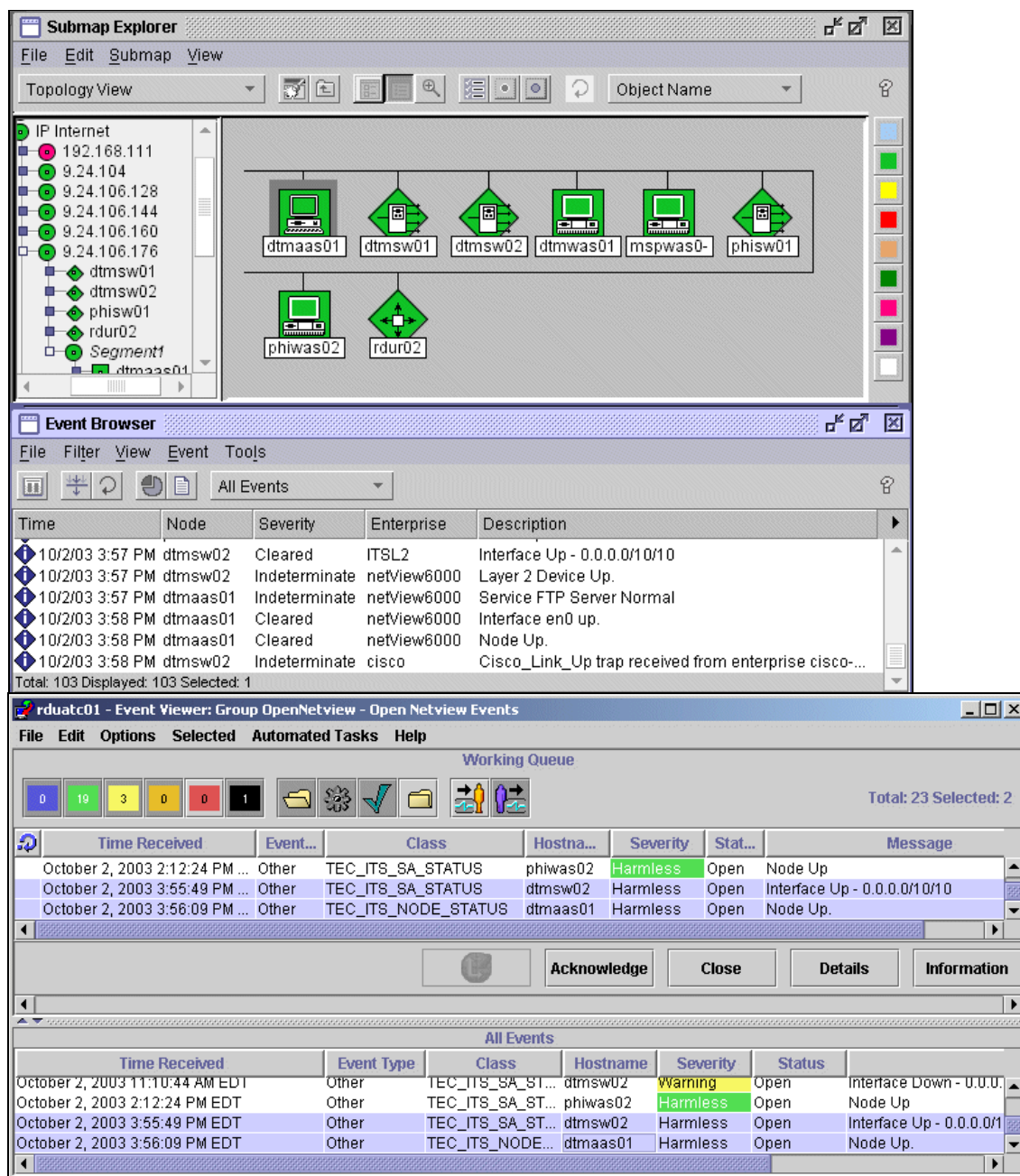


Figure 5-13 Connection re-established

## Case 2: Removing a downstream switch

The second test that we ran involved a problem in a switch hierarchy. As shown in Figure 5-14, you see three switches that are daisy-chained together, emulating a typical hardware switch hierarchy. A layer 3 manager would see the three switches as individual objects, represented by their management interface.

NetView places switches into the segment submap, but is unable to identify switch-related outages. Without IBM Tivoli Switch Analyzer, NetView again would report a single node down, the root cause. A broken link between two switches would not appear.

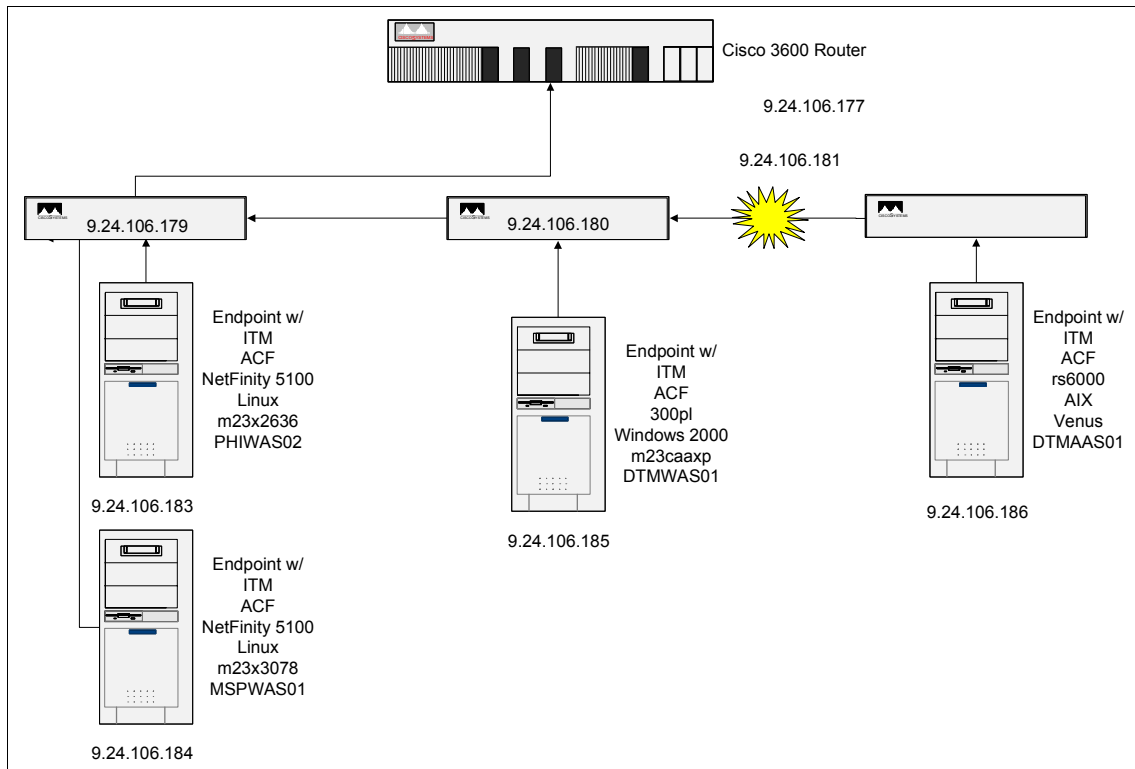


Figure 5-14 Disconnecting a downstream switch

Figure 5-15 shows the results of the combined layer 3 and layer 2 analysis:

- ▶ The NetView topology correctly marked nodes dtmaas01 and switch dtmsw01 as *critical*. These are the two downstream objects that NetView cannot reach anymore through its status poll.
- ▶ The topology shows switch dtmsw01 as marginal. Without IBM Tivoli Switch Analyzer, this switch appears as *up*. NetView can still reach the management

interface of the switch. Through IBM Tivoli Switch Analyzer's root cause analysis, the switch is clearly marked as *marginal*.

- The event console shows the same. It shows node down events for the two downstream devices and an ITSL2 event clearly pointing to the root cause, the *failing* interface of switch dtmsw01.

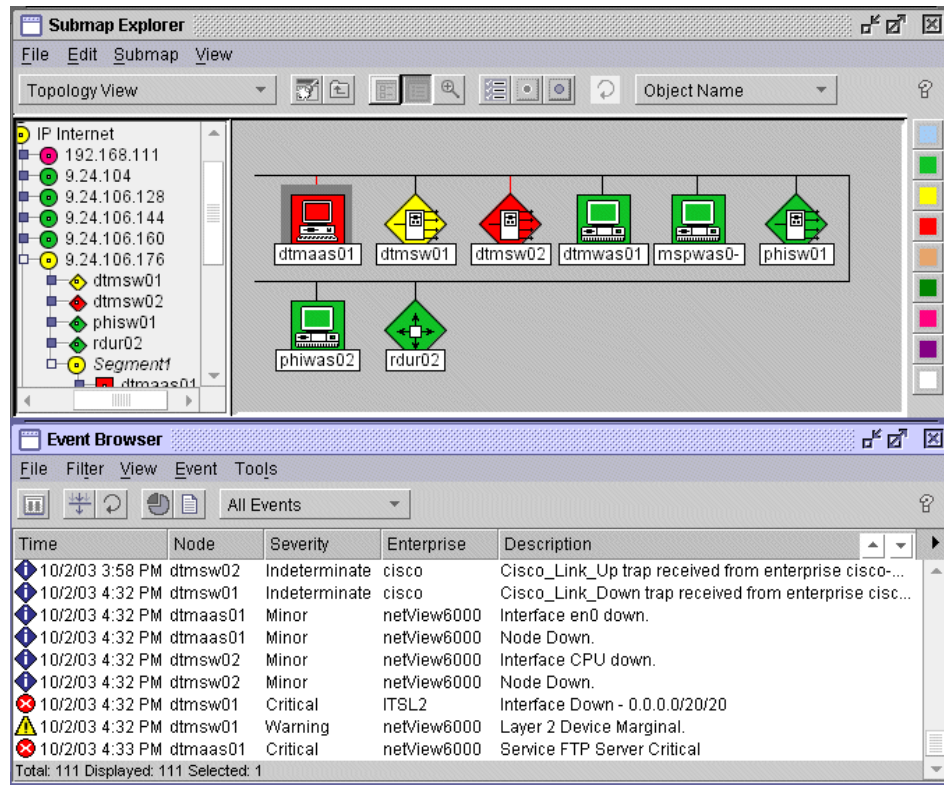


Figure 5-15 A broken downstream link

### Case 3: Defective router link

In our last test, we simulated a more complex issue in an IP network: unreachable networks due to a failing router connected via a switch. Normally, NetView reports the router as either marginal or down and marks the affected IP segments as *unreachable*. By default, NetView does not poll any object in an unreachable segment. Therefore, the status of those devices is *unknown* as long as the segment is unreachable.



Figure 5-16 shows the position in our lab network where we disconnected the IP segment.

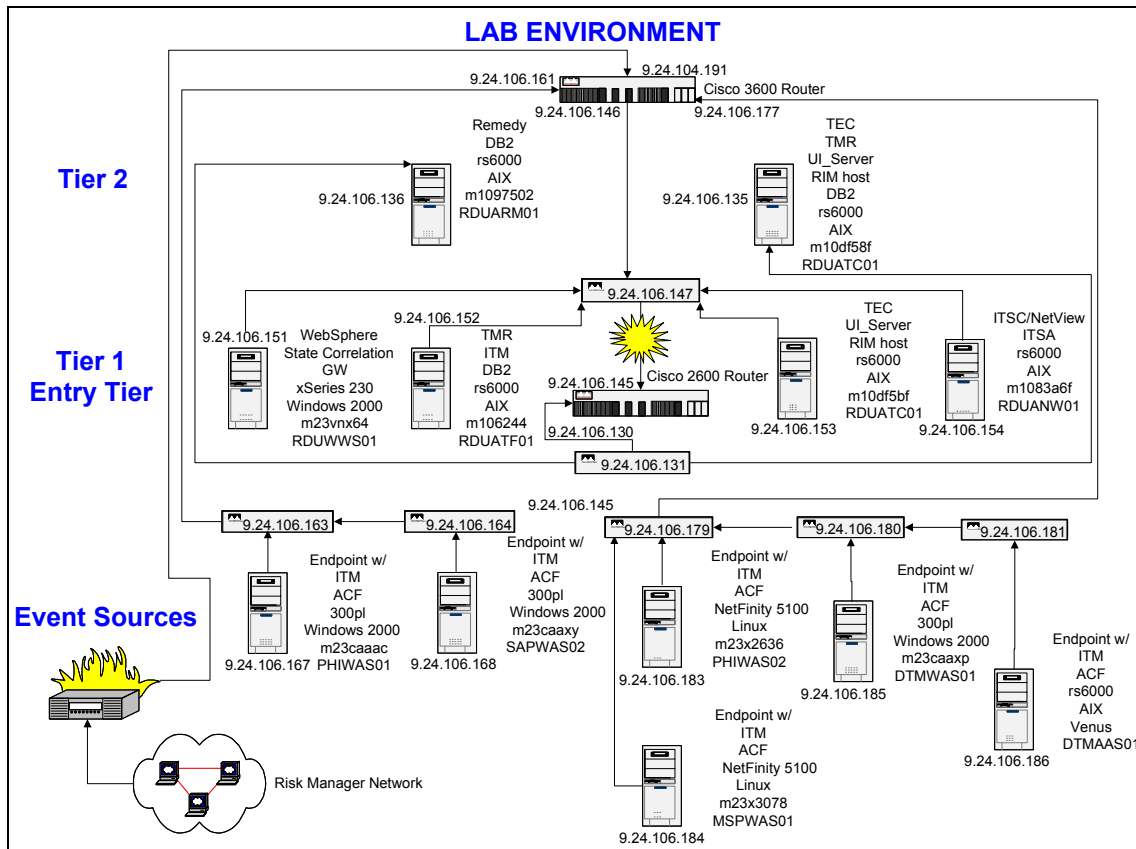


Figure 5-16 A broken router link

Shortly after we pulled the cable, which connected the router to the switch, NetView displayed the results of its findings (Figure 5-17). The results show the NetView IP Internet submap. You can see a critical route rdur01 as unreachable segment 9.24.106 and a marginal segment 9.24.106.144. As shown in Figure 5-17, it appears that the router itself is down.

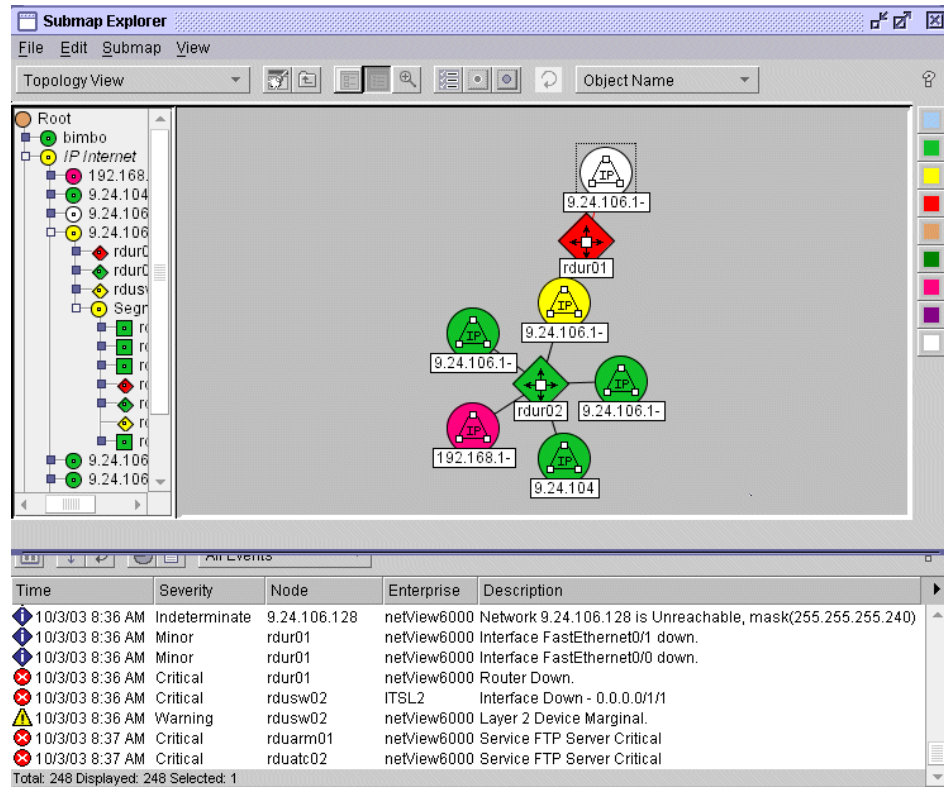


Figure 5-17 A failing router link

If we drill down to the next lower level as shown in Figure 5-18, you see more information:

- ▶ The segment itself is marked *marginal*, because it contains at least two objects where NetView cannot status poll all interfaces.
- ▶ The router, as expected, is marked *critical*, because no interface can be reached by NetView.
- ▶ The switch rdusw02 is marked as *marginal* as a result of an IBM Tivoli Switch Analyzer analysis. If you look at the interface submap of that router, you notice a running management interface for that switch.

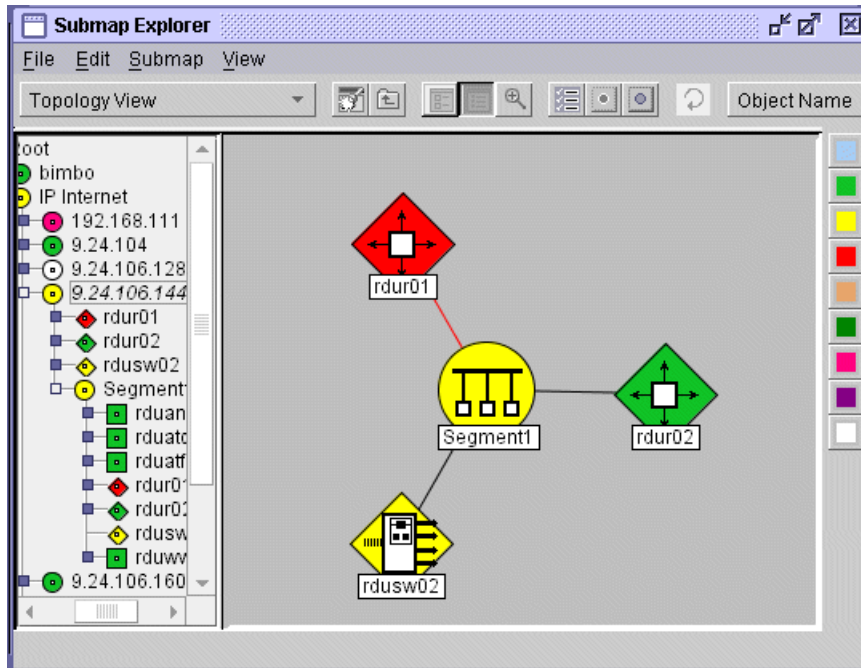


Figure 5-18 Failing router on the segment submap

This information gives us a hint. The router is down and a related switch is marginal. It's likely that the switch has something to do with this situation.

You can finally find the root cause in the event console shown in Figure 5-17. The fifth segment from the top, issued by ITSL2 clearly marks a port as critical.





## Event management products and best practices

This chapter describes the features of IBM Tivoli Enterprise Console with its NetView Integrated TCP/IP Services Component V7.1.4 and IBM Tivoli Switch Analyzer V1.2.1. It also examines how you can use them to implement the event correlation and automation best practices discussed in Chapter 2, “Event management categories and best practices” on page 25.

Only those product features that perform event management functions are described here. For more in-depth discussion about the general capabilities and features of IBM Tivoli Enterprise Console, NetView, and Switch Analyzer, see the following chapters respectively or the product documentation:

- ▶ Chapter 3, “Overview of IBM Tivoli Enterprise Console” on page 85
- ▶ Chapter 4, “Overview of IBM Tivoli NetView” on page 101
- ▶ Chapter 5, “Overview of IBM Tivoli Switch Analyzer” on page 141

## 6.1 Filtering and forwarding events

Best practices dictate filtering as close to the source as possible to minimize bandwidth usage and save event processor cycles (see 2.3.3, “Where to filter” on page 41). In some cases, this implies filtering at the source, and in others at the closest event processor.

IBM Tivoli NetView, Switch Analyzer, and Enterprise Console can all serve as both event sources and event processors. In their roles as event sources, the products are capable of generating events. As event processors, they can receive events and perform actions, such as filtering, upon them.

We discuss the filtering and event suppression capabilities of the products when functioning both as event sources and processors.

### 6.1.1 Filtering and forwarding with NetView

This section covers the event filtering and forwarding capabilities of the NetView network manager, . Its purpose is to introduce the product features that perform these functions and provide some examples. Consult the product documentation for a complete discussion of possible configuration options.

NetView is an Simple Network Management Protocol (SNMP)-based network manager, and as such, functions as a trap receiver. Depending upon the systems management infrastructure, NetView may act as an intermediate manager, passing relevant events to a higher level event processor, or as the management focal point, displaying them on its console and taking actions for them.

More often, NetView is positioned to feed status and events to another event processor such as a business systems manager or enterprise console. Therefore, we concentrate on its forwarding capabilities.

When functioning as an event processor, NetView can forward traps to the IBM Tivoli Enterprise Console as events or to another SNMP manager as SNMP traps. Configuring NetView forwarding really defines filtering, since anything that is not chosen for forwarding is automatically filtered.

There are many methods for filtering and forwarding traps or events using NetView. Some are available to all versions of NetView, and some are operating system specific. The differences are noted as appropriate. Since NetView processes IBM Tivoli Switch Analyzer events, all the forwarding methods discussed here apply to both IBM Tivoli NetView and Switch Analyzer traps.

## Using trapd.conf

This file is used in both NetView for UNIX and NetView for Windows to configure traps. The UNIX version of the file is `/usr/OV/conf/C/trapd.conf`. The Windows one is `\usr\OV\conf\C\trapd.conf`. If a trap is not defined in this file, NetView does not recognize or process it.

This seems to imply that `trapd.conf` should contain entries for only the traps of interest. This is not the case. When SNMP devices are configured to send traps to a trap receiver, they are often capable of sending all or none, or a subset of traps based upon a general category such as security or configuration. Rarely, if ever, is the administrator allowed to select, by trap, which are desired.

Therefore, inevitably, NetView receives unnecessary traps. If `trapd.conf` does not contain a definition for them, NetView generates a trap stating that it received a trap of unknown format. This adds processing overhead to NetView and clutters both the `trapd.log` file and the operator consoles. The cost of determining that a trap is undefined is substantially higher than the identification of a known trap.

Therefore, the *best practice* that we offer here is to ensure that `trapd.conf` contains definitions for any expected traps, not just those of interest.

Operators usually do not understand that the unknown format message was generated by NetView in response to a trap it received and could not process. They often erroneously believe that the message indicates an error with the device that sent the original trap, and dispatch someone to resolve it. The dispatched person checks the SNMP device, cannot find the “unknown format” problem in any of its logs, and does not know how to eliminate the message.

Also, large numbers of these traps can obscure real traps requiring action. This is especially a problem in environments where NetView consoles are used as the primary means of problem notification. Configure NetView for all expected traps to minimize these problems.

Traps of an unknown format, like all traps, contain an enterprise ID. A little research can reveal the vendor associated with that enterprise and possibly identify the equipment that generated the unexpected traps. Check with the vendor to obtain a script or Management Information Base (MIB) containing the appropriate traps.

**Important:** Never edit the `trapd.conf` file directly. Use one of the methods discussed here to add entries to the file.

### ***Adding entries to trapd.conf***

The trapd.conf file contains information about each enterprise from which traps can be received, the format of the trap, and actions to take upon receipt.

Example 6-1 shows entries for the enterprise used by NetView, netView6000, and a node down trap, IBM\_NVNDWN\_EV. The enterprise is represented as a dotted decimal number. Traps that are generated for this enterprise refer to the same number in braces {}. A trap is uniquely identified by enterprise ID and generic and specific trap numbers. The node down event has enterprise 1.3.6.1.4.1.2.6.3 (netView6000), generic trap number 6, which indicates it is an enterprise-specific trap, and specific trap number 58916865. We refer to this example later to explain other fields within the trap definition.

---

#### *Example 6-1 Excerpt from trapd.conf*

text omitted

netView6000 {1.3.6.1.4.1.2.6.3}

text omitted

```
IBM_NVNDWN_EV {1.3.6.1.4.1.2.6.3} 6 58916865 N 4 0 "Status Events"
$3
EVENT_CLASS TEC_ITS_NODE_STATUS
BEGIN_SLOT_MAPPING
  msg $V3
  nodestatus DOWN
  iflist $V8
END_SLOT_MAPPING
SDESC
```

This event is generated by IBM Tivoli NetView when  
it detects a node is down

The data passed with the event are:

- 1) ID of application sending the event
- 2) Name or IP address
- 3) Formatted description of the event
- 4) Timestamp of the event and objid of the node object
- 5) Database name
- 6) Selection Name
- 7) (not used)
- 8) Interface list

EDESC

text omitted

---



There are four ways to add traps to trapd.conf. The best method to use depends upon several factors. Here, we discuss the methods, their pros and cons, and suggestions for when to use each:

- NetView trap configuration window

Figure 6-1 shows an example of adding a trap to NetView for UNIX from the trap configuration window. To access this window, in NetView for UNIX, select **Options** → **Event Configuration** → **Trap Customization: SNMP**. Or from a UNIX command prompt, type:

```
/usr/OV/bin/xnmtrap
```

On Windows, either the NetView main menu or Event Browser windows, you select **Options** → **Trap Settings**, or run \usr\OV\bin\trap.exe.

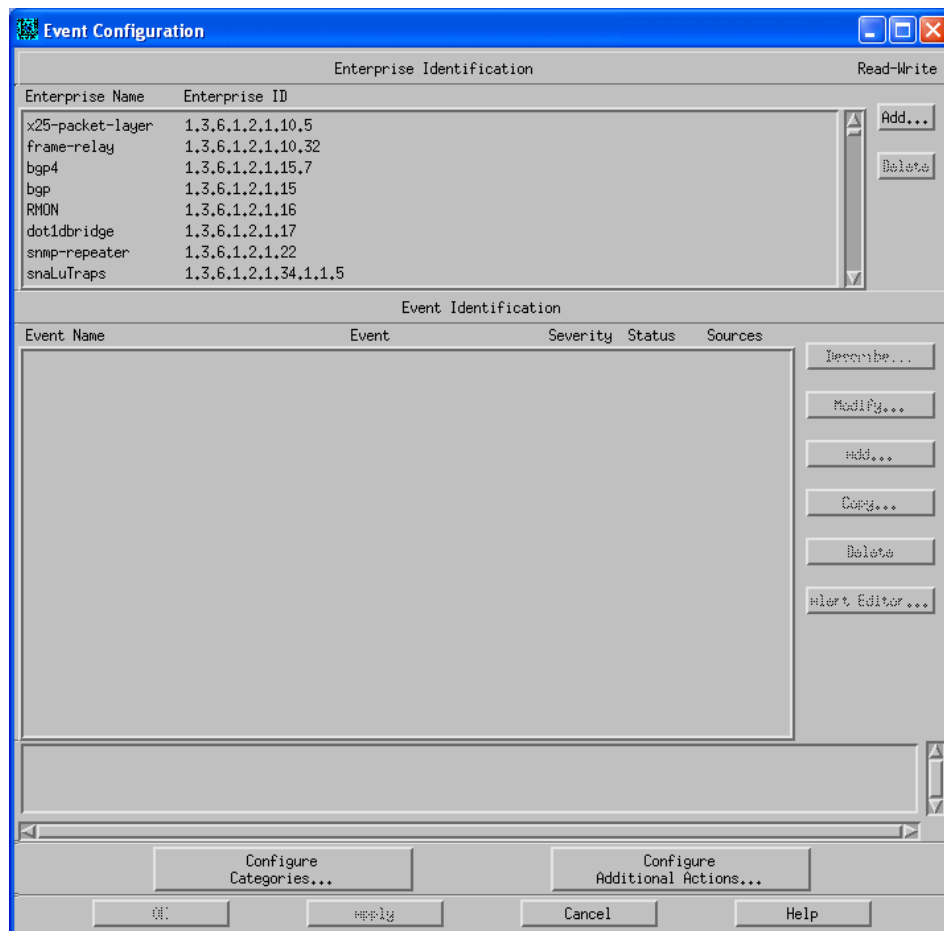


Figure 6-1 NetView trap configuration window

If the enterprise, to which the trap should be added, exists, select it from the top of the window. Otherwise, click the **Add** button and complete the fields as shown in the Add New Enterprise window (Figure 6-2) to add it.

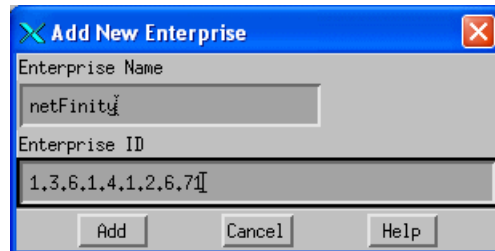
A screenshot of a Windows-style dialog box titled "Add New Enterprise". It has a blue title bar with a green "X" icon on the left and a red "X" icon on the right. The dialog contains two text input fields. The first field is labeled "Enterprise Name" and contains the text "netFinity". The second field is labeled "Enterprise ID" and contains the text "1.3.6.1.4.1.2.6.71". At the bottom of the dialog, there are three buttons: "Add", "Cancel", and "Help".

Figure 6-2 Adding an enterprise to *trapd.conf*

To add the new trap to the highlighted enterprise, click **Add** in the middle section of the Modify Event window (Figure 6-3). Complete the values, and click **OK**.

Use this method when you add a few traps. It becomes cumbersome to use when adding a several of them. It also is difficult to know which values to place in some fields without knowing the source and meaning of the trap.

Figure 6-3 Adding a trap to trapd.conf

► The **addtrap** command

You run this command from a UNIX or Windows command prompt to add new traps and enterprises to the trapd.conf file and to update existing traps. If the **addtrap** command references an enterprise that does not yet exist, NetView defines it.

NetView uses the enterprise ID, generic-trap, and specific-trap values to uniquely identify traps. If a trap already exists in the file with the same identifying values as specified in the **addtrap** command, NetView updates it rather than adding another. After updating the trapd.conf file, NetView sends an event to the trapd daemon informing it of the update.

Example 6-2 shows using the **addtrap** command to update the trapd.conf file. For a detailed explanation about the syntax, see *Tivoli NetView for UNIX Administrator's Reference, Version 7.1*, SC31-8893, or *Tivoli NetView for Windows NT Programmer's Reference, Version 7.1*, SC31-8890.

---

*Example 6-2 The addtrap command to define an IBM 6611 trap*

---

```
addtrap -n ibm6611
-l mytrap
-i .1.3.6.1.4.1.2.6.2
-g 6 -s 16 -o A -t 3
-c "Status Events"
-f !
-F '$E $G $S $T'
-S 4
-C xecho
-A 'Status Event received from 6611 agent $E$G $S'
-e nodeDown
-E msg
-V 'PRINTF ("Node %s down",$V2)'
```

---

This command specifies the following information:

- n The enterprise name is ibm6611.
- i The enterprise ID is 1.3.6.1.4.1.2.6.2.
- g The generic trap number is 6.
- s The specific trap number is 16.
- o The trap is sent from an agent, in this case, the 6611 router agent.
- t The object that generates the trap is to be assigned a status of *Critical* on the map.
- c This is a status event.
- f A specified action (see -C and -A) is performed by the management system when this trap is received.
- F The enterprise name (\$E), generic (\$G) and specific (\$S) event numbers, and the time-stamp (\$T) are displayed in the event cards or list.
- S The trap is a Severity 4 (Critical) trap.

- C The **xecho** command is activated when this event is received.
- A The following arguments are passed to NetView with this event:
  - v This is the event text (Threshold Event received from 6611 agent).
  - v This is the enterprise name (\$E).
  - v This is the generic trap number (\$G).
  - v This is the specific trap number (\$S).
- e This event is forwarded to the IBM Tivoli Enterprise Console with an event class of nodeDown.
- E This event is forwarded to the IBM Tivoli Enterprise Console containing the event text specified by the -V flag.
- V Trap variable 2, the host name, is substituted in the event text for the %s format specifier when this event is forwarded to the IBM Tivoli Enterprise Console.

For more information about the **addtrap** command, refer to the man page.

► Vendor-supplied script

Some vendors provide scripts that can be executed from a UNIX or Windows command prompt to add traps to trapd.conf for their products. The script contains an **addtrap** command for each trap definition required.

This is generally the preferred way to add traps to trapd.conf. A vendor knows the purpose of the trap, when it is generated, and which variables are passed with it. Therefore, the vendor is most capable of setting the values in the **addtrap** command.

► Management Information Blocks

Traps may be packaged in MIBs. When a vendor supplies traps in the MIB format, use the **mib2trap** command to create a script that contains the appropriate **addtrap** commands. This utility checks the MIB file for entries designated as TRAP-TYPE, extracts the appropriate fields from it, and builds a script containing **addtrap** commands. Then you run the script to add the traps to trapd.conf.

Sometimes the **mib2trap** command cannot properly process an MIB file. This may occur if it encounters unexpected syntax or cannot resolve a variable. The **mib2trap** output indicates the line or lines in error. Edit the file to remove any unexpected syntax. To resolve variables, concatenate several MIBs into a single file and re-execute the **mib2trap** command using the combined file as input. Place the MIB file containing the definition of the unresolved variable before the one containing the traps to ensure proper variable resolution.

This method is available for both NetView for UNIX and NetView for Windows.

### ***Filtering in trapd.conf***

Traps defined in `trapd.conf` are assigned a category that determines how the events are grouped in the NetView event window. The categories indicate whether a trap is generated by a status change, application, threshold crossing, node configuration, error, or NetView map or topology change. In Example 6-1 on page 176, the NetView Node Down trap has the category *Status Events*.

In this case, we suggest that you consider this *best practice*. Filter events using LOGONLY to eliminate traps that you never want to see. LOGONLY is a special category used to prevent messages from being displayed or forwarded. The traps are logged in `trapd.log`, but are not forwarded to the applications registered to receive traps. The traps are not forwarded to other SNMP managers or enterprise console servers.

Set this filter using the `-c LOGONLY` parameter of the **addtrap** command or through the Modify Event window shown in Figure 6-4.

Logging the events in `/usr/OV/log/trapd.log` for UNIX or `\usr\OV\log\trapd.log` for Windows ensures that they are available when required for debugging purposes.

Use LOGONLY instead of “Don’t log or display” (`-c IGNORE`). This is essential for troubleshooting flooding conditions. For example, suppose that the NetView server begins to experience performance problems due to an event flood. If the offending event is one that has been suppressed from `trapd.log` and from the console, it is difficult to target the cause of the performance problem. Even if the administrator suspects the event storm as the cause, there is no easy way to identify the culprit event, short of tracing the appropriate daemon.

While NetView for UNIX automatically logs events to `trapd.log`, NetView for Windows does not. Configure the latter by running `nvsetup.exe`. Then click the **Daemons** tab and choose **Trap Daemon**. Select the **Log Events and Traps** check box.

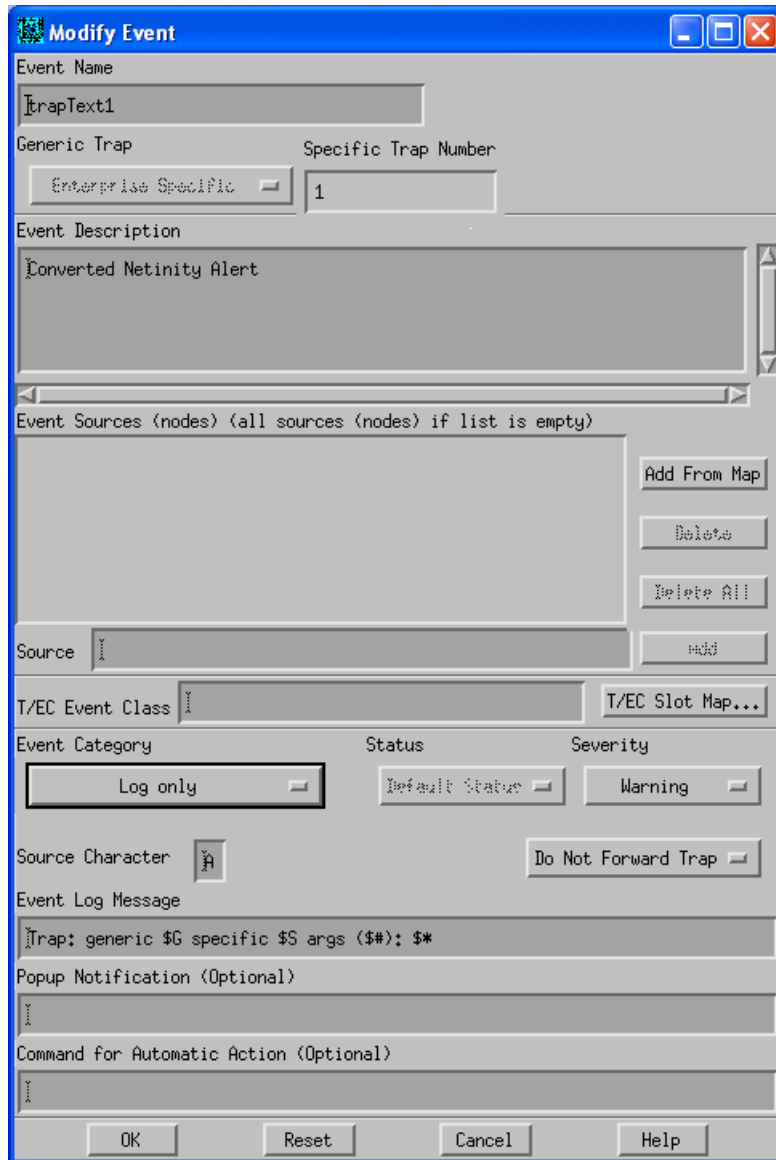


Figure 6-4 Setting category through trap customization window

### **Forwarding SNMP traps using trapd.conf**

NetView can forward traps to another SNMP-based manager such as the IBM Tivoli Enterprise Console SNMP adapter or another NetView. This function is typically used to consolidate events from regional NetView servers to a central NetView server, or to take advantage of the more granular IBM Tivoli Enterprise

Console class assignment capabilities of the SNMP adapter (see “Using the NetView adapter” on page 191).

For NetView for UNIX, trap forwarding is set in the trapd.conf file. Figure 6-5 shows how to flag traps for forwarding in NetView’s trap customization window.

**Modify Event**

Event Name  
TrapText1

Generic Trap: Enterprise Specific      Specific Trap Number: 1

Event Description  
Converted Netinity Alert

Event Sources (nodes) (all sources (nodes) if list is empty)

Source:      Add From Map      Delete      Delete All      Add

T/EC Event Class:      T/EC Slot Map...

Event Category: Status Events      Status: Default Status      Severity: Warning

Source Character: A      Forward Trap: ☒

Event Log Message  
Trap: generic \$G specific \$S args (\$#): \$\*

Popup Notification (Optional)

Command for Automatic Action (Optional)

OK      Reset      Cancel      Help

Figure 6-5 Setting trap forwarding in trapd.conf



Traps that are flagged for forwarding contain the keyword FORWARD in trapd.conf.

In addition to knowing which events to forward, NetView needs to know the destination to which traps should be forwarded. This is set when configuring trapd.

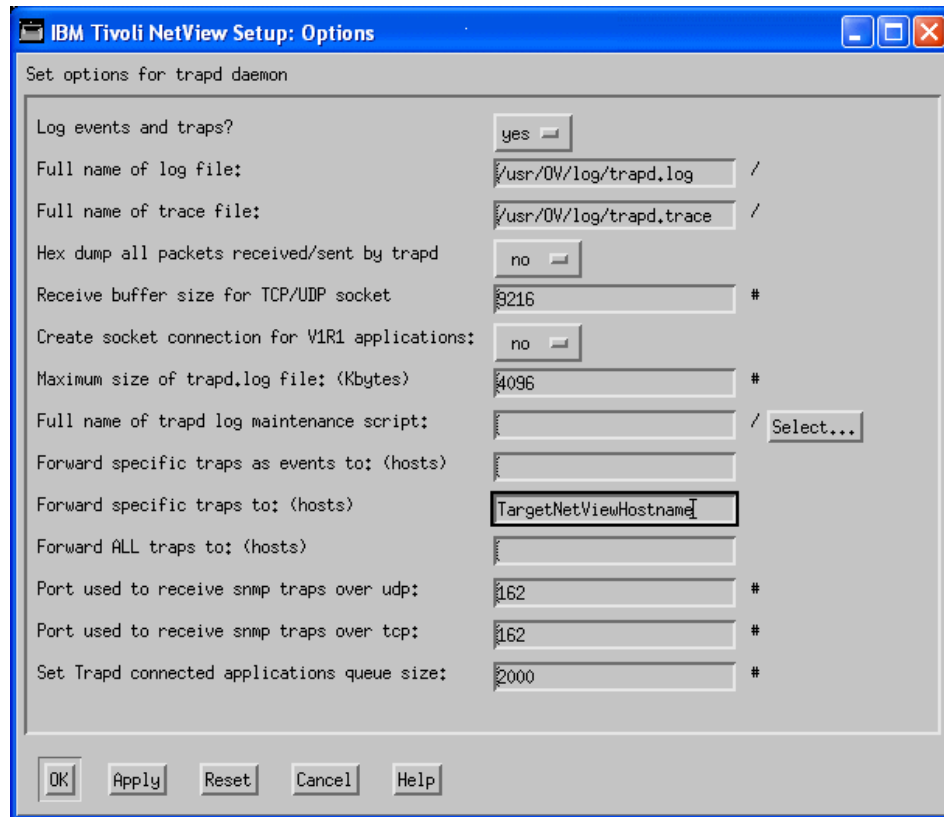


Figure 6-6 Configuring trapd to forward traps

NetView forwards all traps that are flagged in trapd.conf to the hosts specified in the trapd daemon configuration.

### **Using trapfrwd daemon**

The trapfrwd daemon is the NetView for Windows method of forwarding traps to other management stations. Again, it is used in environments where there is higher level SNMP manager that consolidates traps.

The trapfrwd daemon uses the \usr\OV\conf\trapfrwd.conf file to determine where to send traps and what traps to send. This daemon is not started by default. To

configure the trapfrwd daemon, modify the trapfrwd.conf file and then start the trapfrwd daemon.

We explain the trapfrwd.conf file format here. The first section contains the host name and trap port number pairs of the management stations to send traps. These pairs must be enclosed between the [Hosts] and [End Hosts] tags, as shown in Example 6-3.

*Example 6-3 trapfrwd.conf Hosts sample*

---

```
[Hosts]
mgtsys1 1662
mgtsys2 0
[End Hosts]
```

---

The second section, as shown in Example 6-4, contains the enterprise object identifier (OID) and trap number pairs (the trap numbers are from the trapd.conf file). These pairs must be enclosed between the [Traps] and [End Traps] tags.

*Example 6-4 trapfrwd.conf Traps sample*

---

```
[Traps]
1.3.6.1.4.1.2.6.3 591790
1.3.6.1.4.1.2.6.3 589824
[End Traps]
```

---

To add a comment, precede the line with a pound sign (#).

After you configure hosts and traps in the trapfrwd.conf file, start the trapfrwd daemon by entering the following commands on the command line:

```
ovaddobj \usr\ov\lrf\trapfrwd.lrf
ovstart trapfrwd
```

### ***Maintaining trapd.log***

Another configuration option for the trapd daemon is the size of the trapd.log file. This parameter is set to 4096 KB by default. When trapd.log reaches its maximum size, it is moved to trapd.log.old in the same directory, and a new trapd.log is written.

In environments with many, often unwanted, events, it is possible that trapd.log.old is replaced several times a day. This means there is not even 24-hours worth of trap data online for debugging purposes.

Therefore, we offer this *best practice*: Save old trapd.log data.

The trapd daemon can be configured to automatically archive the trapd.log data to a file or database. Configure trapd to run a custom script or the supplied trapd.log\_Maint script when trapd.log reaches maximum size. Root permissions are required to perform this task on NetView for UNIX.

The trapd.log\_Maint script does the following processing of the data in the trapd.log.old file, depending on the parameters you set for the trapd.log\_Maint script:

- ▶ Transfers the data to a relational database
- ▶ Archives the data in the specified directory

The data is archived in a file that includes a Julian date and time stamp in the file name to indicate when the data was archived. For example, the file name trapd.log.94215153001 indicates that this file was archived on 3 August 1994 at 3:30:01 p.m.

Discard archived data that is older than the specified maximum age. Verify that the maximum amount of disk space used to store archived trapd.log data has not been exceeded. When the specified limit is reached, the oldest trapd.log data is discarded.

**Note:** The archive maintenance actions do not affect trapd.log data stored in a relational database.

To maintain the trapd.log file by configuring the trapd daemon, follow these steps:

1. Enter **serversetup** on the command line to access the Server Setup application.
2. Select **Configure** → **Set options for daemons** → **Set options for event and trap processing daemons** → **Set options for trapd daemon**.
3. The Set options for trapd daemon panel (Figure 6-6) opens. Make the necessary changes to the entry fields:
  - a. Specify the maximum size of the trapd.log file.
  - b. Enter the full name of the trapd log maintenance script.
  - c. Enter the full path name of any script you want to use, or click the **Select** button and select the **trapd.log\_Maint** script from the list of choices.
  - d. Click **OK**.

If you did not select the trapd.log\_Maint script, the trapd daemon is configured as specified. Skip to step 5. If you selected the trapd.log\_Maint script, make the necessary changes to the trapd.log\_Maint parameters that are displayed:

- Directory for storage of archived trapd.log files
- Maximum age of any archived trapd.log file
- Maximum total size of all archived trapd.log files
- Migrate data to SQL database

Refer to the online help for additional information about the entry fields.

4. Click **OK**.
5. Click **Close**.

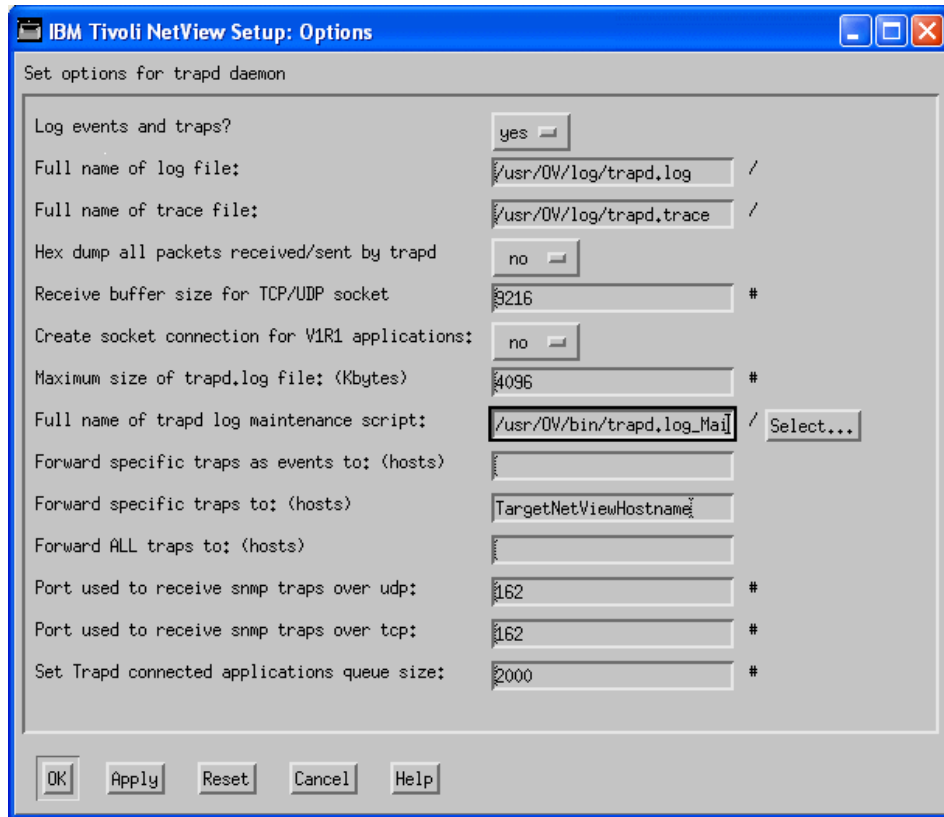


Figure 6-7 Configuring trapd to save old data

Storing old trapd.log data ensures that there is sufficient data for debugging purposes. The file can be referenced to determine whether an event is received at NetView and to display other events that were received around the time of a failure.

You need to have trapd save old data when trapd fills up. The trapd daemon can be set up to automatically save data to files or a database. In either case, you

must purge the old data. Sometimes when doing problem determination, it is useful to see if NetView receives an event.

Events are stored in the SQL database by default. To prevent the SQL database from growing too large, the trapd daemon periodically removes the oldest events from the database. You can control how many events are preserved and scheduled when, or if purging should occur. To do this, select **Options** → **Server Setup** and use the trapd daemon page. Optionally, you can specify that events are written to the log file `\usr\ov\log\oldevents.log` before they are purged from the database. Because this file grows over time, you need a policy on how to archive this file. This option is available only on a NetView server.

## Using nvserverd

The nvserverd daemon can be used on NetView for UNIX systems to forward events to the IBM Tivoli Enterprise Console event server. It is the easiest method of forwarding events from a NetView for UNIX system to an IBM Tivoli Enterprise Console event server.

**Note:** NetView for Windows uses a different method to send events. See “Using the NetView adapter” on page 191.

The format of the IBM Tivoli Enterprise Console events formed are defined in the trapd.conf file. Configure the IBM Tivoli Enterprise Console class and slots associated with a trap through the NetView trap configuration screen. You open this window by selecting **Options** → **Event Configuration** → **Trap**

**Customization: SNMP** in NetView for UNIX or by running the following command from a UNIX command prompt:

```
/usr/0V/bin/xnmtrap
```

Set the class to assign to the IBM Tivoli Enterprise Console server in the field marked T/EC Event Class. In Figure 6-26 on page 266, the IBM Tivoli Enterprise Console class is set to TEC\_ITS\_ROUTER\_STATUS.

Click **T/EC Slot Map** in the NetView trap configuration window to change slot variables. See the procedure outlined in “Setting event severity in NetView for UNIX” on page 270 for more details about setting slot variables. NetView for UNIX comes with several events already defined with IBM Tivoli Enterprise Console classes and slots that are used with the IBM Tivoli Enterprise Console rules supplied with the NetView product. Be sure to activate the NetView BAROC and RLS files supplied with the product in the IBM Tivoli Enterprise Console server or servers, which are to receive events from NetView.

To set up event forwarding using nvserverd, run **serversetup** and choose **Configure** → **Configure event forwarding to T/EC**. The Configure event

forwarding to IBM Tivoli Enterprise Console panel (Figure 6-8) opens. Change Forward events to Tivoli event server? to **Yes**, and enter the host name of the T/EC server. Also, supply the name of a NetView rule. Only traps that pass through the NetView rule are forwarded.

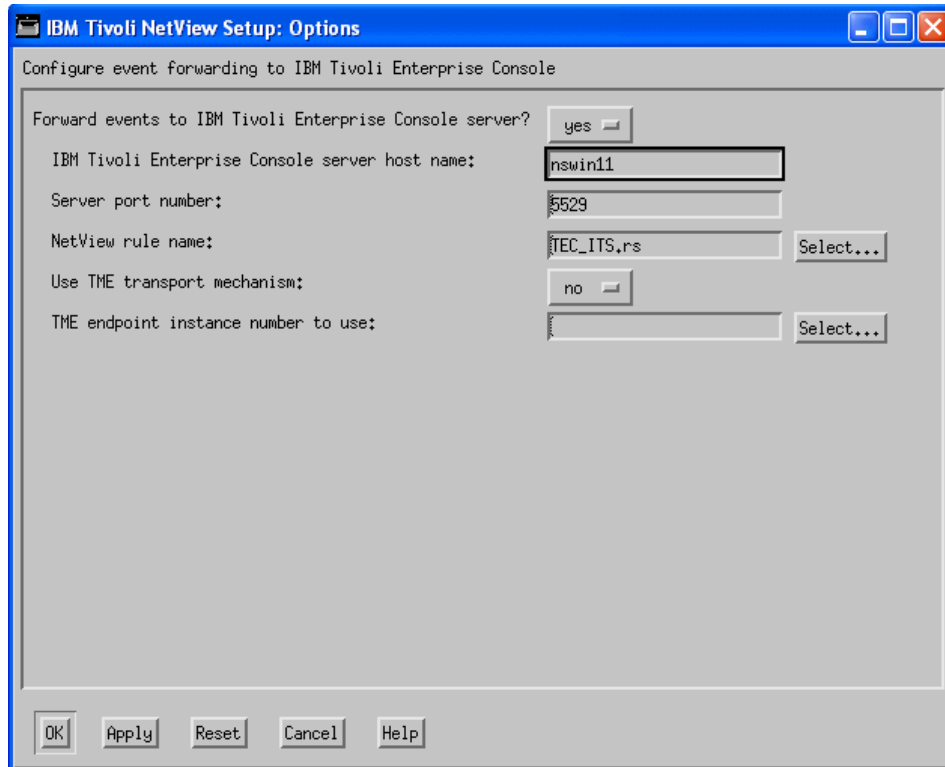
The image shows a Windows-style dialog box titled "IBM Tivoli NetView Setup: Options". The subtitle is "Configure event forwarding to IBM Tivoli Enterprise Console". The dialog contains several configuration fields: a checkbox for "Forward events to IBM Tivoli Enterprise Console server?" which is checked and labeled "yes"; a text field for "IBM Tivoli Enterprise Console server host name:" containing "nswin11"; a text field for "Server port number:" containing "5529"; a text field for "NetView rule name:" containing "TEC\_ITS.rs" with a "Select..." button to its right; a checkbox for "Use TME transport mechanism:" which is unchecked and labeled "no"; and a text field for "TME endpoint instance number to use:" which is empty with a "Select..." button to its right. At the bottom of the dialog are five buttons: "OK", "Apply", "Reset", "Cancel", and "Help".

Figure 6-8 Configuring event forwarding to T/EC

**Note:** The configuration information is stored in the `/usr/OV/conf/tecint.conf` file. If you prefer, you can edit this file directly instead of using the Server Setup application.

NetView supplies four rules out of the box as listed in Table 6-1.

Table 6-1 NetView supplied rules

Rule	Description
Default.rs	No events are forwarded.
Forwardall.rs	All events are forwarded.
sampcorrIuld.rs	Forwards an interface down trap if a node up trap is not received for the same device within 10 minutes.
sampcorrNdNu.rs	Forwards a node down trap if a node up trap is not received for the same device within 10 minutes.

We offer this *best practice*: Develop a NetView rule that forwards only events of interest. Using forwardall.rs forwards every event that NetView receives. Many are not of interest on an enterprise level. Suppress these to eliminate clutter on the operator's console and reduce IBM Tivoli Enterprise Console processing cycles.

Sometimes you may want more control over an event. For example, Cisco or Cabletron traps are sometimes generated with a variable indicating the real problem. Defining IBM Tivoli Enterprise Console event classes and slot variables in trapd.conf requires all traps of the same enterprise and specific and generic trap numbers to be mapped to the same IBM Tivoli Enterprise Console class.

If you want to assign these to their own IBM Tivoli Enterprise Console classes to make correlation or automation easier at the IBM Tivoli Enterprise Console event server, it is cumbersome using nvserverd. For example, sometimes a script is executed upon receipt of an event to generate a second event using **snmptrap** or an event using **wpostmsg** or **postmsg** to populate additional slots of interest, or map the problem to a different IBM Tivoli Enterprise Console class.

## Using the NetView adapter

The NetView adapter can be used to forward events to IBM Tivoli Enterprise Console from either NetView for UNIX or NetView for Windows. It is the only method supplied with NetView for Windows to forward events directly to the IBM Tivoli Enterprise Console event server. NetView for Windows can forward the events to another SNMP manager, such as the IBM Tivoli Enterprise Console SNMP adapter or NetView for UNIX, which in turn can forward to IBM Tivoli Enterprise Console.

NetView for Windows and NetView for UNIX can also run a script upon receipt of an event to issue a **postmsg** or **wpostmsg** to forward the event to the IBM Tivoli Enterprise Console event server.

There are some advantages of using the NetView adapter, even in a NetView for UNIX environment. When assigning IBM Tivoli Enterprise Console classes using `nvserverd`, every event associated with a trap number is given the same class. Sometimes this is not granular enough. Also, you may want to populate an IBM Tivoli Enterprise Console slot with a value, but that value is not contained within a separate trap variable. Using `nvserverd`, there is not an easy way to assign that value to its own IBM Tivoli Enterprise Console slot.

For example, NetView generates trap 58720263 to report thresholds exceeded for any MIB variable that it is checking. It uses trap 58720264 to report threshold re-arm. You may want to report the errors in a separate IBM Tivoli Enterprise Console classes based on the MIB variable. For example, one IBM Tivoli Enterprise Console class can be used for thresholds on Cisco's `avgBusy5` MIB variable, another IBM Tivoli Enterprise Console class used for Cisco's `bufferFail` variable, etc. This makes it easier for those viewing the events in the IBM Tivoli Enterprise Console. In some cases, it may simplify the creation of event correlation sequences at the IBM Tivoli Enterprise Console event server.

Using `nvserverd`, this requires NetView to run a script upon receipt of the trap. The script parses the event, and, based on the value of `blap2`, generates either a second trap with a user-defined trap number. Or you can run `wpostemsg` or `postemsg` to forward an event to IBM Tivoli Enterprise Console.

### ***Installing the NetView adapter***

Adapters can send events to the event server using a TME interface or a non-TME interface. Both types of interfaces send events using an ordinary TCP/IP channel. The difference between the two interfaces is the method used to establish the connection.

A TME interface establishes a connection using services provided by the TME 10 Framework. Therefore, adapters that use this interface are referred to as *TME adapters*. A non-TME interface establishes connections using standard interprocess communication mechanisms (for example, opening an IP socket). Therefore, adapters that use this interface are called *non-TME adapters*.

NetView adapters can run as TME or non-TME adapters. Choose the type that you want to run based on whether you are running Tivoli Framework on your NetView server. To use the adapter, perform the following steps:

1. Install the adapter.
  - On a NetView for UNIX system, run the `tecad_nv6k.cfg` file that comes with the IBM Tivoli Enterprise Console adapter.
  - For NetView for Windows, copy the appropriate executable. Enter either of the following statements in `\usr\OV\bin`, depending upon whether you are using the TME or non-TME version of the adapter



```
copy tecad_nv6k_tme.exe tecad_nv6k.exe
copy tecad_nv6k_non_tme.exe tecad_nv6k.exe
```

2. Register the adapter to NetView by entering the following command in a command window:

```
ovaddobj \usr\ov\lrf\tecad_nv6k.lrf
```

This is performed automatically in NetView for UNIX by executing the installation script **tecad\_nv6k.cfg**.

3. Identify the event server to be used by editing \usr\ov\conf\tecad\_nv6k.conf (Windows) or /usr/OV/conf/tecad\_nv6k.conf (UNIX).

- For the TME version of the adapter, specify:

```
ServerLocation@Event Server
```

- For the non-TME version, specify:

```
ServerLocation=hostname
```

Here *hostname* is the name of the host running the Tivoli Enterprise Console (non-TME adapter).

```
ServerPort=number
```

Here *number* is the port number used by the Tivoli Enterprise Console event server to listen for events. Specify 0 (zero) if the event server is using portmapper, which is typically the case.

4. Customize IBM Tivoli Enterprise Console to understand the events from NetView. This is done by importing the BAROC file into IBM Tivoli Enterprise Console, along with any rules that may have been provided with the adapter. See the NetView or IBM Tivoli Enterprise Console documentation for more details.

### **Configuring the NetView adapter**

Several files, listed in Table 6-2, are used by the NetView adapter.

*Table 6-2 NetView adapter files and their purpose*

File	Purpose
tecad_nv6k (UNIX) or tecad_nv6k.exe (Windows)	NetView adapter executable
tecad_nv6k.baroc	IBM Tivoli Enterprise Console class definitions for the default set of events generated by the adapter
tecad_nv6k.cds	Mapping of raw trap information to IBM Tivoli Enterprise Console event classes and attributes
tecad_nv6k.conf	Global configuration options for the adapter

File	Purpose
tecad_nv6k.err	Tracing configuration options
tecad_nv6k.lrf	Daemon registration file. As with all NetView daemons, the NetView adapter must be registered.
tecad_nv6k.oid	Mapping of SNMP object identifiers to names
tecSmartSetFilter.Conf (Windows only)	Smartsets for which associated traps are forwarded

We focus our discussion on configuring the NetView adapter to filter events. For additional information about other configuration options, see the product documentation.

### ***Filtering by smartset***

When using NetView for Windows, traps may be filtered based upon smartset membership for individual traps using the tecSmartSetFilter.conf file. This file can be initially configured for NetView traps from the TecConfigurator, and then modified manually to include additional traps from any enterprise.

Smartset filtering is based on nodes. Use only smartsets that have nodes as members. The node name is extracted from the second varbind for NetView enterprise traps and from the agent\_addr field for all others.

Each trap is grouped by enterprise. The first line of a group must begin with the token ENTERPRISE followed by the enterprise ID in a dotted decimal format. The enterprise line can be followed by any number of traps, one per line. Any number of smartsets can be listed following a trap specified by a Generic Id and a Specific Id. If a trap has no smartsets following it, that trap passes for all nodes.

The IBM Tivoli Enterprise Console adapter only processes these traps if they are also associated with a node in one of the listed smartsets, or if no smartset is listed. All traps that are not entered in this file automatically pass through to the regular IBM Tivoli Enterprise Console adapter filtering.

In Example 6-5, the Router Down trap from enterprise 1.3.6.1.4.1.2.6.3 (NetView) is sent only for the devices included in smartset routers. The Node Up trap is sent for devices in either the CriticalNodes or the DNSServers smartsets.

**Note:** The traps listed here must also pass the filtering defined in tecad\_nv6k.conf.

#### Example 6-5 Smartset filtering entries

---

```
ENTERPRISE 1.3.6.1.4.1.2.6.3      # NetView enterprise
6 58916971 Routers                # Router Down
6 58916864 CriticalNodes DNSServers # Node Up

ENTERPRISE 1.3.6.1.2
2 0 ImportantNodes                # Link down traps
6 1 WebServers
```

---

#### **Filtering in tecad\_nv6k.cds**

Traps that pass the pre-filter in NetView for Windows, and all traps from NetView for UNIX, are mapped to IBM Tivoli Enterprise Console events using the definitions in the tecad\_nv6k.cds Class Definition Statement file.

Unlike some of the other IBM Tivoli Enterprise Console adapters, the NetView adapter does not provide a method of generating a class definition statement (CDS) file from a format file. Any changes to the mapping of traps to IBM Tivoli Enterprise Console classes and slots must be coded using the somewhat cryptic syntax of this file.

**Note:** If you change the CDS file, you must verify the BAROC file entry for the event is still valid. Adding IBM Tivoli Enterprise Console classes, slots, or both to the CDS necessitates adding them to the BAROC file as well.

If you intend to customize events, it is useful to know that the following keywords may be used in class definition statements:

- ▶ **\$COMMUNITY:** Specifies the trap community string.
- ▶ **\$ENTERPRISE:** Specifies the enterprise object identifier of the object generating the trap.
- ▶ **\$SOURCE\_TIME:** Specifies the value of sysUpTime of the object generating the trap.
- ▶ **\$TYPE:** Specifies the generic trap type number (0 through 6).
- ▶ **\$SPECIFIC:** Specifies the enterprise-specific trap type number.
- ▶ **\$AGENT\_ADDR:** Specifies the address of the object generating the trap.
- ▶ **\$VARBIND:** Specifies a list of all non-fixed attributes.

Look at Example 6-6. The first entry applies to enterprise 1.3.6.1.4.1.2.6.3 (NetView). The specific trap number 58916864 is used to designate Node Up traps. The trap is mapped to class TEC\_ITS\_NODE\_STATUS. Plus, several IBM Tivoli Enterprise Console slots are set based on the NetView variables. For

example, the IBM Tivoli Enterprise Console slot msg is set to the fourth variable as defined in the SELECT section, or nvEventDescr.

It is interesting to note the differences between the two entries. The nodestatus and specific trap numbers are the only differences. This means that both Node Up and Node Down are mapped to the same IBM Tivoli Enterprise Console class (TEC\_ITS\_NODE\_STATUS), and the nodestatus IBM Tivoli Enterprise Console slot must be consulted when correlating the events.

---

*Example 6-6 Excerpt from the tecad\_nv6k.cds file*

---

text omitted

```
CLASS TEC_ITS_NODE_STATUS
  SELECT
    1: ATTR(=,$ENTERPRISE) , VALUE(PREFIX, "1.3.6.1.4.1.2.6.3" ) ;
    2: $SPECIFIC = 58916864 ;
    3: ATTR(=, "nvObject" ) ;
    4: ATTR(=, "nvEventDescr" ) ;
    5: ATTR(=, "nvAppINbr" ) ;
    6: ATTR(=, "VB_8" ) ;
  FETCH
    1: IPADDR($V3);
    2: IPNAME($AGENT_ADDR) ;
  MAP
    adapter_host = $F2 ;
    origin = $F1 ;
    hostname = $V3 ;
    msg = $V4 ;
    category = $V5 ;
    nodestatus = 1 ; # UP
    iflist = $V6 ;
END

CLASS TEC_ITS_NODE_STATUS
  SELECT
    1: ATTR(=,$ENTERPRISE) , VALUE(PREFIX, "1.3.6.1.4.1.2.6.3" ) ;
    2: $SPECIFIC = 58916865 ;
    3: ATTR(=, "nvObject" ) ;
    4: ATTR(=, "nvEventDescr" ) ;
    5: ATTR(=, "nvAppINbr" ) ;
    6: ATTR(=, "VB_8");
  FETCH
    1: IPADDR($V3);
    2: IPNAME($AGENT_ADDR) ;
  MAP
    adapter_host = $F2 ;
```

```

origin = $F1 ;
hostname = $V3 ;
msg = $V4 ;
category = $V5 ;
nodestatus = 2 ; # DOWN
iflist = $V6 ;
END

```

text omitted

---

If desired, you may change the IBM Tivoli Enterprise Console classes to be unique or use printf statements to format variables differently. VALUE clauses may be used on the ATTR keywords within the SELECT section of an entry as shown in Example 6-7. In this example, the trap is checked to ensure the ifDescr starts with Serial. The ifDescr is a name given to represent a MIB variable with SNMP object ID 1.3.6.1.2.1.2.2.1.2.

---

*Example 6-7 Using VALUE and comparative operators in the CDS*

---

```

SELECT
1: ATTR(=,"ifDescr"), KEY(!=,1),
VALUE(PREFIX,"Serial");

```

---

There is a corresponding definition in the tecad\_nv6k.oid file to define this mapping, as shown in Example 6-8.

---

*Example 6-8 Mapping of MIB variable to name in tecad\_nv6k.oid file*

---

```

# STANDARD MIB
#"mib-2" "1.3.6.1.2.1
"sysUpTime" "1.3.6.1.2.1.1.3"
"ifIndex" "1.3.6.1.2.1.2.2.1.1"
"ifDescr" "1.3.6.1.2.1.2.2.1.2"
text omitted

```

---

See the *IBM Tivoli Enterprise Console Adapters Guide, Version 3.9*, SC32-1242, for more information about coding the CDS file.

**Note:** If a trap does not match any entry in the CDS file, it is not forwarded to IBM Tivoli Enterprise Console. The default class, TEC\_ITS\_BASE, should catch any NetView traps that are not mapped to a more specific IBM Tivoli Enterprise Console class.

### ***Filtering in tecad\_nv6k.conf***

The event formed using the mapping defined in the CDS file is sent to the next filtering point, the tecad\_nv6k.conf file. If a trap has passed through the filters defined in the tecSmartSetFilter.conf file and has been successfully assigned an IBM Tivoli Enterprise Console class in the CDS file, it is subjected to the filters defined in this file.

The tecad\_nv6k.conf file has most of the same keywords as other IBM Tivoli Enterprise Console adapter .conf files. Specify FilterMode=IN to send only events matching filter, or FilterMode=OUT to excluding the listed events. Use filter statements to specify IBM Tivoli Enterprise Console classes and slots for the events that should be filtered in or out.

In Example 6-9, FilterMode=IN is specified. This means only the events which match one of the filter statements in the file are forwarded to IBM Tivoli Enterprise Console. All Link\_Down events are forwarded to IBM Tivoli Enterprise Console, but only TEC\_ITS\_INTERFACE\_MANAGE events with slot variable manage=2 (unmanaged) are forwarded.

#### ***Example 6-9 Excerpt from tecad\_nv6k.conf showing filtering options***

---

```
(text omitted)
EventMaxSize=4096
WellBehavedDaemon=TRUE
AdapterErrorFile=/usr/ov/conf/tecad_nv6k.err
BufEvtPath=%SystemRoot%/system32/drivers/etc/Tivoli/tec/nv6k.cache
BufferFlushRate=5

# The following events are the only ones that are forwarded

ServerLocation=tmeserver
ServerPort=0
FilterMode=IN
Filter:Class=Cold_Start
Filter:Class=Link_Down
Filter:Class=Link_Up
Filter:Class=Warm_Start
Filter:Class=TEC_ITS_L2_NODE_STATUS
Filter:Class=TEC_ITS_INTERFACE_ADDED;action=2
Filter:Class=TEC_ITS_INTERFACE_MANAGE;manage=2
Filter:Class=TEC_ITS_SA_STATUS
Filter:Class=TEC_ITS_INTERFACE_STATUS;ifstatus=1
Filter:Class=TEC_ITS_SERVICE_STATUS
(text omitted)
```

---

The FilterCache keyword can help to prevent traps from being cached in the event that IBM Tivoli Enterprise Console is unreachable. Use this keyword for traps that are time dependent, those for which action cannot be taken after a certain interval has passed. The syntax of the Class statement is the same for FilterCache as for Filter, FilterCache:Class= class\_name; attribute= value;...; attribute= value.

### ***Best practices for using the NetView adapter***

Recommendations for using the NetView adapter depend heavily on an organization's processing environment. Here are some general guidelines:

- For NetView for UNIX, use the NetView adapter if additional granularity of IBM Tivoli Enterprise Console classes and slots is necessary.

The reasons for additional granularity are covered in “Using the NetView adapter” on page 191. Use the SELECT section of the CDS entry to set the conditions for which the entry applies. Set the VALUE keyword on the ATTR statements to appropriate values. The value can be set to a constant if selecting based on the value of a trap variable. To parse a trap variable for strings, use CONTAINS, PREFIX, SUFFIX, and other comparative operators (=, !=, <, <=, >, >=) within the VALUE keyword.

- Filter the TEC\_ITS\_BASE class.

Since this IBM Tivoli Enterprise Console class is used as a catch-all for traps that are not mapped to specific classes, many undesired messages are assigned to the TEC\_ITS\_BASE class. Forwarding them to IBM Tivoli Enterprise Console clutters the console with unnecessary events.

Periodically review the trapd.log file in NetView to see if you are receiving traps that should be forwarded. Then ensure that they are properly mapped in the CDS file and forwarded with appropriate smartset pre-filters and tecad\_nv6k.conf filters.

- Consider using the adapter for communicating through a firewall.

IBM Tivoli Enterprise Console can be configured to listen for events on a specific port. Specifying this same port in the ServerPort parameter the tecad\_nv6k.conf file ensures that NetView uses it when forwarding events.

### **Using rule sets**

NetView comes with a rules engine that can take actions on events. We discuss the options for rules in more detail later in this chapter. In this section, we discuss only those pieces of rules that apply to filtering.

The NetView rules are configured on NetView for UNIX only, using the *ruleset editor*. The rules provide actions to take upon events. A default is set for the

event stream (pass or block). The action blocks within the rule further define event processing.

The rules can be created on a NetView for UNIX system and then used on a NetView for Windows machine. However, there are some limitations. Since NetView for Windows does not support some of the actions (namely, Action, Block, Override Status and Severity, Pager, Resolve, and Set State Nodes), do not use these in any rule sets that you will move to your NetView for Windows machine. In addition some functions, while supported on NetView for Windows, do not work the same way. For example, while In-Line Action nodes are supported on Windows, you cannot use them to launch a Windows console application (one which displays a graphical user interface (GUI) to the operator), since background processes, such as the correlation daemon, do not have console access in Windows.

The rules are used in several ways that relate to event management such as for console display and event forwarding. In the “Using nvserverd” on page 189, the daemon is configured by supplying a ruleset name. All events that are not initially dropped are subject to these rules to determine if they will be forwarded to IBM Tivoli Enterprise Console using nvserverd.

A second use of rules is for console display. Operators using the NetView display, either for problem determination or as their primary means of notification for networking events, can filter what is displayed based on the rules.

A third use is in ESE.automation. Any events received by NetView are subject to processing by the rules defined in this file. The rules can contain selection criteria to prevent processing from occurring for particular devices. This concept is discussed further in 6.9.1, “Using NetView for automation” on page 338.

The rules can be used effectively to filter out traps from certain devices. This can be accomplished by verifying membership in a smartset or checking the value of a trap variable. Consider using rules to filter based on business impact, filtering in the traps from certain devices, and excluding traps from others.

### **Additional best practices for NetView filtering**

We have presented ways in which traps can be filtered in NetView. Each organization should decide which traps to eliminate based on infrastructure, systems management tool usage, and best practices, as outlined in 2.3, “Filtering” on page 39.

However, there are certain traps that warrant special discussion. These traps are encountered in many environments that can cause NetView performance issues. Filtering recommendations are presented for these traps.



- Filter authentication traps from NetView's consoles.

An *authentication trap* is a message generated by a device to inform that it received a message with improper credentials. A typical cause of authentication traps is SNMP requests that specify an incorrect community name.

Often, the authentication trap does not contain the name or address of the device that made the invalid request or the community string that it was attempting to use and thus has little value. Also, support personnel dispatched to investigate an authentication trap sometimes err in believing the device reporting the problem is the one configured incorrectly, and do not determine why it is occurring.

The cause of an authentication trap is often an application configured to query a device using the wrong community string. The application (such as an SNMP manager like NetView or a custom script) may poll a device at regular intervals for performance data, resulting in a flood of authentication traps. These can clutter NetView's console and cause trapd.log to reach its maximum size frequently.

Ideally, these traps should be filtered at the source by configuring SNMP devices to suppress them. Inevitably, though, a networking device is deployed and configured to send authentication traps. If no one tries to access the device with the incorrect community name, there is no issue. However, if a program, such as an SNMP manager, attempts access, this can result in a flood of authentication traps.

Filtering the trap from NetView's console prevents these messages from obscuring meaningful traps. Logging it in trapd.log allows it to be available for debugging purposes, should NetView experiences performance degradation.

It may be argued that authentication traps represent security breaches, and as such, should be reported. However, an intrusion detection system is better equipped to assess security threats and is the preferred choice to report them.

- Filter Telnet session data.

Again, there may be large volumes of Telnet session data, depending upon the environment. These traps can flood NetView, obscuring real events and cluttering consoles. Since session data is needed primarily for security or auditing, it is best reported using a tool designed for those purposes.

- When using NetView as an intermediary event processor, pass events requiring action (including notification and trouble ticketing) to the IBM Tivoli Enterprise Console server and perform those functions there. Only perform the functions in NetView when it acts as the focal point in an event management hierarchy.

IBM Tivoli Enterprise Console generally has greater event processing functionality and capability than NetView while using fewer cycles for comparable function. Moreover, the cost of trap transmissions to the IBM Tivoli Enterprise Console server is minimal.

- Do not send Cisco syslog events sent to NetView as traps.

Cisco devices have the ability to send their syslog events as traps to an SNMP manager. This may seem like a good idea until the sheer volume of events chokes the NetView server.

It is better to use remote logging servers to which multiple Cisco devices log their messages. The IBM Tivoli Enterprise Console logfile adapters or event adapters running on the logging servers may then be configured to filter large numbers of unnecessary messages. This reduces the number of events received at NetView and minimizes bandwidth and cycles used, as well as adhering to the old adage, filter closest to the source.

Keep in mind that when using logging servers for syslog events, you may need extra logging servers across firewalls, depending on your environments security policies.

### **Filtering by limiting monitoring scope**

In its role as an event source, NetView monitors IP devices for status and performance and generates traps to report the errors it detects. Specifically, NetView periodically tests the interfaces on every monitored device to see if each one is reachable and issues a trap when one changes status. Likewise, if configured to do so, it checks performance-related MIB variables and sends a trap when thresholds are exceeded or re-arm. The traps are sent to the event processing code within NetView that performs the filtering and forwarding actions described previously.

By default, NetView discovers and monitors all the IP devices on the subnets it manages. This includes network equipment such as routers and switches, well as IP-addressable servers, printers, and workstations. Most organizations want to see events for only a subset of these devices. For example, user workstations may be rebooted many times during the day and powered off every evening. It is not practical to send events to the help desk or network operations center or open a trouble ticket to report every time a user workstation shuts down.

NetView can eliminate these unwanted traps by either preventing a device from being discovered and monitored or by unmanaging it.

Therefore, we recommend this *best practice*: When possible, prevent discovery of devices for which events are not desired.

While unmanaging devices prevents NetView from generating traps for them, there are reasons why it is better to prevent the devices from being discovered. NetView maintains information about each object it discovers within its databases. IP address, host name, subnet mask, number of interfaces, and vendor are a few of the many characteristics NetView records for each device. Discovering unwanted objects increases the size of the database, using storage and adversely affecting performance.

It is also difficult to ensure devices do not accidentally become managed. A NetView user may unmanage and remanage a network segment or object that contains the unmanaged device, inadvertently causing it to become managed. Therefore, unmanage discovered devices to prevent NetView from generating traps for them only if you cannot prevent their discovery.

**Note:** Limiting discovery prevents NetView from generating traps for devices. It does not affect NetView's ability to receive traps from those devices. If a device is configured to send its unsolicited traps to NetView, the traps are sent regardless of whether NetView discovers the device.

If you truly do not want any events for a device, ensure it is not configured to send its SNMP traps to NetView.

### ***Limiting device discovery***

During initial discovery, the scope of discovery can be limited. NetView allows limiting the discovery of network objects to those residing on:

- ▶ Its own subnet only
- ▶ The local backbone
- ▶ All networks within reach

This controls the number of objects that NetView discovers and manages. In most cases, the initial discovery options are global. This means that they do not limit any object type from being discovered.

A more granular way to control device discovery is through the *netmon seed file*. Originally introduced to support the automatic discovery of particular devices and network segments, it has been enhanced to provide a means of including or excluding network objects by SNMP object ID, IP address range, or individual device.

Code entries in the seed file according to Table 6-3. To pass complete discovery control to the netmon seed file, enter @limit\_discovery anywhere in the seed file. Use an exclamation mark (!) to exclude devices and address ranges.

Note that the seed file is read and interpreted only when netmon starts. If you make changes to the file, recycle the netmon daemon so they take effect. Keep in mind that this file defines the set of objects that you want to discover. If a node meets the criteria for discovery, it becomes part of your network topology. After it is discovered, you need to unmanage it or use other filtering methods to control the events for the node.

Table 6-3 Seedfile entries to control discovery

Example	Comment	Explanation
9.24.104.111	Single entry	To discover a single node, enter its name or IP address. These entries are single entries; no wildcards are allowed. If a name cannot be resolved via DNS or /etc/hosts, it is ignored.
9.*.104.1-100	Range using wildcards * and ?, where * resolves 0 to n characters and ? resolves a single character	The range of addresses is discovered.
@ oid 1.3.6.1.4.1.9..*	SNMP object ID prevents all Cisco devices from being discovered.	

See the NetView product manuals for details about these and other functions of the seed file.

In this case, we suggest the following *best practice*: Use the seed file to control discovery.

The seed file increases the likelihood that devices of interest are discovered and prevents unimportant ones from being found. Include routers, switches, key servers, and other network infrastructure devices in the seed file to encourage their discovery. Depending upon your IP addressing scheme, either include address ranges for important devices you want discovered, or exclude the ranges of addresses that apply to devices that you do not want to see.

**Unmanaging devices**

NetView objects can be unmanaged either through the NetView console or through several other ways. The most popular method involves using the NetView console. To do this, select the object or objects to unmanage multiple objects on the native NetView console. Then select **Options** → **Unmanage Objects**. This unmanages all selected objects and any accociated child submaps.

To suppress events from IP nodes that are unmanaged in the open map, in the Event Display window, select **Options** → **Unmanaged Nodes** → **Suppress Traps**. This menu option is a toggle button. To resume seeing the traps, select this menu option again.

## 6.1.2 Filtering and forwarding using IBM Tivoli Enterprise Console

This section explains the reasons and ways to filter and forward events using IBM Tivoli Enterprise Console.

### Logfile adapters

Usually, an adapter sends all events to the event server. You can optionally specify events that can or cannot be sent to the event server. You can do this by specifying the event class and such information as the origin, severity, or any other attribute=value pair that is defined for the event class. The class name specified for an event filter entry must match a defined class name; an adapter does not necessarily have knowledge of the class hierarchy.

You can try to filter events in the adapter to save processing and correlation time. Depending on how you specify the Filter and FilterMode keywords, filtered events are either sent to the event server or discarded.

To send specific events to the event server, set FilterMode to IN. To discard specific events, set FilterMode to OUT (the default value). Create Filter statements to match the specific events that you want discarded or sent, depending on your FilterMode keyword.

You can also use Tcl regular expressions in filtering statements. The format of a regular expression is `re: 'value_fragment'`.

For more information about how to use these functions, see the *IBM Tivoli Enterprise Console Adapters Guide, Version 3.9, SC32-1242*.

### Event buffer filtering

When an adapter is unable to connect to the event server or Tivoli Enterprise Console gateway, it sends the events to a file if the BufferEvents keyword is set to YES. You can filter events sent to a cache file, similar to filtering events for the event server by using the FilterCache keyword.

There are no default event cache filters in the configuration files shipped with adapters.

### Format files

A format file serves as the lookup file for matching messages to event classes. When the format file is used for this purpose, all format specifications in the file

are compared from top to bottom. In situations where there are multiple matching classes for a message, the last matching format specification is used. If no match is found, the event is discarded.

A format file also serves as the source from which a CDS file is generated.

### ***Class definition statement file***

CDS files are used by an adapter to map incoming raw events to a particular class and to define event attributes before forwarding the event to the event server. No alterations to this file are necessary to use an adapter unless you alter the corresponding .fmt file (if any). If any event definition is changed in a CDS file, the corresponding event class definition in the BAROC file may also need changing.

## **IBM Tivoli Enterprise Console gateways**

IBM Tivoli Enterprise Console gateways should be placed as close to their corresponding event sources as possible. Depending on the amount of processing required for each gateway, including the new features of state correlation, if enabled, you may need to investigate the performance of each gateway to ensure that it is capable of managing the number of events and their respective correlation that it receives from its various event sources.

During our laboratory examples, we did not find or investigate any performance issues caused by the enablement of state correlation on an IBM Tivoli Enterprise Console gateway. Of course, our environment does not map exactly to your existing environment, and we did not run performance tests or scalability tests during our exercises. You may need to investigate this feature of the IBM Tivoli Enterprise Console gateways to ensure that you do not experience problems in a production environment.

**Note:** IBM Tivoli Enterprise Console Gateway state correlation can apply to all NetView, TME, and non-TME events received by the IBM Tivoli Enterprise Console gateway.

### ***Match rules***

Matching rules are stateless. This means that they perform passive filtering on the attribute values of an incoming event. A matching rule consists of a single predicate. If the predicate evaluates to true, the trigger actions, which are specified in the rule, are executed.

For example, a good practice is to create a state correlation rule, with a match predicate, to filter events from default root event classes such as EVENT or NT\_BASE.

If you choose to implement a rule such as the one shown in Example 6-10, you must install it on every IBM Tivoli Enterprise Console gateway to prevent unwanted events from being forwarded to your IBM Tivoli Enterprise Console servers.

*Example 6-10 State correlation rule*

---

```
<rule id="rootClasses.matchAndExclude">
  <eventType>EVENT</eventType>
  <eventType>NT_Base</eventType>

  <match>
    <predicate>
      <![CDATA[
        true
      ]]>
    </predicate>
  </match>

  <triggerActions>
    <action function="Discard" singleInstance="true"/>
  </triggerActions>
</rule>
```

---

**Note:** State correlation rules are created using Extensible Markup Language (XML) syntax version 1.0. Follow the document type definition (DTD) provided in the \$BINDIR/TME/TEC/default\_SM/tecse.dtd file.

You can find more information about writing state correlation rules in the *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9, SC32-1234*.

## IBM Tivoli Enterprise Console rules

It is possible to perform event filtering at the IBM Tivoli Enterprise Console event server, using the traditional IBM Tivoli Enterprise Console rules and rule sets in prolog at each event server. However, it is a best practice to filter as near to the source as possible. Use filtering at the event server only for events that cannot be filtered elsewhere, for internally generated events and for security purposes.

### ***Events that cannot be filtered nearer to the source***

Sometimes it is impossible or requires too much work to filter an unwanted event before it arrives in the event server. For these cases, you can easily use the out-of-the-box event filtering rule set. It gives fast results, without overwhelming the server processing.

### ***Internally generated events***

Sometimes IBM Tivoli Enterprise Console generates internal events, and you may not want to receive some of them. For example, if a rule sends TEC\_Notice events within different severities and you use severities greater or equal to WARNING for your correlations, you may want to filter events with severity equal to HARMLESS. In this case, create an entry in the event filtering event\_filtering.rls file to drop that events of class TEC\_Notice and severity HARMLESS, as shown in Example 6-11.

#### ***Example 6-11 Drop event of class TEC\_Notice and severity HARMLESS***

---

```
create_event_criteria(harmless_tec_notice,%event criteria name
    'TEC_Notice',                %class to filter on
    yes,                        % fire on non-leaf only (yes/no)
    [ ['severity', within, ['HARMLESS']] ] % criteria based on slots
    ],
    % record all criteria elements into the even_filter_criteria record
    record(event_filter_criteria, [harmless_maintenance,
        harmless_heartbeat,
        harmless_tec_notice])
```

---

### ***Avoiding unwanted unsecure events***

Another type of event to filter is events from unknown sources that can may use breaches in the IBM Tivoli Enterprise Console system to possibly cause over processing or even worst effects. You can use two different approaches to filter these events:

- ▶ Writing rules that filter events from unknown sources
- ▶ Changing the default IBM Tivoli Enterprise Console configuration to try to close breaches

An example for the first approach is to create an entry, in the event filtering rule set, to accept events only from known sources or known hosts.

For the second approach, there are some ideas that can be used to try to close or at least make it more difficult to find breaches:

- ▶ Change the default reception port on the server (5529) and gateway (5539).
- ▶ Disable non-TME events in the gateway.
- ▶ Enable Tivoli Management Framework Security, since IBM Tivoli Enterprise Console uses it for its communications.



You can use these recommendations separately or in conjunction to filter events. Depending on your environment, you may have more or less security issues.

### ***Event filtering using event\_filtering.rls***

The event filtering rule set contains rules that filter out unwanted events based on customizable criteria. Filtered events do not appear at any console and are not stored in the event cache. The most recommended cases to use this rule set are the preceding sections. Here are some tips for using this filtering option:

- ▶ The event\_filtering.rls rule set is inactive by default. You must activate it on the event server.
- ▶ Import this rule set to the target event server before all other rule sets, since this prevents unnecessary processing of unwanted events on the event server.

To activate the rule set in your rule base, follow these steps:

1. Using a text editor, modify the corresponding statements in the rule\_sets file. Specify the active or inactive keyword as appropriate.
2. Use the **wrb -imptgtrule** command to import the event filtering rule set into the event server target as shown in this example:

```
wrb -imptgtrule event_filtering -before maintenance_mode EventServer testRb
```

3. Recompile the rule base using the **wrb -comprules** command.
4. Reload the rule base using the **wrb -loadrb** command.

### **Console filtering**

Another way to construct filters in an IBM Tivoli Enterprise Console environment is through consoles. With this option, filters can be created for viewing events, so each operator or group of operators has a view with its related events. This helps operators while working with events by preventing them from viewing unnecessary events. Inside the console events can be filtered and separated into groups, called *event groups*. The event viewer provides one last level of filters online.

#### ***Console***

Best practice says that each person should see only the events that requires an intervention from them. Therefore, event consoles should be created for each operator or group of operators with those events for which they are responsible.

#### ***Using event groups***

Use event groups to filter events in different groups. This offers a quick view of the event activity of all event groups for an event console, making it easier to view the status of different applications, networks, or systems.

### ***Using console filters***

Use console filters to filter the events in the working queue based on severity, status, and operator ownership, to help you focus on important events. When you filter events inside one event group, other event groups are not affected.

## **6.1.3 Filtering and forwarding using IBM Tivoli Monitoring**

IBM Tivoli Monitoring is an agent used to monitor systems for certain conditions, and as such, an event source. There are a few ways to configure it to filter unwanted events from occurring.

### **Only deploy resource models of interest**

If you are using NetView, for example, to do network performance monitoring, you may not want duplicate information from IBM Tivoli Monitoring. Therefore, you do not use the network interface resource model.

### **Forward events of interest and suppressing others**

IBM Tivoli Monitoring has the means of filtering before an event is even sent to IBM Tivoli Enterprise Console. When you configure your IBM Tivoli Monitoring Resource Models, set them to send an event only after all of your requirements for defining that problem are met. You can set up your resource models to only send an event after a problem occurs based on an algorithm which you specify using holes and occurrences, or only forwarding an event when a certain threshold is met.

When you set this up, consider that, when you send an event, you only want to send events for problems that require action.

You can either configure this in the default Resource Models that are supplied with IBM Tivoli Monitoring or by modifying the profiles through the Tivoli Framework desktop as shown in Figure 6-9.

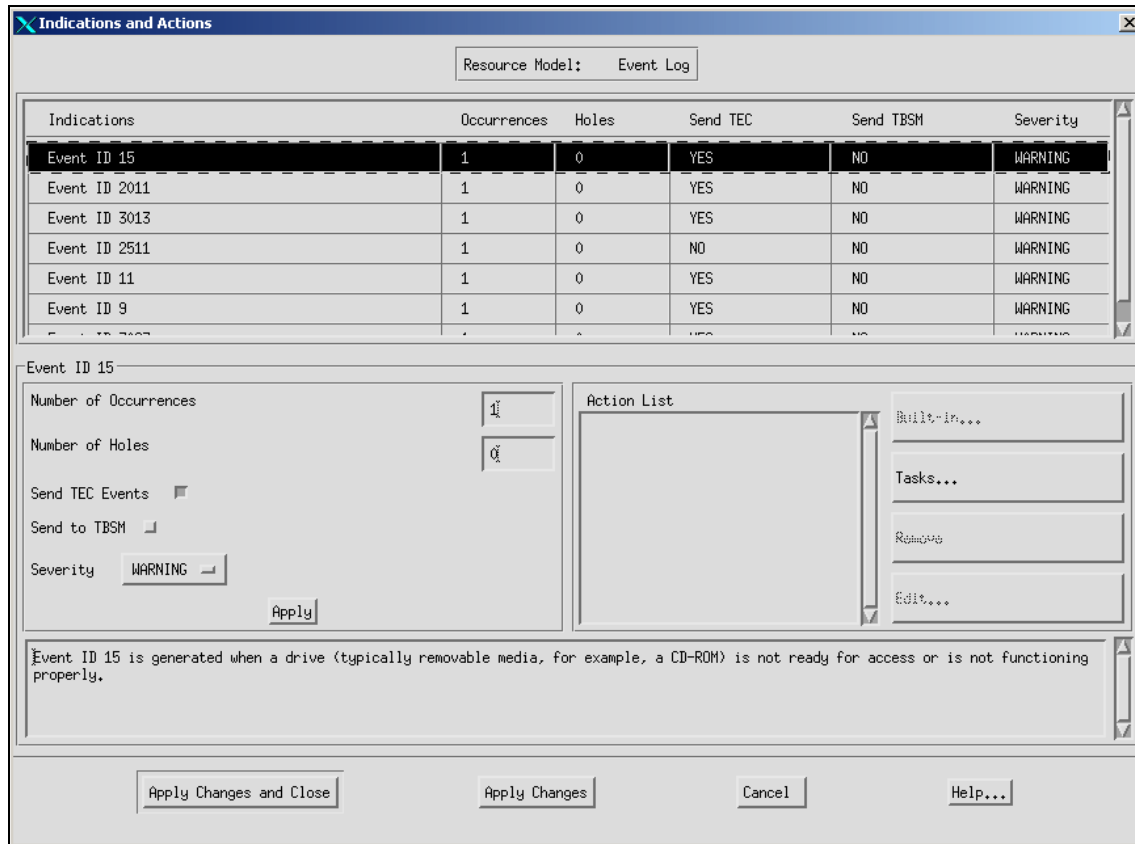


Figure 6-9 IBM Tivoli Monitoring resource model profile

Optionally, you can modify the definition of the resource model via the Resource Model Builder (RMB) utility provided with IBM Tivoli Monitoring. Using this option only changes the default actions of the resource model. The Tivoli Administrator can still change those attributes via the Tivoli Desktop within the IBM Tivoli Monitoring resource model profile.

### Separate profiles to limit which machines send which messages

If you are interested in seeing a certain event type from only a subset of machines, create a separate IBM Tivoli Monitoring profile for distribution to those machines. Configure the appropriate resource model to send the event in the profile distributed to those machines and to not send an event when defined in other profiles distributed to other machines.

## 6.2 Duplicate detection and throttling

Always perform duplicate detection and throttling as close to the source as possible.

### 6.2.1 IBM Tivoli NetView and Switch Analyzer for duplicate detection and throttling

IBM Tivoli NetView and Switch Analyzer have certain ways to assist in preventing duplicate events and helping with throttling via the following methods:

- ▶ NetView does state changes, which prevents duplicates.
- ▶ SNMP devices usually send state changes as well.
- ▶ May still receive two messages from different SNMP sources (for example, NetView and the router itself) for the same problem. This can be handled by suppressing one all the time, or handling through correlation.
- ▶ Mid Level Managers (MLMs) can throttle.
- ▶ NetView can throttle through rules, if necessary (pass on match).

### 6.2.2 IBM Tivoli Enterprise Console duplicate detection and throttling

This section discusses duplicate detection and throttling for each level of IBM Tivoli Enterprise Console event management, from the source (logfile adapter) to the event server.

#### Logfile adapter

The logfile adapter cannot detect duplicate events and should not be used to try to do this. Duplicate detection at this level depends on the source and if it detects and throttles duplicate events. For example, some versions of UNIX syslog detect consecutive duplicate events and show only the first. The logfile adapter sends the events in the way that they are in the source. In this case, it just sends the first event in the sequence.

#### Gateway

Use IBM Tivoli Enterprise Console gateway to throttle events before they go to the event server. This *best practice* prevents the event server from receiving bursts of events, through event summarization and primary correlation near the source. A situation where you should use IBM Tivoli Enterprise Console gateway for throttling events is for peak events. See 6.2, “Duplicate detection and throttling” on page 212.

In the following example, the fictitious company XYZ has a monitor on a Windows 2000 box that triggers every time that the processor percent utilization is over 95%. There are two problems with these events. First, one single event is not meaningful, since it's normal to have CPU over utilization in peak moments. The second problem is that in certain periodic times all the servers are stressed in their use, because of end of month processing, for example. These generate bursts of events and sometimes overload the event server.

The first problem was addressed in previous versions of IBM Tivoli Enterprise Console with the event\_thresholds.rls rule set. The second problem can still cause problems, which is why event throttling in the gateway, with state correlation, is better.

**Note:** The event\_thresholds.rls rule set can and must still be used, but just for the events that cannot be correlated with state correlation in the gateway, or for further correlation.

To solve these problems, the state correlation rule shown in Example 6-12 was created using the threshold predicate.

---

*Example 6-12 State correlation rule using the threshold predicate*

---

```
<rule id="nt.threshold_utilization">

  <eventType>NT_Processor_Util_Perc</eventType>

  <!-- I 'm only interested when at least 20 NT_Processor_Util_Perc events
happen for the same host within 10 minutes. -->
  <threshold thresholdCount="20" timeInterval="600000"
triggerMode="firstEvent">
    <cloneable attributeSet="hostname"/>
    <predicate>
      <![CDATA[
        # always succeeds.
        true
      ]]>
    </predicate>
  </threshold>
  <triggerActions>
    <action function="TECSummary" singleInstance="false"/>
  </triggerActions>
</rule>
```

---

In Figure 6-10, you can see events that arrived after state correlation. The events with the class NT\_Processor\_Util\_Perc are summarized. Instead of having one event every five seconds (frequency that the Windows NT® monitor triggers) in the worst moments, you have one event every 100 seconds. Note that after the configured number of occurrences is reached and the action is triggered, the counter resets and can trigger again if the error condition persists.

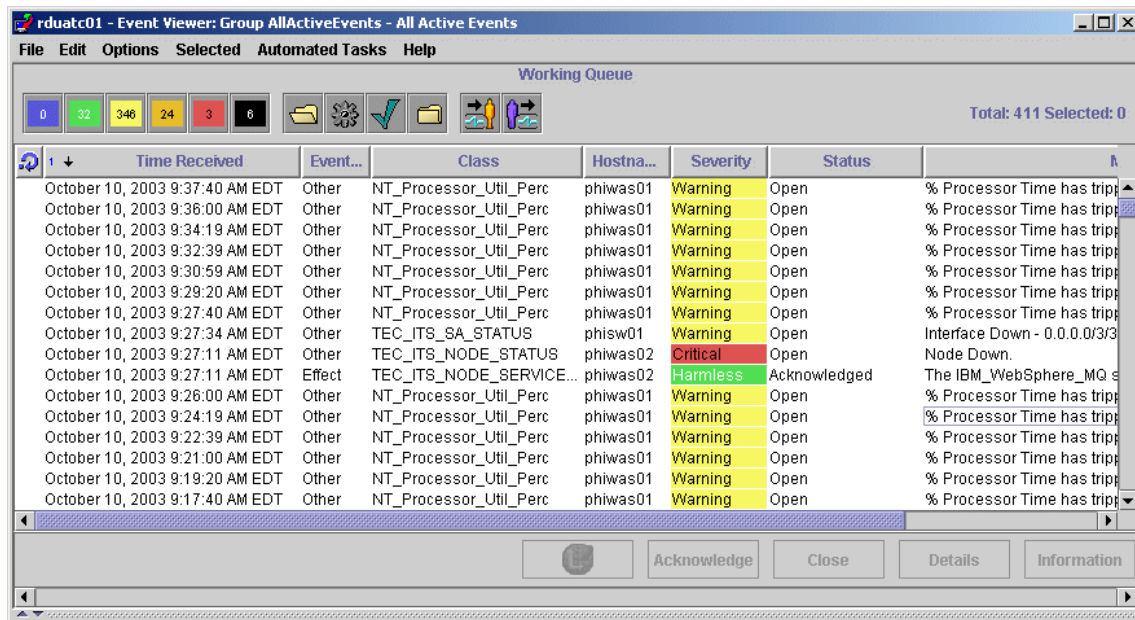


Figure 6-10 IBM Tivoli Enterprise Console Event Viewer

This example describes the state correlation threshold predicate, with the parameters triggerMode="firstEvent" and timeIntervalMode=fixedWindow (default). Changing these parameters changes the way it works. The sending modes specified by the triggerMode attribute are described in the following sections.

### ***firstEvent***

This mode sends the first event received during the time window.

### ***lastEvent***

This mode sends the last ( $n$ th) event received during the time window.

### ***allEvents***

This mode sends all events 1 through  $n$ , This is the default mode.

## forwardEvents

This mode sends all events after the  $n$ th event until it resets.

Use the allEvents and forwardEvents with care. In some cases, using these sending modes is almost the same or even the same as not correlating.

Figure 6-11 shows the same example that we described earlier, but with the forwardEvents send mode. It demonstrates a bad use of this parameter. As you can see in the figure, after the threshold is triggered, all events are sent. A burst of events can generate problems in higher levels of event management.

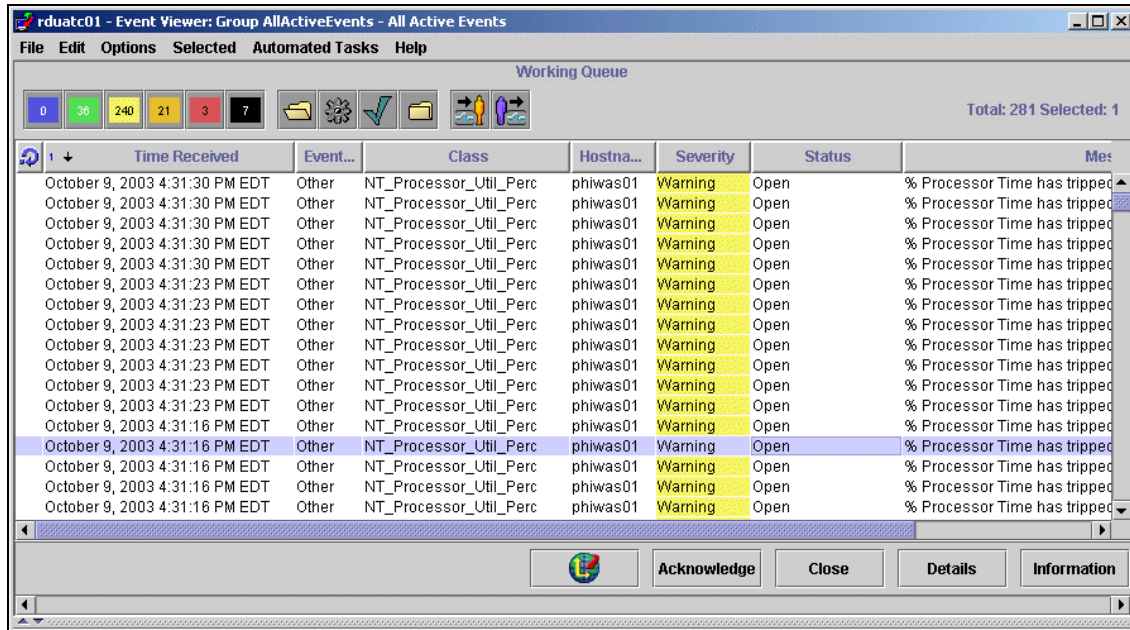


Figure 6-11 forwardEvents send mode

The time interval mode parameter indicates whether the time interval is fixed or sliding:

- **fixedWindow:** The timer starts when the first matching event arrives and closes after the time interval expires. In order for the threshold to be triggered,  $n$  matching events must be received within that time window. If the threshold is not reached within the specified time window, the rule resets, and the next subsequent matching event starts a new time window.

With a fixed window, the threshold is triggered if  $n$  events arrive within  $t$  seconds of the first event.

- ▶ **slideWindow:** A separate timer is effectively started when each matching event arrives, resulting in overlapping virtual time windows. In this case, the threshold is triggered if  $n$  matching events arrive within any time window.  
With a sliding window, the threshold is triggered if  $n$  events arrive within  $t$  seconds of each other.

The *duplicate* and *collector* rules are used for duplicate detection. You should also use them to reduce the amount or summarize the events that go to the events server. Both are powerful options, but work in a slightly different way. With duplicate rules, the first event is sent to the event server and the events that follow are suppressed. In collector rules, all events are buffered and then, after a determinate amount of time, are sent summarized to the event server.

You can find examples of these functions in the *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9*, SC32-1234.

## Event server

In the event server, duplicate events are considered the event instances of the same class that have the following characteristics:

- ▶ The same values for all attributes are defined with the dup\_detect facet set to YES.
- ▶ If there are no attributes defined with the dup\_detect facet set to YES, all events of that class are duplicates.

Generally, duplicate events are not kept in the event database, but are instead used to increase the severity of the event or to count the number of times the event has been received. Duplicate events are managed with the first\_duplicate and all\_duplicates predicates.

### ***dup\_detect***

This facet defines the criteria to determine whether two events are the same, indicating duplicates of each other.

**Note:** Setting the dup\_detect facet only provides a definition. You must create rules to test for duplicate events and specify the actions to take when they're detected by rule processing.

Two events are considered duplicates if they have the same values for all attributes defined with the dup\_detect facet set to yes and if they are of the same event class. For example, assume the following event class definition:

```
TEC_CLASS:  
Person ISA EVENT
```



```

DEFINES {
name:STRING,dup_detect=yes;
city:STRING,dup_detect=yes;
employer:STRING;
hobbies:STRING;
};

```

The following events are considered duplicates because the attribute values of their respective name and address attributes are the same (assuming both events are of the same event class):

```

<"Joe","Lafayette","ABC Widgets","Computers">
<"Joe","Lafayette","XYZ Widgets","Ham Radio">

```

By default, dup\_detect is no.

### 6.2.3 IBM Tivoli Monitoring for duplicate detection and throttling

An *event* is a change in the status of a resource. Within IBM Tivoli Monitoring Version 5.1.1, an event notifies you that a specified resource state is abnormal or problematic. In the workbench, a distinction is made between an indication and an event.

An *indication* is generated when the state of a given resource meets specific criteria you have defined. However, an indication does not trigger any action. Only when indications are aggregated do they become an event. The cycles during which the indication is generated are called *occurrences*. The cycles during which no indication is generated are called *holes*.

Only events can trigger some actions, notify that there is a problem in your resource state and, if enabled, send notification to the IBM Tivoli Enterprise Console server and IBM Tivoli Business Systems Manager.

Assume that there is a resource property that changes its value rapidly. The decision tree is visited every cycle. As a part of this, the value of the resource property is retrieved. In Figure 6-12, the vertical dashed lines represent the moments of the queries. The point at which the dotted lines meet the graph are values that are the results of the inquiries. The one horizontal dashed line represents the threshold. Values above that line are considered potential problems and trigger an indication.

Every time the values of the resource properties exceed the thresholds, the Resource Model generates an indication. If the value of the resource property drops below the threshold for a particular cycle, then no indication is generated, and the event aggregator counts the non-occurrence of an indication as a hole.

If we define an event as four occurrences and one hole when we configure our profile, the event is generated when the fourth indication occurs. If we have two consecutive holes, then the occurrence count is reset to zero and a clearing event is sent if it is configured to send a clearing event.

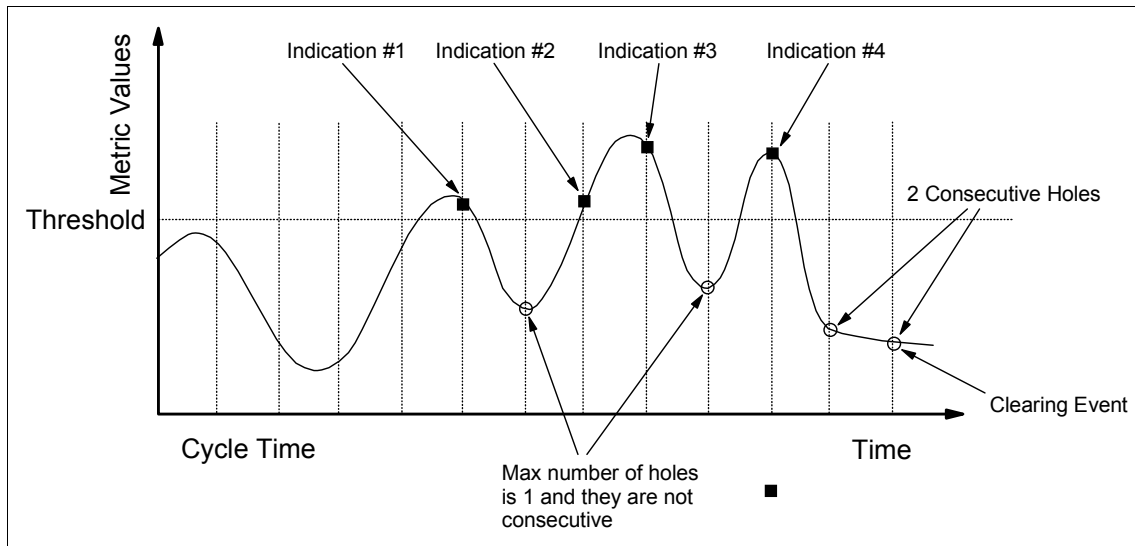


Figure 6-12 IBM Tivoli Monitoring holes versus occurrences chart

## 6.3 Correlation

This section focuses on performing correlation with IBM Tivoli NetView, IBM Tivoli Switch Analyzer, IBM Tivoli Enterprise Console, and IBM Tivoli Monitoring.

### 6.3.1 Correlation with NetView and IBM Tivoli Switch Analyzer

We now focus on the available methods of correlation with NetView and IBM Tivoli Switch Analyzer.

#### Router Fault Isolation (RFI)

NetView performs status polling for the machines it is managing using either Internet Control Message Protocol (ICMP) pings or SNMP queries. The intervals at which it polls is set for individual or groups of devices or by default in the `/usr/OV/conf/ovsnmp.conf` file. Traditionally, if NetView did not receive a response to its status poll, it marked the node or router down and issued a down trap.

Often during a network failure, the path from the NetView server to portions of the network is broken. Prior to router fault isolation, NetView attempted to poll the devices in the unreachable part of the network and generated down traps when they did not answer. This resulted in many segment, node, and interface down traps, particularly in networks with a large number of nodes on the far sides of routers. When the failure was corrected, NetView generated numerous up traps for each device it could again successfully reach.

This plethora of events had several drawbacks:

- ▶ Increased the difficulty of determining the original cause of the network failure
- ▶ Slowed network traffic considerably with the large number of status polls to the occluded area
- ▶ Created performance problems and unreliable status reports if the events were forwarded to the IBM Tivoli Enterprise Console and IBM Tivoli Enterprise Data Warehouse

### ***RFI overview***

The RFI function rectifies these problems. When NetView detects a node or interface is down, RFI first checks the status and accessibility of the router interfaces connected to the subnet on which the node or interface resides. During the router check, each interface and its subnet are analyzed. An unresponsive interface triggers checks of the interface and any connecting routers.

RFI generates appropriate Router Down or Router Marginal traps for conditions detected. It also simplifies the notification action by issuing one summary alert identifying the router nearest the fault.

When active, the Router Fault Isolation feature generates the events shown in Table 6-4 to alert users to important status changes.

*Table 6-4 Router fault isolation events*

Event	Network status
Router Marginal	At least one router interface is down. At least one other interface on that router is up.
Router Down	All interfaces are not responding, but at least one connected subnet is reachable. (The router is not in an occluded region.)
Router Unreachable	The network management workstation cannot query the router because it is an occluded region.
Router Up	All the interfaces have responded successfully. This event is issued on initial discovery and following a recovery from one or more interfaces being down.

Event	Network status
Network Unreachable	All router interfaces in the subnet have stopped responding.
Network Reachable	After one interface is successfully polled, the network is once again reachable.

The NetView maps display unreachable networks and router nodes or interfaces as white symbols. Note that non-router nodes and interfaces in unreachable subnets are not changed to Unreachable (white).

When NetView is used to manage a network with a high proportion of nodes to routers, Router Fault Isolation can significantly reduce the number of Node Down events that are false alarms. Router Fault Isolation detects which nodes are actually down and which nodes are simply unreachable because the router fault occludes them from the management station. Router Fault Isolation relies on connectivity tests and responds instantly to dynamic routing changes.

Router Fault Isolation also suppresses polls and status events for all non-router nodes and interfaces in unreachable subnets. After a partition is repaired, the first successful status poll from inside an unreachable subnet triggers a recovery. To speed the initiation of recovery, you can also manually ping any node in an unreachable region.

To reduce the false signals from a Node Down event for a device in an area with Unreachable status, NetView does not generate Node Down or Interface Down events for any node in the area with Unreachable status. The first Interface Down event that triggers an evaluation that results in declaring that the status of the subnet is Unreachable is also suppressed. Status polling to end nodes in subnets with Unreachable status is suppressed by default.

If NetView is not managing any routers in a particular subnet, then NetView can determine when that subnet is unreachable. It does this using a probabilistic algorithm, which determines when it is highly likely that the subnet is unreachable. NetView automatically uses this algorithm for subnets where there are no managed routers. However, this algorithm only determines the reachability of the subnet. If it is unreachable, Node Down and Interface Down events, including the first event, are not generated.

Router Fault Isolation and Mid-Level Manager can function together and object status should remain accurate. However, in some cases, both could poll the same nodes. This causes extra processing because routers are polled by both. In some situations, this causes extra network traffic.

## ***RFI configuration***

To configure the RFI mode, use the Server Setup application. Select **Configure → Set options for daemons → Set options for topology, discovery and database daemons → Set options for netmon daemon and set the Router Fault Isolation Mode.**

There are three modes that you can configure for RFI:

- ▶ **Disabled mode:** No attempt is made to determine the reachability or root cause. Routers generate node status events, instead of the root cause router status events.
- ▶ **Router Fault Isolation mode:** By dynamically evaluating the status of routers, NetView determines the reachability of subnets and the root cause of the partition or problem.
- ▶ **Probabilistic mode:** By dynamically evaluating the status of members of a subnet, NetView determines whether it is more likely that the subnet itself is unreachable or whether the devices are down. This mode is disabled if the subnet contains less than a configured number of managed devices. This mode is automatically used for subnets with no managed routers if the RFI mode is active. You can fine-tune this algorithm using the properties defined in the netmon.conf configuration file. See the /usr/OV/conf/netmon.conf file for more information.

The Server Setup application provides the ability to treat ambiguous router interfaces that are not responding in unmanaged subnets as though their status is either Unreachable or Down. In the Set options for netmon daemon window of Server Setup, set the Router Fault Isolation: Treat ambiguous nonresponding router interfaces in unmanaged subnets as field to either Unreachable or Down. See the /usr/OV/doc/RouterFaultIsolation.htm file for information about using this option.

## ***Stopping the RFI feature***

The Router Fault Isolation feature is active by default in NetView. To disable this feature, use the -K 0 option for netmon. To control the suppression of polling traffic to routers (including unreachable routers), use the -k option for netmon. Refer to the netmon man page for more information. For a detailed explanation of Router Fault Isolation, see the description in the /usr/OV/doc/RouterFaultIsolation.htm file.

## ***NetView RFI***

This document provides a detailed description about the NetView Router Fault Isolation feature, an explanation of how Router Fault Isolation works, what events are generated to isolate root cause, and how to understand switch and ambiguous cases. It ends with a section on netmon configuration for RFI.

This feature addresses two problems:

- ▶ The identification of the root cause
- ▶ Suppression of unnecessary events and network polling after a partition

### ***Understanding the problem***

When a network failure occurs and the NetView management workstation cannot reach devices or device interfaces in the failed portion of the network, that portion of the partitioned network is considered to be unreachable. For example, when one or more router interfaces fail, they can occlude a portion of the network, such as other subnets. This makes these subnets invisible, inaccessible, or unreachable due to breakdown or blockage in the connectivity at a point between the network management workstations and the devices themselves.

When a router or router interface fails and occludes part of a network from NetView, NetView reports many Interface Down and Node Down events for all interfaces and devices that it can no longer reach. It also reports Segment Down and Network Down events for the segments and subnets that it can no longer reach. After the failure is corrected, NetView reports another set of events to indicate that the interfaces and devices are back up again and that the segments and networks are accessible. The NetView events display can then display these events. NetView can also forward these events to IBM Tivoli Enterprise Console, if you configure NetView for this.

Not only does this proliferation of events make it difficult to determine the root cause, the large number of status polls to the occluded area can slow network traffic considerably. If NetView also forwards these events to IBM Tivoli Enterprise Console, the events can cause performance problems and unreliable status reports.

You can use RFI to detect occluded portions of the network, namely one or more subnets. RFI marks these on the NetView map and identifies the root cause by reporting a Router Down or Router Marginal event. It suppresses polls and events for all non-router nodes and interfaces that are occluded. It also suppresses events from the polling of router interfaces.

RFI works well at suppressing events when NetView manages a high proportion of nodes to routers. This significantly reduces the proportion of Node Down events that represent false alarms.

In addition, RFI generates an event for each Router Down and Router Marginal event. It also relies on connectivity tests, so it responds instantly to dynamic routing changes.

### ***How a partition is identified***

When NetView detects that a node or an interface is down, RFI checks the status of router interfaces. It starts with those that are connected to the subnet where NetView detected the Down status.

If all the router interfaces of that subnet are non-responding, RFI concludes that the subnet is Unreachable. It then dynamically analyzes the routers that reported a non-responding interface. The router analysis, for each interface, checks both the status of the interface and the reachability of the subnet to which it is connected. During this check any non-responding interface triggers a check on its subnet, which in turn trigger checks of other routers showing a non-responding interface. In this way, the analysis rapidly spreads determining the extent of the partition or partitions.

### ***How a router is analyzed***

During the router check mentioned in the previous section, each interface and its subnet is analyzed. If all the subnets to which it is connected are Unreachable, the router is also considered Unreachable. If at least one subnet is found to be reachable, then the router is considered Up, Marginal, or Down depending on the status of the interfaces. If at least one interface is Up and at least one is Down, then it is Marginal. If no interface is Up, then the router is considered Down.

Therefore, if a router is marked as Marginal or Down, then because it is reachable, the problem has now been isolated to the area of this router.

### ***How the partition appears on the map***

With the new status, NetView displays in white those unreachable networks, router nodes, and router interfaces. All other symbols in the occluded submaps are unaffected, and their map and object status remain unchanged.

To see the extent of the partition or partitions, look at the top-level maps showing the router and network symbols. Symbols in the occluded regions appear in white. Due to performance considerations, only network, router, and router interface symbols turn white. At the segment level, only router symbols appear white to indicate that the subnet is unreachable and all nodes are occluded.

### ***How a recovery is triggered***

After a partition is repaired, the first successful status poll from inside an Unreachable subnet triggers a recovery. The algorithm follows a similar subnet-router-subnet proliferation approach used to discover the partition. In this way, the recovery rapidly and proactively spreads resetting the status of the network to again reflect the propagation of the network's member interfaces.

### ***Suppressed polling in a partition***

During a partition, all polling, ICMP and SNMP, is suppressed to non-router nodes in Unreachable subnets. For Unreachable routers, configuration polling is suppressed, but by default, status polling is not.

In V6.0, RFI does not suppress poll traffic for Unreachable routers. In V6.0.1, you can use a switch in netmon to suppress poll traffic to occluded routers. See “Netmon configuration” on page 225.

When all status polling to Unreachable regions is suppressed, automatic recovery depends on a successful status poll to a root cause router. To speed the recovery initiation, you can also manually ping any node in an Unreachable region.

### ***Understanding switch and ambiguous exceptions***

This section helps understand the effects of switches, bridges and other layer 2 connectivity devices and unmanaged subnets.

#### ► Layer 2 connectivity devices

RFI can determine only that a subnet is occluded. As a result, if a switch goes down and causes a segment within a subnet to become unreachable, RFI cannot determine whether any nodes are actually occluded that are shown as being down within that subnet.

However, RFI isolates the problem to a subnet and can determine when to look further inside the subnet. RFI identifies a root cause router. If this router is Marginal (at least one Up interface), then we know we can get to it. Therefore, the interfaces that are marked as Down must signal real problems and we need look no further.

If the router is Down, then it is possible that either the router is offline and is the real problem, or it is occluded due to a failure in connectivity in a layer 2 connection device.

#### ► Unmanaged subnets

If a router with one or more non-responding interfaces is connected to at least one reachable subnet, then we can conclude it is a root cause. That is, it is reachable, but it is also reporting a problem. However, if all the subnets a router is connected to are unreachable, we conclude the router is unreachable. Ambiguity arises when a router is connected to at least one unmanaged subnet and all the other connected subnets are marked as Unreachable. If an unmanaged subnet were really reachable, that would signal the router as the root cause.

By default, RFI treats unmanaged subnets as unreachable in ambiguous cases. This allows the router to be marked as Unreachable if all the other managed subnets are Unreachable. If NetView is connected to this router via



one of the managed Unreachable subnets, it makes sense that this router is also Unreachable.

However, consider the case where NetView is connected to this same router solely via one of the unmanaged subnets and that interface was actually the problem and caused the occlusion of the router. All the interfaces are non-responsive and all the managed subnets are Unreachable, resulting in this router being declared Unreachable. Note that if NetView has an alternative connection to this router, the other subnets are not Unreachable and the root cause can be identified. To ensure cases like this are identified as a root cause, you can choose from two courses of action:

- For each router, ensure that any unmanaged subnet is not the only path to the network management station.
- Set the RFI option for netmon (-n) to treat unmanaged subnets in the ambiguous case as Reachable. See “Netmon configuration” on page 225.

### ***Netmon configuration***

The following netmon switches control aspects of RFI. These switches are not available in the Server Setup GUI. To change the default behavior, you must add the desired switch to the netmon.lrf file and then enter the following commands:

```
/usr/OV/bin/ovstop netmon  
/usr/OV/bin/ovdelobj /usr/OV/lrf/netmon.lrf  
/usr/OV/bin/ovaddobj /usr/OV/lrf/netmon.lrf  
/usr/OV/bin/ovstart netmon
```

This is similar for Windows NT, except that you just change the directory slashes

### ***Enabling and disabling RFI***

The RFI feature is active by default in NetView V6.0.1. If you run V6.0, contact your Tivoli Representative for a free script that enables this feature. If this feature is not desired, you may disable it by using the **-K 0** option for netmon as shown here (note to use an uppercase “K”):

```
-K 0 | 1
```

Note the following explanation:

- 0** Turn Event Suppression OFF
- 1** Turn Event Suppression ON. This is the default.

### ***Reduced polling to routers***

As a part of RFI, the new option **-k 2** has been added to netmon for V6.0.1 which prevents status polling to any router whose current status is Unreachable. If you use this option, you may need to manually ping a node in an occluded area to start recovery for some isolated routers.

This option has following syntax (note to use a lowercase “k”):

-k 0 | 1 | 2

Note the following explanation:

- 0 Don't suppress any pings or SNMP requests.
- 1 Suppress pings and all SNMP requests to non-routers in Unreachable subnets. This is the default.
- 2 Suppress pings and all SNMP requests to Unreachable routers.

### ***Handling the ambiguous case***

By default, if all the other subnets to which a router is connected are Unreachable, RFI considers unmanaged subnets as Unreachable. To change this default behavior so that unmanaged subnets in ambiguous situations are treated as reachable, set the -n option for netmon.

If the -n switch is present, unmanaged subnets are treated as reachable in ambiguous cases. The default behavior, when the -n switch is not present, is to treat unmanaged subnets as unreachable in ambiguous cases.

### **Correlation using NetView rules**

NetView provides the capability of creating rule sets to correlate events or take action on them. The ruleset editor enables you to graphically create a rule set comprised of event-processing decisions and actions that are represented by icons (nodes).

There are two types of nodes. *Decision nodes* are used to test conditions, and pass or block the event based on the results of the test. *Action nodes* perform such functions as forwarding the event to the Event Display application, resolving it, executing a script or command, and issuing a call to a pager. Table 6-5 lists the available nodes.

Table 6-5 Available nodes for NetView event correlation

Node	Description
Action	Specifies the action to be performed when an event is forwarded to this node. Fields from the trap being processed are available as environment variables. The specified action can be any operating system command, the full path name of any shell script or executable, or any NetView command.
Block event display	Prevents events from being forwarded to the Event Display application. Use this node if you changed the default processing action to pass (forward) events to the Event Display application and you do not want to forward events that meet specific conditions. A trap that is processed through this node is marked so that it is not handled by the default processing action specified for the rule set.

Node	Description
Check route	<p>Checks for communication between two network nodes and forwards the event based on the availability of this communication. For example, you can use this node to check the path from the manager to a device before forwarding a node down trap.</p> <p><b>Note:</b> The check route node does not check the status of the node. It checks only the availability of the path to the node.</p>
Compare MIB variable	Compares the current value of a MIB variable against a specified value. When a trap is processed by this node, the ruleset processor issues an SNMP GET request for the specified MIB variable.
Event attributes	Compares any attribute of the incoming event to a literal value. You can use this node to check for events generated by a particular device.
Forward	Forwards the event to applications that have registered to receive the output of the rule set. A trap that is processed through this node is marked so that it is not handled by the default processing action specified for this rule.
Inline action	Specifies the action to be performed when an event is forwarded to this node. Unlike a command specified in an Action node, a command specified in an Inline Action node is not sent to the actionsvr daemon. Instead, the command is executed immediately, and processing continues to the next node if the return code of the action matches the return code you specify within the specified time period.
Override	<p>Overrides the object status or severity assigned to a specific event and updates applications that have registered to receive the output of the rule set. The Event Display application is registered to receive the output.</p> <p>For example, you can use this node to change to Major when a node down event is received for a router. Use this node with the Query database field node to override status or severity for specific device types.</p>
Pager	Issues a call to a pager that is defined in a NetView user profile. You should have already configured the paging utility.
Pass on match	Compares some attribute of the event being processed with an attribute of all traps received in a specified period of time.
Query database smartset	Tests whether the node is a member of the specified smartset and takes the specified action (forward or block) if it is.
Query database field	Compares a value from the NetView object database to a literal value or to a value contained in the incoming event. You can use this node to check if the originating device is a router.
Reset on match	Compares some attribute of the event being processed with an attribute of all traps received in a specified period of time. This node is similar to the Pass on match node. The exception is that, if a match is found, the event is not passed on to the next node in the rule set and processing stops.

Node	Description
Set database field	Sets the value of any NetView non-Boolean object database field. Fields that have TRUE or FALSE values cannot be changed.
Query global variable	Queries the value of the global variable that has been previously set using the Set global variable node.
Resolve	Forwards a message to all registered applications indicating that a previous event is resolved. By default, the Event Display application is registered to receive the output from rule sets. The receiving application determines how to handle a trap that has been forwarded from this node. This node is frequently used in conjunction with the Pass on Match node. You can use the Resolve node to delete an interface or node down event from the Event Display application when an interface or node up event is received. A trap that is processed through this node is marked so that it is not handled by the default processing action specified for the rule set.
Set global variable	Sets a variable for use within the rule set. For example, use this node to set a flag whose value is checked later in the rule set using the Query global variable node. When the rule set is finished processing, the global variable is no longer in effect.
Set MIB variable	Issues an SNMP SET command to set the value of a variable in the MIB representing any network resource. For example, you can use this node to change the system contact for a particular device.
Set state	Sets the correlation state of an object in the NetView object database. The current state is updated in the corrstat1 field in the object database. The previous value in the corrstat1 field is moved to the corrstat2 field. This process continues until the current state and as many as four previous states are stored in the object database. You can view the correlation state by selecting the object and then selecting the Display Correlation Status option from the context menu.
Thresholds	Checks for repeated occurrences of the same trap or of traps with one or more attributes in common. You can use this node to forward an event after receiving the specific number of the same event received within a specific time period. Use this node with the Trap settings node to identify a specific trap number.
Trap settings	Specifies a specific trap to be processed and is identified by a pair of generic and specific trap numbers. The window displays a list of enterprise names and IDs. When you select an enterprise ID, a list of generic and specific trap numbers for that enterprise is displayed in the Event name and Specific fields. Select one or more traps from this list.

The rule sets are created using the ruleset editor. You can invoke the editor by entering the **nvrEdit** command on the command line, or by selecting **Tools → Ruleset Editor** from the NetView console. A template of available nodes and a work area for creating the rule is displayed (Figure 6-13).

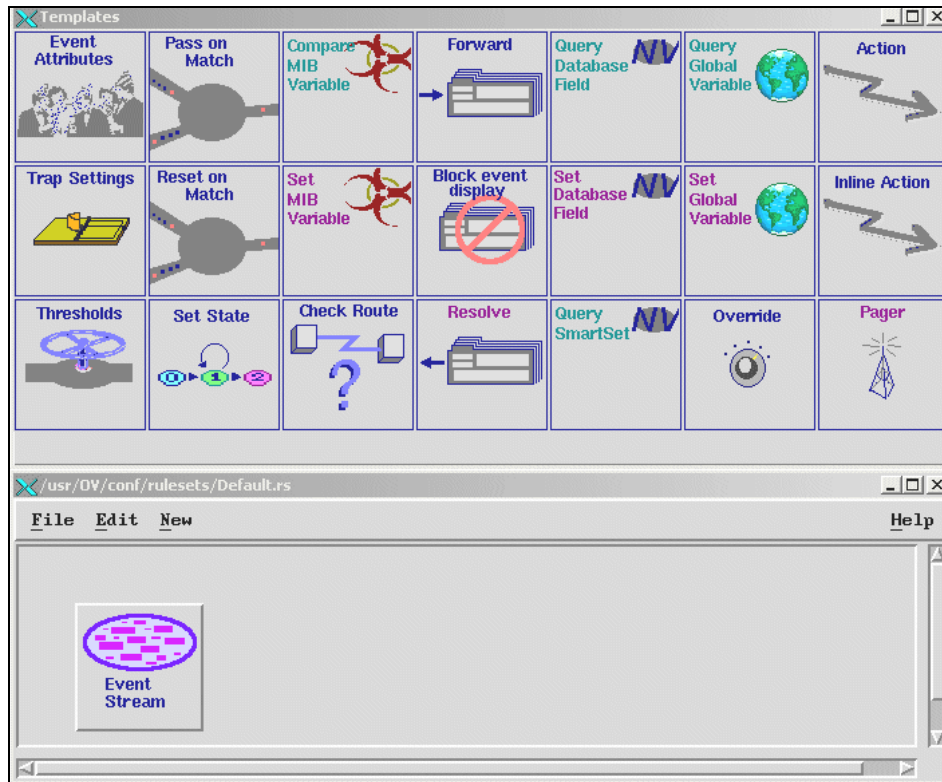


Figure 6-13 NetView ruleset editor

Drag and drop the appropriate nodes into the workarea and connect them logically using Edit → Connect Two Nodes. For more information about this process, see Chapter 5, “Correlating Events”, in *Tivoli NetView for UNIX Administrator’s Guide, Version 7.1*, SC31-8892.

After you create a rule set, you can use it in several ways:

- ▶ To limit the events displayed in a workspace (see 6.1.1, “Filtering and forwarding with NetView” on page 174)
- ▶ To perform automated actions not related to displaying the events (see 6.9.1, “Using NetView for automation” on page 338)
- ▶ To control forwarding of events to the Tivoli Enterprise Console (see 6.1.1, “Filtering and forwarding with NetView” on page 174)

All nodes are useful. In this chapter, we focus on those that are most relevant to event correlation and automation best practices. Other decision nodes are discussed only in the context of the examples provided.

## IBM Tivoli Monitoring Query command

The purpose of the **itmquery** command is to query IBM Tivoli Monitoring servers for endpoint information and display the results. Two configuration files are associated with this command line utility:

► **/usr/OV/conf/itm\_servers.conf**

This file contains IBM Tivoli Monitoring server account information. Use the **--add-server**, **--remove-server**, and **--verify-server-info** options to configure this file. The server account information in this file is used by both the **itmquery** utility and by the **servmon** IBM Tivoli Monitoring attribute discovery tests.

► **/usr/OV/conf/itm\_attributes.conf**

This file is used to configure the IBM Tivoli Monitoring Resource Model product matching used by both the **itmquery** utility and the **servmon** IBM Tivoli Monitoring attribute discovery tests.

In addition to enabling configuration of the IBM Tivoli Monitoring server account information used by the **servmon** IBM Tivoli Monitoring attribute discovery tests, this command line utility also enables obtaining useful IP addresses for **netmon** seed file creation. For example, this utility can enable you to create a **netmon** seed file containing the IP addresses of all the endpoints that are actively monitored by one or more IBM Tivoli Monitoring servers. Or, instead of endpoints, you may want to be more restrictive and create a **netmon** seed file that contains only the IBM Tivoli Monitoring monitored IBM WebSphere® endpoints for all monitored IBM Tivoli Monitoring servers.

### Usage

To further explain how to use this command, Example 6-13 displays the command's man page.

#### *Example 6-13 itmquery man page*

---

```
Usage: itmquery
[-h | --help]
[--add-server server_name [--port port_number]]
[--remove-server server_name]
[--verify-server-info]
[--dump-endpoints]
[--dump-products-for-endpoints <true_or_false>]
[--server server_name]
[--logconfig log4j_config_file]
```

Where:

-h, --help	display this help and exit
------------	----------------------------

<code>--add-server</code>	the server to add to the <code>itm_servers.conf</code> file
<code>--port</code>	when using the ' <code>--add-server</code> ' option, this option can also be passed to specify a non-default value for the <code>oserv</code> port on the server machine
<code>--remove-server</code>	the server to remove from the <code>itm_servers.conf</code> file
<code>--verify-server-info</code>	attempts to query every server configured in the <code>itm_servers.conf</code> file and informs you of whether the account information is valid
<code>--dump-endpoints</code>	dump the IP addresses of all endpoints being monitored by the ITM servers listed in the <code>itm_servers.conf</code> file
<code>--dump-products-for-endpoints</code>	if set to ' <code>true</code> ' the endpoint information listed with the ' <code>--dump-endpoints</code> ' switch will display with the recognized products for each endpoint
<code>--server</code>	when using the ' <code>--dump-endpoints</code> ' switch, you can use this option to specify the single server to use when performing the query
<code>--logconfig</code>	<code>log4j</code> configuration filename

---

### ***itmquery examples***

Here, we outline some examples of using the **itmquery** command and a description of those examples:

- ▶ `itmquery --add-server nodeA`  
Prompts for the user name and password and adds a new entry to the `/usr/OV/conf/itm_servers.conf` file.
- ▶ `itmquery --add-server nodeA --port 8999`  
Similar to the last example, but the entry to be added to the `itm_servers.conf` file contains non-default port information. In this case, we explicitly specify that port 8999 is to be used when attempting to connect to the `oserv` port on `nodeA`. The `oserv` port on `nodeA` is rebounded to use port 8999, and this is not the default value for this port.
- ▶ `itmquery --remove-server nodeA`  
Removes the `nodeA` server entry from the `itm_servers.conf` file.
- ▶ `itmquery --verify-server-info`  
Verifies that all configured entries in the `/usr/OV/conf/itm_servers.conf` file contain accurate IBM Tivoli Monitoring login information.

- ▶ `itmquery --server nodeA --dump-endpoints --dump-products-for-endpoints false`  
Lists the endpoints for IBM Tivoli Monitoring server nodeA.
- ▶ `itmquery --dump-endpoints`  
Lists the endpoints for all IBM Tivoli Monitoring servers configured in the `/usr/OV/conf/itm_servers.conf` file. For each endpoint, the recognized products are also listed. You can configure which products are recognized by changing the `/usr/OV/conf/itm_attributes.conf` file.
- ▶ `itmquery --dump-endpoints --dump-products-for-endpoints false`  
Lists the endpoints for all IBM Tivoli Monitoring servers configured in the `/usr/OV/conf/itm_servers.conf` file.

### 6.3.2 IBM Tivoli Enterprise Console correlation

IBM Tivoli Enterprise Console has many ways to perform correlation. In this section, our goal is to list those methods, along with reasons behind them.

#### State correlation engine

The newest form of correlation available within IBM Tivoli Enterprise Console 3.9 is the state correlation engine, available on IBM Tivoli Enterprise Console gateways. This feature is not enabled by default, and must be configured for use. The rules for this engine are developed via XML and not the common Prolog format used with the normal IBM Tivoli Enterprise Console event server rule bases. You can develop these rules from a central location (Tivoli Management Region (TMR) server) and distribute them to the appropriate gateways via MDIST.

For information about configuring an IBM Tivoli Enterprise Console gateway to enable state correlation, follow the steps outlined in the *IBM Tivoli Enterprise Console User's Guide, Version 3.9*, SC32-1235.

For correlation purposes, we discuss two of the six rule types available with the state correlation engine: pass-through and reset on match.

#### *Pass-through rules*

The pass-through rule forwards the trigger event only if a specific set of events arrives within a specified time interval. If the required events arrive before the timer expires (optionally is a specific sequence), the trigger event is forwarded. If they do not arrive, the timer resets and the trigger event is not forwarded.



### ***Reset on match rules***

The reset on match rule forwards the trigger event only if a specific set of events does not arrive within a specified time interval. If the required events arrive before the timer expires (optionally a specific sequence), the trigger event is not forwarded. If they don't arrive, the timer resets and the trigger event is forwarded.

A situation that can be used as an example of this predicate is for NetView event Node Down. This is not a trusted event since it can be generated by a problem with the NetView polling to the node and it can still be alive. To be more sure that IBM Tivoli Enterprise Console receives only an event if the node is really down, a state correlation rule using the predicate reset on match can be used.

**Note:** The best practice to address this situation is to create a rule directly on NetView. However, if you can't or it requires too much work, the IBM Tivoli Enterprise Console gateway is the next step.

In Example 6-14, a rule was created to identify Node Down events and wait six minutes (this time was defined because the default polling interval for NetView is five minutes) after it arrives. If no Node Up event arrives in this time, the Node Down event is sent. If it was a polling error or time out and the next polling succeeds, no event is generated.

#### *Example 6-14 Code for reset on match rule*

---

```
<rule id="nodeDown.resetOnMatchOfUp">
  <eventType>TEC_ITS_NODE_STATUS</eventType>

  <resetOnMatch timeInterval="360000" randomOrder="false">
    <cloneable ignoreMissingAttributes="false" attributeSet="hostname"/>
    <predicate>
      <![CDATA[
        &source == "nvserverd" &&
        &nodestatus == "DOWN"
      ]]>
    </predicate>
    <predicate>
      <![CDATA[
        &source == "nvserverd" &&
        &nodestatus == "UP"
      ]]>
    </predicate>
  </resetOnMatch>
</rule>
```

---

The first test made with this rule was to unplug the network cable for node PHIWAS01, in our lab environment, for more than six minutes. Figure 6-14 shows that the Node Down event was received six minutes after the Interface Down event for that node. Then, the reset on match predicate for state correlation creates a time window that matches the first criteria. In the defined time window, if no event is received in matching the second criteria, then the event is sent. The interface down event for that node was not filtered to easily present the case.

Time Received	Event	Class	Hostname	Severity	Status
October 14, 2003 6:20:31 PM EDT	Other	TEC_ITS_NODE_STATUS	phiwas01.phil...	Harmless	Closed
October 14, 2003 6:14:31 PM EDT	Effect	TEC_ITS_INTERFACE_STATUS	phiwas01.phil...	Harmless	Closed
October 14, 2003 6:14:31 PM EDT	Other	TEC_ITS_SA_STATUS	phisw01.phil...	Warning	Open
October 14, 2003 5:42:41 PM EDT	Other	WebSphere_MQ_QueueManagerUnavai...	phiwas02	Critical	Open
October 14, 2003 5:42:41 PM EDT	Other	WebSphere_MQ_QueueManagerUnavai...	phiwas02	Critical	Open
October 14, 2003 5:38:08 PM EDT	Other	WebSphere_MQ_Low_ChannelsActive	phiwas01	Warning	Open
October 14, 2003 5:36:56 PM EDT	Effect	TEC_ITS_INTERFACE_STATUS	phiwas01.phil...	Harmless	Closed
October 14, 2003 5:36:56 PM EDT	Other	TEC_ITS_NODE_STATUS	phiwas01.phil...	Harmless	Open

Figure 6-14 IBM Tivoli Enterprise Console Event Viewer Node Down

When the Node Up event arrives after the time window expires, it goes to the event server that correlates it and closes the Node Down event (Figure 6-15).

Time Received	Event	Class	Hostname	Severity	Status
October 14, 2003 6:25:58 PM EDT	Other	TEC_ITS_SA_STATUS	phisw01.phil...	Harmless	Open
October 14, 2003 6:25:44 PM EDT	Effect	TEC_ITS_INTERFACE_STATUS	phiwas01.phil...	Harmless	Closed
October 14, 2003 6:25:44 PM EDT	Other	TEC_ITS_NODE_STATUS	phiwas01.phil...	Harmless	Open
October 14, 2003 6:20:31 PM EDT	Other	TEC_ITS_SA_STATUS	phisw01.phil...	Warning	Open
October 14, 2003 6:14:31 PM EDT	Effect	TEC_ITS_INTERFACE_STATUS	phiwas01.phil...	Harmless	Closed
October 14, 2003 6:14:31 PM EDT	Other	TEC_ITS_SA_STATUS	phisw01.phil...	Warning	Open
October 14, 2003 5:42:41 PM EDT	Other	WebSphere_MQ_QueueManagerUnavai...	phiwas02	Critical	Open
October 14, 2003 5:42:41 PM EDT	Other	WebSphere_MQ_QueueManagerUnavai...	phiwas02	Critical	Open

Figure 6-15 IBM Tivoli Enterprise Console Event Viewer Node Up

If the Node Up event arrives before the time window expires, neither this event nor the Node Down event are sent to IBM Tivoli Enterprise Console. In an other test, the network cable was unplugged from node PHIWAS02. As the Node Up event arrives in the IBM Tivoli Enterprise Console gateway, a time window is created by state correlation. The node is turned on again (simulating a polling problem). Before the time window expires, a Node Up arrives. The time window is closed and no events are sent to the IBM Tivoli Enterprise Console event server.

Figure 6-16 illustrates this test. The Interface Down event at 6:30 shows that the node PHIWAS02 was down at that moment. As the Interface Up event arrives before the six-minute time window, no Node Down or Node Up event is created.

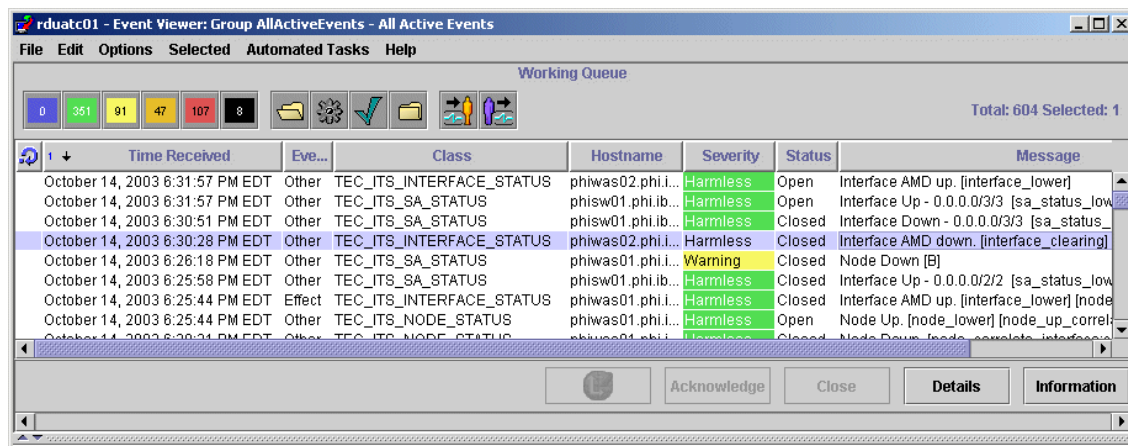


Figure 6-16 IBM Tivoli Enterprise Console Event View Interface Down

## Rule bases

The most popular and well-known form for correlation within IBM Tivoli Enterprise Console is via the IBM Tivoli Enterprise Console event server's rule base. Let's first start by discussing rulebase design approaches.

### Rule writing best practices

This is not an exhaustive list of best practices, but a key set to consider when you must create prolog rules for IBM Tivoli Enterprise Console event servers.

- ▶ Use the `commit_action`, `commit_rule`, and `commit_set` rule language predicates frequently.
- ▶ Set severity using `bo_set_slot_val`, not by calling a task.
- ▶ Use `create_event_sequence` to define event sequences.
- ▶ Send events of the same class to clear problem events. Use slot values to indicate up or down. Be consistent with NetView's out-of-box approach. This

can be done easily using format files. All of these steps from the event source assist greatly in correlating events within a rule.

- ▶ Write rules to drop unnecessary events immediately (as one of the first rules encountered so that events do not have to process all rules in the rule base).
- ▶ Monitor the number of rule timers used. You can cause yourself many problems when this is not managed appropriately.
- ▶ When writing rules, limit the use of the following predicates:
  - all\_instances
  - all\_duplicates
  - generate\_event predicates

These cause expensive processing when overused and cause damaging effects to the IBM Tivoli Enterprise Console event server.

- ▶ Use INT32 for fast comparisons.
- ▶ Avoid executing scripts from within a rule whenever possible. This is also very expensive processing and should be limited.

### ***Design approaches for rule bases***

There are several ways to structure an IBM Tivoli Enterprise Console rule base. Here are three of the most common ways, along with their pros and cons, and suggestions for when to use them:

- ▶ Use rule sets for specific event sources.

One method of structuring a rule base is to use separate rule sets that contain rules that are applicable to events from a specific source or monitoring tool. Using this approach, all the rules required to handle products, such as IBM Tivoli Monitoring and NetView, are grouped together in their own rule sets.

This implementation method is relatively easy to deploy for products that supply IBM Tivoli Enterprise Console rule sets, such as NetView. It is also convenient to use when consolidating the output of tools that may generate rules. An example of this is the IBM Event Management and Monitoring Design (EMMD) proprietary tool. This generates rules for events from event sources. All the rules that apply to an event source can be grouped together in a single rule set.

Another advantage of this approach is maintainability. Knowing that all the rules that apply to an event source reside in a single rule set means that the administrator knows exactly where to look to modify rules or add new ones.

However, the rule base can become cumbersome. Grouping all the actions associated with the events from a single source usually means that there are more rules. For example, the trouble ticketing rules for each event source may reside in the rule set for that source. Similarly, event correlation, escalation, and notification rules may exist in the rule sets for each source. This means

that there are multiple places within the rule base where the same function is performed.

- Use rule sets for business applications.

Grouping rules together based on business function may be used in environments that implement monitoring of one application at a time. To handle the event processing required for events from each application, new rule sets are created and added to the rule base.

This has some of the same advantages and disadvantages as grouping by product. Administrators know where to find rules for an application and can easily implement rules for new applications. The rule base can grow potentially larger than in the product approach. This is because the same events may be required to monitor different applications. Not only are such functions as trouble ticketing duplicated, but correlation sequences for events may be as well.

- Use rule sets for event processing functions.

When you structure the rule base in a modular fashion by function, you can implement separate rule sets for functions. Such functions include event sequence creation, event correlation and escalation, synchronization, trouble ticketing, and notification.

Performance is perhaps the best reason to implement this type of rule base. There are most likely the fewest rules with this approach. Not only are functions performed one time, but the code required to perform correlation and escalation can handle events from multiple sources as well.

When implemented properly, this type of rule base is easy to maintain. When integrating new event sources, ensure that they follow the conventions that are already coded. This allows the existing rules to handle the new events with little or no changes.

The drawback of structured rule bases is that it requires more planning to implement and more skill to maintain. The administrator must now know what each rule does and how to format new events to fit the rules.

### ***Lab approach***

In our case study, we chose a hybrid structure for the rule base. The phrase *best practices* dictates to use the modular rule base for efficiency. However, it also suggests to use as much out-of-the-box functionality as possible to maximize the amount of support provided by vendors whose rule sets are implemented. And it recommends that you minimize the effect of product upgrades and changes on the rule base.

Separate rule sets were implemented for functions such as trouble ticketing and notification. Correlation sequences were used as supplied in rule sets for the various products tested.

### ***Rule set sequencing and dependencies***

Within the TEC\_RULES subdirectory of a rulebase directory, there is a file called *rule\_sets* that indicates which rule sets are active and in which order they should be processed. The sequencing of rule sets is important. Suppose a rule set drops an event and then commits the rule base. Rules that follow are never executed. If another rule set correlates the event with a primary event, and adds information to the primary based upon the fields in the event, that rule set needs to precede the rule set that drops the event.

Also, some rule sets depend upon supporting functions supplied by other rule sets. A trouble ticketing rule set may determine the severity of the ticket it is opening based upon event severity. Therefore, any rules that escalate event severity need to be placed before the trouble ticketing rule.

When IBM Tivoli Enterprise Console is initially installed, the supplied rule sets are implemented in the proper order. If an organization decides to change the ruleset order, keep in mind the following guidelines:

- ▶ The event filtering rule set (*event\_filtering.rls*) should be near the beginning of the sequence, preferably first. This avoids unnecessary processing of events that are to be filtered out by this rule set.
- ▶ The maintenance mode rule set (*maintenance\_mode.rls*) should be near the beginning of the sequence, preferably immediately after *event\_filtering.rls*. This avoids unnecessary processing of events sent from systems in maintenance mode.
- ▶ The correlation rule set (*correlation.rls*) should be near the end of the sequence. This ensures that any event-specific correlation rules run before the general-purpose correlation rules.
- ▶ The escalation rule set (*escalate.rls*) should be near the end of the sequence and should come after *correlation.rls*. This ensures that event severities can be appropriately adjusted after other processing takes place.
- ▶ The e-business rule set (*ebusiness.rls*) should be near the end of the sequence and should come after the NetView rule set (*netview.rls*) and any other rule sets that process events from e-business services monitored by IBM Tivoli Monitoring products.
- ▶ The notification rule set (*notify.rls*) should be near the end of the sequence, preferably last. This ensures that notification is based on the most current event information.

- ▶ The escalation rule set (`escalate.rls`) depends upon the notification rule set (`notify.rls`) to provide e-mail notification of escalated events. If you want to use this function, both rule sets must be activated.
- ▶ The e-business rule set (`ebusiness.rls`) depends upon the dependency rule set (`dependency.rls`), which supports definition of dependency relationships. If you want to use the e-business rules, both rule sets must be activated.
- ▶ The e-business rule set (`ebusiness.rls`) generates missed heartbeat events in response to certain network events. If you want to handle these events properly, the heartbeat rule set (`heartbeat.rls`) must also be activated.
- ▶ The NetView rule set (`netview.rls`) correlates heartbeat events with other network events. If you want this correlation to take place, the heartbeat rule set (`heartbeat.rls`) must also be activated.

See Chapter 7, “A case study” on page 357, for information about the rule base implemented in our testing.

## Out-of-the-box rules

For more information about how the out-of-the-box rule sets work, see *IBM Tivoli Enterprise Console Rule Set Reference, Version 3.9*, SC32-1282.

### ***cleanup.rls***

This rule set is responsible for purging old events from the event cache. In general, events that remain open for a long time without being addressed should be closed, on the assumption that they are resolved or are otherwise inactive.

The rule set is based on three parameters that can be adjusted to meet specific company needs. The `_default_span` parameter defines the time-out value that controls how old open events must be before they are closed automatically. The default value is 48 hours. The `_cleanup_interval` parameter defines the frequency that old events are cleaned. The default is 30 minutes. The `_cleanup_list` parameter defines the list of severities to be cleaned. The default is HARMLESS and UNKNOWN. A fourth parameter, `_cleanup_admin`, is used to set the administrator to automatically close the events.

The default value for the `_default_span` parameter can be too big since you may not want to have all these events in your rule cache, during all this time. Events with severity of HARMLESS or UNKNOWN are commonly used for correlation in the time that they arrive and next to it. If they stay a long time in the cache, it generates unnecessary over processing. We recommend that you change this parameter to a smaller value if it does not affect your correlations.

The rule can be used to clean more severe events. However, in this case, a good practice is to clone the rule and create one for each event.

### ***correlation.rls***

You can customize the correlation rule set by modifying statements in the `correlation_parameters` action of the `correlation_configure` rule. The *latency option* is configurable.

You can specify the time range, or latency, to be used when searching the event cache for events to correlate. By default, searches are limited to 10 minutes (600 seconds) backward in the event cache. To change the latency, modify the statement that sets the `_correlation_time_window` parameter:

```
_correlation_time_window =seconds
```

*Seconds* is the number of seconds that represents how far backward you want to search the cache for events.

### ***dependency.rls***

The dependency rule set contains rules that support the definition of dependency relationships used by the e-business rule set (`ebusiness.rls`). Before you use the e-business rule set, you must define the dependency relationships that apply to the e-business services and network resources in your environment.

To define these relationships, create a text file that contains a series of dependency statements, each of which is converted into a dependency fact in the knowledge base. Each dependency statement is on a separate line and consists of three parts, separated by white space:

```
host_a dependency_type host_b
```

*host\_a* is the fully qualified name of the host that has a dependency on another host. *dependency\_type* is the nature of the dependency. *host\_b* is the fully qualified name of the host upon which *host\_a* depends. The following example shows three dependency statements:

```
phiwas01 WMQ_DEPENDS_ON_WMQ phiwas02  
phiwas02 WMQ_DEPENDS_ON_WMQ phiwas01
```

These statements define the following dependency relationships:

- ▶ The first statement indicates that WebSphere MQ services on `phiwas01` depend upon WebSphere MQ services on `phiwas02`.
- ▶ The second statement indicates that WebSphere MQ services on `phiwas02` depend upon WebSphere MQ services on `phiwas01`.

To see if the dependency is really running, and which dependencies are on the knowledge base, open the `$DBDIR/dependencies.pro` file. The contents should resemble those in Example 6-15.



#### Example 6-15 The *dependencies.pro* file contents

```
# cat $DBDIR/dependencies.pro
dependency(WMQ_DEPENDS_ON_WMQ,phiwas01,phiwas02) .
dependency(WMQ_DEPENDS_ON_WMQ,phiwas02,phiwas01) .
#
```

**Note:** Each dependency fact represents a single, unidirectional dependency relationship. Therefore, if two interconnected hosts have mutual dependencies on one another, you must define a separate dependency fact for each direction of the relationship.

After you finish defining dependency relationships in the text file, use the **wrb -imptdp** command to load these relationships into the knowledge base as dependency facts:

```
wrb -imptdp filename
```

*filename* is the name of the text file that contains the dependency statements.

To remove dependency relationships, use the **wrb -deldp** command:

```
wrb -deldp filename
```

#### ***ebusiness.rls***

This rule does not use events from the IBM Tivoli Enterprise Console MQ adapter. It only uses events from IBM Tivoli Monitoring profiles created by the following three IBM Tivoli Monitoring applications:

- ▶ IBM Tivoli Monitoring for Business Integration: WebSphere MQ
- ▶ IBM Tivoli Monitoring for Web Infrastructure: WebSphere Application Server
- ▶ IBM Tivoli Monitoring for Databases: DB2®

To use the rule, set the *fqhostname* slot value. To set this slot, you need IBM Tivoli Monitoring Fix Pack 5.

During our lab testing, our lab environment was not configured with Fix Pack 5, as this fix pack was released shortly after our lab work was completed. To work around this problem, we create a rule named *set\_fqhostname*. This rule filled the slot *fqhostname* with information from the *hostname* slot, as shown in Example 6-16.

#### Example 6-16 *set\_fqhostname rule*

---

```
rule: set_fqhostname:
(
    event: _event of_class within ['WebSphere_MQ_ChannelThroughputProblem',
                                    'WebSphere_MQ_QueueManagerUnavailable',
                                    'WebSphere_MQ_ChannelNotTransmitting',
                                    'WebSphere_MQ_ChannelStartupError',
                                    'WebSphere_MQ_QueueFilling',
                                    'WebSphere_MQ_ChannelNotActivated',
                                    'WebSphereAS_high_DBPool_faults',
                                    'WebSphereAS_high_DBPool_avgWaitTime',
                                    'WebSphereAS_high_Transaction_avg_response_time',
                                    'WebSphereAS_high_Transaction_timeout_ratio',
                                    'WebSphereAS_high_DBPool_percentUsed',
                                    'DB2_Down_Status',
                                    'DB2_High_ConnectionErrors',
                                    'DB2_High_ConnWaitingForHost',
                                    'DB2_High_MostRecentConnectResponse',
                                    'DB2_High_DB2ApplicationAgent_TotUserCpuTime',
                                    'DB2_High_ApplicationAgent_TotSystemCpuTime',
                                    'DB2_High_PctConnectionsUsed',
                                    'DB2_High_CurrentConnections',
                                    'DB2_High_HostTimePerStatement',
                                    'DB2_High_NetworkTimePerStatement',
                                    'DB2_High_TimePerStatement',
                                    'DB2_High_InstanceAgents_PctAgentsWaiting',
                                    'DB2_High_ApplicationAgents_Workload',
                                    'DB2_High_InstanceAgents_AgentCreationRatio']

    where[
        hostname: _hostname
    ],
    reception_action: set_fqhost_host:
    (
        bo_set_slotval(_event, fqhostname, _hostname),
        re_mark_as_modified(_event, _),
        commit_action
    )
).
```

---

To show how IBM Tivoli Enterprise Console e-business out-of-the-box rules work, two queue managers were created on two different servers, and a channel between them was established. The queues in one manager are accessed by the other, so, a dependency is created. This dependency is reproduced for the e-business rules through the dependency rules.

When IBM Tivoli Enterprise Console receives an MQ event, e-business correlation looks for a matching cause event from the dependent queue manager. If it's found, it correlates the event with its cause.

Figure 6-17 shows the reception of a Queue Manager Unavailable event from PHIWAS02, followed by a Channel Not Transmitting event from PHIWAS01. Since both servers are dependent and the Channel Not Transmitting event was originated because the queue manager on the other server was not available, the rule uses the link\_effect\_to\_cause predicate and correlates the effect with its cause.

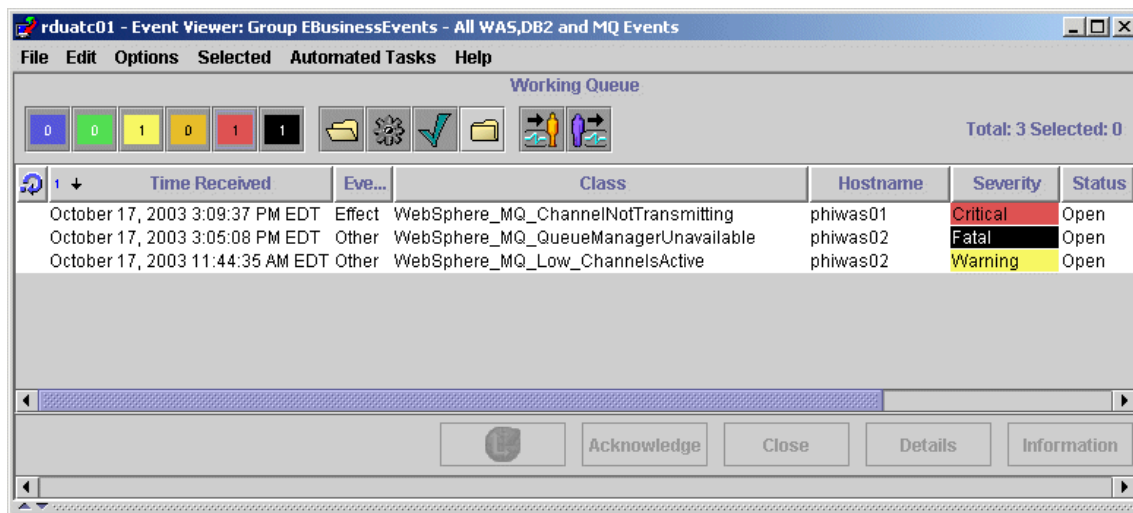


Figure 6-17 IBM Tivoli Enterprise Console Event Viewer with WebSphere MQ Event

The link\_effect\_to\_cause predicate updates the cause\_date\_reception and cause\_event\_handle attributes of the effect event so that these attributes contain a reference to the cause event. The value of the date\_reception attribute of the cause event is placed in the cause\_date\_reception attribute. The value of the event\_handle attribute of the cause event is placed in the cause\_event\_handle attribute.

In Figure 6-18, you can see the WebSphere\_MQ\_ChannelNotTransmitting event pointing to the Websphere\_MQ\_QueueManagerUnavailable event as its cause. The Causing event ID and Causing event received from the cause event are in the effect event. You can compare it with the time in Figure 6-17.

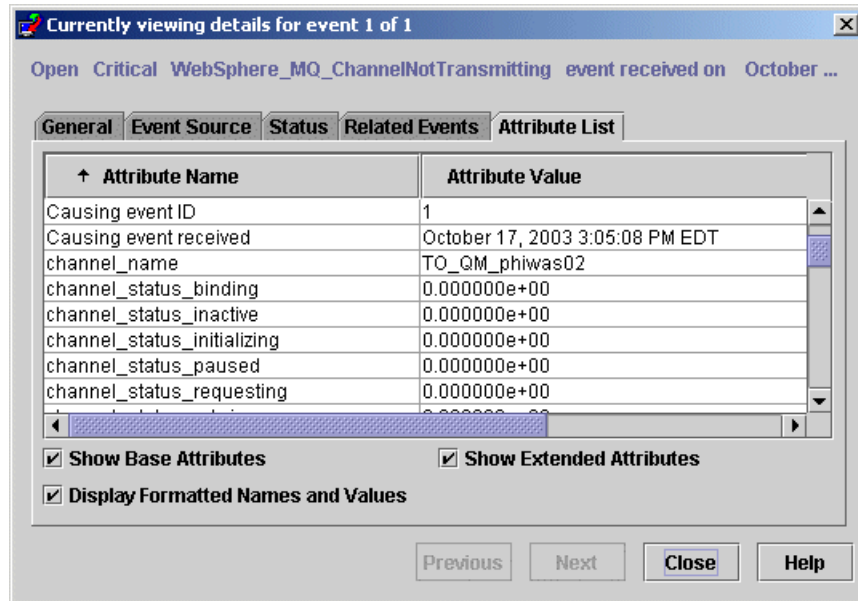


Figure 6-18 Event details for WebSphere MQ causing event

### 6.3.3 IBM Tivoli Monitoring correlation

IBM Tivoli Monitoring performs correlation within resource models. Many of the Windows resource models include automatic correlation by default. If you want to perform more extensive correlation, you must edit the definition of a resource model or create a new resource model via the Resource Model Builder.

It is never a good idea to modify the definition of an existing resource model. Instead, copy the definition of a source resource model into a new resource model and perform your customizations there. This prevents corruption of existing resource models and gives you the ability to back out of a possible mistake without massive ramifications.

## 6.4 Notification

Notification is the means by which appropriate personnel are informed about conditions requiring action. While best practices dictate using the trouble-ticketing system to notify, sometimes this is not practical. For example, there may not be a trouble-ticketing system within the event management hierarchy. Or the organization may be decentralized and use different systems management tools for different IT resources.

In these cases, you may want to use a means other than the trouble-ticketing system for notification. The following sections outline the capabilities of IBM Tivoli products to provide notification.

### **6.4.1 NetView**

When used as a primary rather than intermediary management tool, NetView needs to notify for conditions that require action. NetView can perform most of the notification methods discussed earlier. This section summarizes NetView's capabilities in reporting problems.

#### **Consoles**

The NetView consoles are one means of informing someone of conditions that require intervention. Consoles typically contain two major sections that perform this function: the topology view and the event summary.

Several different types of consoles are available with NetView. While the capabilities and look of each console type differs, the status and event information portrayed in them is similar.

Since the focus of this section is event notification, only those features of the topology and event views used for this purpose are covered. NetView's diagnostic capabilities are outside the scope of this discussion.

#### ***Topology view***

The topology view shows the hierarchy and status of the resources managed by NetView. Objects that represent resources, such as routers, switches, hubs, and servers, are color-coded to delineate their status. Red typically indicates that the resource is down. Yellow means that some interfaces on the resource are down. Green indicates fully operational.

These colors usually indicate the layer 3 connectivity status of the device. However, they can also indicate layer 2 problems and user-defined conditions.

#### ***Event view***

While useful for indicating something is wrong, topology views do not show details about the specific condition requiring action. The purpose of the event display is to provide this additional information.

#### ***Pop-up windows***

NetView has the capability to open warning windows upon receipt of a trap. This function is typically used in environments where operators need to watch several consoles or to perform activities in addition to console monitoring. The flashing pop-up window with its optional sound indicator draws the operator's attention to

the console on which a critical event is displayed. Second and third shift operators in particular find this function useful.

NetView for UNIX provides two executables to open pop-up windows: **ovxbeep** and **ovxecho**. To display a pop-up window, specify a shell script when a specific event occurs that executes the **ovxbeep** or **ovxecho** command when a specific event occurs. If you execute the **ovxbeep** command in the shell script, an error message window is displayed *with* an audible alarm. If you execute the **ovxecho** command in the shell script, an error message window is displayed *without* an audible alarm. The shell script must export the display to the appropriate workstations before executing the **ovxecho** or **ovxbeep** commands. Also the **xhost** command must have been run on the workstations where the pop-up window is to be displayed.

You specify the name of the shell script in the Optional Command and Argument format section of the Event Configuration window. For example, you want to display a pop-up window when NodeA or NodeB fail. For NodeA, you want to include an alarm. You also want to send an electronic-mail notice of the failure. Here are the steps:

1. In the Options pull-down menu, click **Event Configuration** → **Trap Customization: SNMP**.
2. In the Event Configuration window, select or enter:
  - Enterprise name: netview6000
  - Event: Specific 58916865
3. Click **Modify**.
4. The Modify Event window opens. Enter the following in the Command for Automatic Action field:  
`< ShellScriptPath > $2`
5. Click **OK** to close the Modify Event window.
6. Click **OK** to close the Event Configuration window.

**Note:** If you filter an event for which you configured a command for automatic action, the actions specified in the shell script are still executed. If the shell script executes the **ovxbeep** or **ovxecho** command, for example, an error message window is displayed even though the event was filtered.

In Example 6-17, the \$2 parameter passes the name of the device that generated the alert to the script. The shell script checks the \$2 flag to see whether it is NodeA or NodeB that generated the alert. If it is NodeA, the shell script calls a program, /usr/OV/bin/ovxbeep, that displays a window and an audible alarm. If it is NodeB, the shell script calls the program,

/usr/OV/bin/ovxecho, that displays a window without a sound. For either node, an electronic-mail notice is sent to the addresses specified in the shell script.

---

*Example 6-17 Shell script for node down trap*

---

```
#!/bin/ksh
# example.sh
Shell script for node down trap from the netview6000 enterprise
# (specific = 58916865). Displays warning messages and sends e-mail.
export DISPLAY=NodeA.austin.tivoli.com:0
export DISPLAY=NodeB.austin.tivoli.com:0
if [ $1 = NodeA.austin.tivoli.com ]; then
/usr/OV/bin/ovxbeep $1" is down"

echo $1" is down" | mail oper1@manager.austin.tivoli.com
fi
if [ $1 = NodeB.austin.tivoli.com ];then
/usr/OV/bin/ovxecho $1" is down"
echo $1" is down" | mail oper2@manager.austin.tivoli.com
fi
```

---

See the /usr/OV/prg\_samples/nnm\_examples/beeper/beep\_951x sample shell script for more examples.

### ***NetView console best practice***

In general, use the NetView console for diagnostics, and not for watching events, in regard to notification.

## **Rules**

This section describes some of the NetView rules used for notification.

### ***Paging***

A pager is responsible for issuing a call to a pager that has been defined in a NetView user profile. You should have already configured the paging utility. The paging utility uses the pager number and carrier information defined in the user profile. The window contains the following relevant fields:

- ▶ **User ID:** Specifies the NetView user ID of the person to be paged. If pager information is not found in the NetView user profile or there is no NetView ID for the user, a window opens in which you can enter the user ID and pager information. Then a user profile is created or updated.
- ▶ **Message Text:** Specifies message text to be delivered with the page. The message can include trap data passed in environment variables.

To use the NetView paging utility through an event correlation rule (using the Pager node) or from the command line (using the **nvpage** command) with an analog line for modem communications, perform the following steps for the modem that is attached to your system:

1. Add and configure a TTY device for modem communications using the UNIX **mkdev** command.
2. Test the modem communication through the TTY device using a communications program, such as **ate**, provided with your operating system.
3. Customize the paging utility configuration files, which are located in the `/usr/OV/conf` directory:
  - **nv.carriers**: Lists the defined carriers. Add the appropriate entries for all paging carriers used at your site. The numeric IDs are accepted on the Modem line. The Y/N field indicates the pager type. If numeric IDs are accepted, the pager type is numeric. If numeric IDs are not accepted, the pager type is alpha.

See the `nv.carriers` man page for more information.

- **\*.modem**: Contains the default information for the modem. The asterisk (\*) represents the name of the modem file. The following modem files are provided:
  - *ibm5853.modem*: For the 2400 baud IBM Model 5853 modem
  - *ibm7855.modem*: For the IBM Model 7855
  - *newhayes.modem*: For most Hayes compatible modems
  - *oldhayes.modem*: For Hayes compatible modems that do not understand the extended AT command set
  - *qblazer.modem*: For Hayes compatible modems
  - *blank.modem*: For you to copy and customize

Usually, you do not need to change the values in the modem file. If a modem file is not provided for the modem you are using, use the `blank.modem` file as a template.

See the `modem` man page for more information.

- **nvpager.config**: Lists the defaults that are the physical characteristics of the modem.

Specify the TTY device that you already configured and tested. Change the modem characteristics to reflect the values configured on the TTY device. Add the name of the modem file that corresponds to the modem dedicated to paging. See the `nvpager.config` man page for more information.



- **nvpaging.protocols**: Defines the characteristics of the paging protocols: TAP, IXO, PET, and PAKNET. The Protocol field in the nv.carriers file specifies the paging protocol being used and points to the nvpaging.protocols file for configuration information. If you are using a paging protocol similar to TAP, IXO, PET, and PAKNET, copy the information provided for one of these protocols and modify it with the appropriate information for the protocol you are using.

See the nvpaging.protocols man page for more information.

After you update the configuration files, stop and restart the nvpagerd daemon to make the changes available to the paging utility.

4. Create NetView security user profiles for those individuals who you want to page automatically through an event correlation rule set.

When you use the Pager node in an event correlation rule set, you specify the user ID of the person you want to page. The NetView security user profile defines the user ID and the paging information. It is not necessary to activate security to access the paging information in a user's profile.

You can also send a page from the command line using the **nvpage** command. See the man page for more information.

## 6.4.2 IBM Tivoli Enterprise Console

IBM Tivoli Enterprise Console has various methods of notification. This section describes those methods.

### Consoles

The IBM Tivoli Enterprise Console Version 3.9 now has two ways to view events on a console:

- Java-based console
- Web-based console

The Java-based console is usually installed locally on an administrator's desktop. In IBM Tivoli Enterprise Console Version 3.9, it is mainly used for the console configuration. Tivoli administrators responsible for that configuration should have access to this console. Figure 6-19 shows an example of the configuration window of the Java-based console.

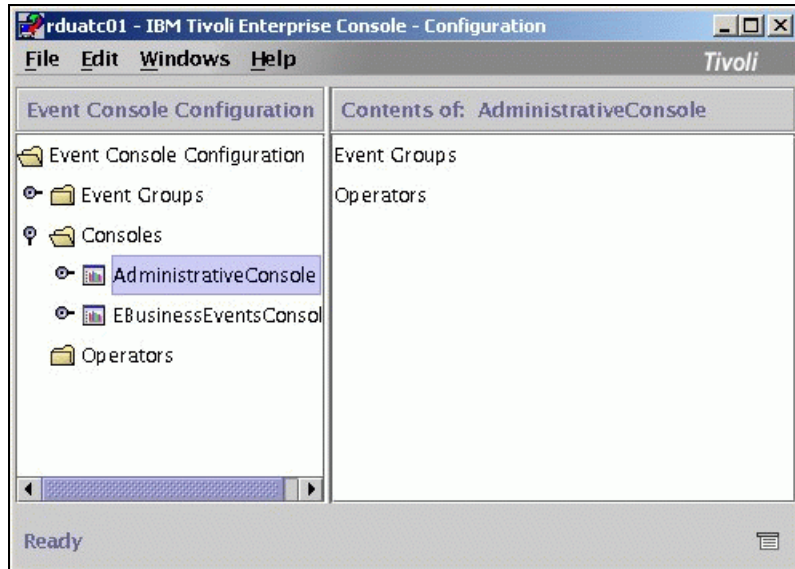


Figure 6-19 Example of an administrator's configuration window

Also with this console, you have the functionality to view, close, acknowledge, and run automated tasks for events. Operators can use this version of the console to view all the events for which they are responsible. Figure 6-20 shows an example of the event viewing section of the Java-based console. The main disadvantage is that you have to install it on a workstation before you can view it.

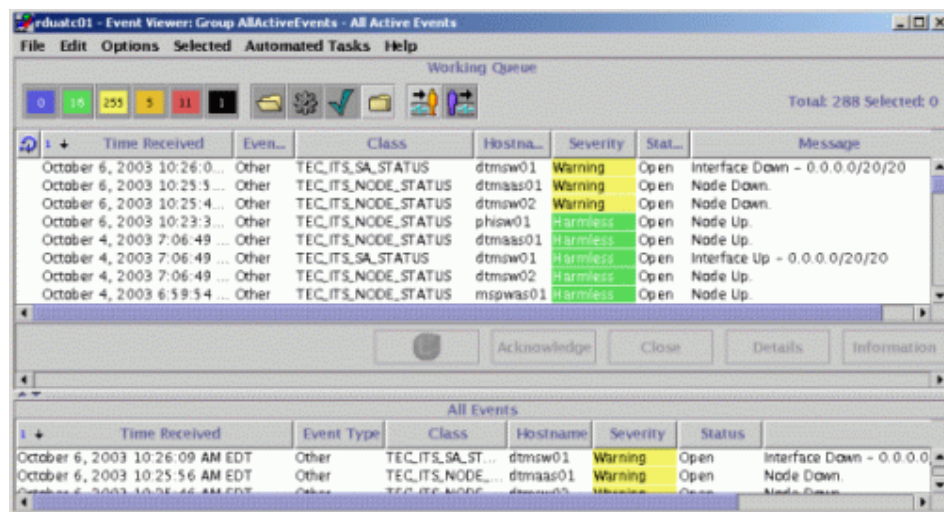


Figure 6-20 Example of the Event Viewer window on the Java console

The other option now available with IBM Tivoli Enterprise Console 3.9 is the Web-based console. This console runs on a WebSphere Application Server that can be installed via the installation wizard for IBM Tivoli Enterprise Console 3.9. You can then log in to that WebSphere Application Server via a Web page that directs you to the IBM Tivoli Management Region where IBM Tivoli Enterprise Console is located.

This console is only used for viewing events. No configuration can be done from the Web console. However, operators can acknowledge, close, and run automated tasks from the events. The Web console has its advantages in that no software needs to be installed on a workstation before you view it. Figure 6-21 shows an example of the new IBM Tivoli Enterprise Console Web-based console.

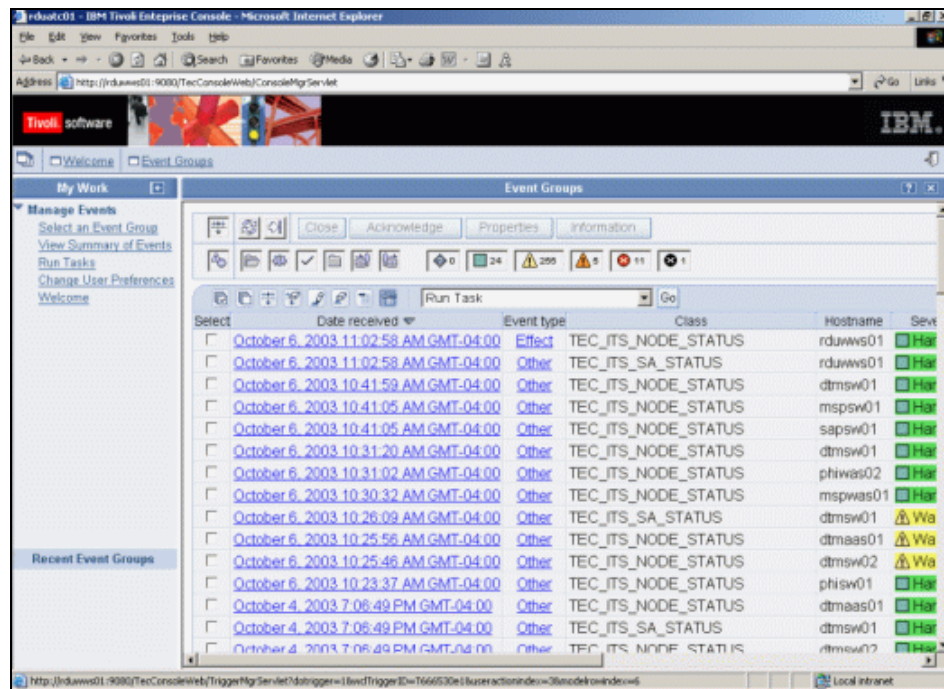


Figure 6-21 Example of Web console event viewer

### 6.4.3 Rules

You can also use rules for notification within IBM Tivoli Enterprise Console. The most popular are described here.

## E-mailing and paging

E-mailing and paging can be handled directly from the IBM Tivoli Enterprise Console event servers rule base. There is an out-of-the box rule set supplied for these actions called `notify.rls`. This rule notifies, as defined in the rule, on the severity of the event and the event class. You must define within this rule the method in which you want to notify (page or e-mail) the person or group and their addresses.

**Note:** You must activate and modify the out-of-the box rules supplied with IBM Tivoli Enterprise Console 3.9 according to your environment so they can work properly.

### *Customizing `notify.rls` for which events to send notifications*

In the `notify.rls` rule, find the section shown in Example 6-18 located within the `notify_configure` rule.

*Example 6-18 `notify_configure` within the `notify.rls` rule set*

---

```
rerecord(notify_list, 'EVENT',
          ['MAIL',                                     % type of notification
- MAIL/PAGE
          'Administrator',                             % user to notify
          'Administrator@EventServer']], % user email/page
address

          % Set class-specific notification information
          rerecord(notify_list, 'TEC_Error',
          ['MAIL',                                     % type of notification
- MAIL/PAGE
          'Administrator',                             % user to notify
          'Administrator@EventServer']] % user email/page
address
      ),
```

---

This section is where you set the type of notification to send the user to send it to and their address. The top section starting with `rerecord(notify_list, 'EVENT',` sets the notification method for all events which are separated later based on severity in the following rules. For example, if we want to send all events by e-mail to administrator Bob at `bob@company.com`, we can modify this section as shown in Example 6-19.

*Example 6-19 Sending an e-mail within the notify.rls rule set*

---

```
rerecord(notify_list, 'EVENT',  
          ['MAIL',                                     % type of notification  
- MAIL/PAGE  
          'Bob',                                     % user to notify  
          'bob@company.com']], % user email/page address
```

---

The bottom section is where you can configure notifications based on the IBM Tivoli Enterprise Console class. In Example 6-18, the rule is configured for the TEC\_Error class to send the administrator an e-mail. If you want to send notifications on different classes, you add them here. For example, if we want to send an e-mail to Bob when an event comes in with the class TEC\_Start, we add an entry to this rule as shown in Example 6-20.

*Example 6-20 Sending TEC\_Start events via e-mail via the notify.rls rule set*

---

```
rerecord(notify_list, 'TEC_Error',  
          ['MAIL',                                     % type of notification  
- MAIL/PAGE  
          'Administrator',                             % user to notify  
          'Administrator@EventServer']] % user email/page  
address  
),  
rerecord(notify_list, 'TEC_Start',  
          ['MAIL',                                     % type of notification  
- MAIL/PAGE  
          'Bob',                                     % user to notify  
          'bob@company.com']] % user email/page address  
),
```

---

The rest of the rule deals with sending the notifications based on each type of severity. The rules send a notification on any event if the event severity is changed to *Fatal* or *Escalated*. The rules also send a notification if an event is re-opened. Table 6-6 lists the types of notification sent on each severity if the default parameters of the rule remain unchanged.

*Table 6-6 Notification types and their severities*

Severity	Notification	Escalation
Fatal	Notify via e-mail	Notify via e-mail
Critical	Notify via e-mail	Notify via e-mail
Minor	Notify via e-mail	Notify via e-mail

Severity	Notification	Escalation
Warning	No notification	No notification
Harmless	No notification	No notification
Unknown	No notification	No notification

It is a good idea to notify only on events that are problems or require actions. You may want to modify the remaining rules in this set to only notify for those events. If you want to send pages through this rule, follow these steps:

1. Create a script called `TEC_Notify.sh` to carry out your paging based on the paging software that you are using.
2. Place the script in the `$BINDIR/TME/TEC/scripts` directory.
3. Uncomment the following line in the `TEC_TEMPLATES/notify.pro` file:  

```
%exec_program(_event,'scripts/TEC_Notify.sh',_fmt_str,[],'YES')
```
4. Add that template to the `Tec_Templates`.
5. After you are done modifying these rules, compile, load and restart the event server.

This notification rule must be placed at the every end of the order in your rule base. This is so that an event goes through all of the correlation before making it to this rule to be notified. Therefore, you know that you are notifying for a valid problem.

### ***Scripts***

Another way to notify from the rule base is to run custom scripts from events meeting certain criteria from the rule base using the `exec_program` predicate from the `notify.rls` rules. Doing this is completely custom. However, modifying the scripts to keep the notification information is a little easier because you do not have to compile and restart the event server every time you have to make a change. Keep in mind that executing scripts from a rule is a highly taxing operation to the system processor and should be moderated.

Before you make changes to the following rule, comment out the `notify_configure` rule. This information is no longer necessary since you are keeping your addressing information within the scripts. Next you modify the `notify_for_fatal_events` rule to run a script. Example 6-21 shows the original format of the rule.

*Example 6-21 Original notify\_for\_fatal\_events rule*

---

```
rule:
notify_for_fatal_events:
(
    event: _event of_class _class
        where [
            severity: _severity equals 'FATAL',
                status: equals 'OPEN',
                msg: _msg
        ],
    reception_action:
    (
        first_duplicate(
            _event, event: _dup_ev
            where [ status: _status outside [ 'CLOSED' ] ] ),
        commit_rule
    ),
    reception_action:
    (
        (recorded(notify_list, _class, [_type,_user_name,_user_addr]) ->
            assert_notify(_event,_class),
            true
        );
        assert_notify(_event,'EVENT')
    ),
    commit_rule
)
).
```

---

Example 6-22 shows the modifications that you need for this rule to use our script to perform notification.

*Example 6-22 Modified notify\_for\_fatal\_events rule for using script notification*

---

```
rule:
notify_for_fatal_events:
(
    event: _event of_class _class
        where [
            severity: _severity equals 'FATAL',
                status: equals 'OPEN',
                msg: _msg
        ],
    reception_action:
    (
        first_duplicate(
```

```

        _event, event: _dup_ev
        where [ status: _status outside [ 'CLOSED' ] ] ),
        commit_rule
    ),
    reception_action:
    (
        exec_program(_event, 'scripts/notify_bob.sh', '', [], 'NO'),
        commit_action
    )
).

```

---

This rule now runs the `notify_bob` script whenever a fatal event comes in. We changed the reception action to run our script instead of using the template to send an e-mail. Example 6-23 shows the contents of the `notify_bob.sh` script that you may use to send an e-mail to Bob.

---

#### *Example 6-23 notify\_bob.sh script*

---

```

#!/usr/bin/ksh
NOW=`date`
echo $NOW, $hostname, $class, $msg, $date >>/tmp/notify_bob.out

PATH=/bin:/usr/bin:/usr/ucb:/usr/lib
export PATH

sendmail -F TEC -t << __EOF__
To: bob@company.com
Subject: Notification from TEC
$msg happened on $hostname, please fix this Bob or it's your job
__EOF__

```

---

In this script, we send an e-mail to Bob with the subject *Notification from TEC*. We can now log the notification information in the `/tmp/notify_bob.out` log file. We can also use variables from the IBM Tivoli Enterprise Console event to populate our e-mail, such as `$msg` and `$hostname`. It is now a little easier to add and remove recipients to this e-mail. We can even customize the message, without restarting the IBM Tivoli Enterprise Console event server.

## Options

Another way to notify from the rule base is to use *fact files*. These fact files keep large amounts of information that the rule you are using consults. For notification purposes, we can use a fact file to store the address information with which the rule will use to send the notification. You can separate your events by:



- ▶ Host name: Send each event to a recipient based on the hostname slot in the event. For example, a event coming from host name rduwws01 goes to a support team in Raleigh. This information is included in the fact file.
- ▶ Severity: You can send different event severities to different groups.
- ▶ TEC\_Class: For example, you can send notifications for all TEC\_ events to the group that is responsible for maintaining IBM Tivoli Enterprise Console.

Here is an example of how we can modify the notfiy.rls rule to consult a fact file for notification information by using the hostname slot. First, we create our fact file /usr/local/Tivoli/custom/facts/tec\_r.address.pro with our address information, as shown in Example 6-24.

---

*Example 6-24   tec\_r.address.pro fact file*

---

```
server('rduatf01','Bob','bob@company.com')
server('dtmwas01','Peter','peter@company.com')
server('sapwas01','Bill','bill@company.com')
server('phiwas01','Jackie','jackie@company.com')
```

---

Assuming that our fact file is already compiled, we modify our rule. We now consult our fact file and use it to fill in the admin and address slots that we added to our events (see Example 6-25).

---

*Example 6-25   Rule modification to use a fact file for notification*

---

```
rule:
load_addresses:
(event: _event of_class 'TEC_Start'
where [ ],
reception_action: load_address_fact_file:
(
reconsult('/usr/local/Tivoli/custom/facts/address')
)
).
```

```
rule:
set_admin_and_address:
(
event: _event of_class _class
where [
hostname: _hostname
],
reception_action:
```

```

(
server(_hostname, _admin, _address),
bo_set_slotval(_event, admin, _admin),
bo_set_slotval(_event, address, _address),
re_mark_as_modified(_event, _)
)
).

rule:
notify_for_fatal_events:
(
  event: _event of_class _class
  where [
    severity: _severity equals 'FATAL',
    status: equals 'OPEN',
    msg: _msg
  ],
  reception_action:
  (
    exec_program(_event, 'scripts/notify_support.sh', '', [], 'NO'),
    commit_action
  )
).

```

---

With this rule, any fatal event that comes into IBM Tivoli Enterprise Console consults our fact file. If there is an entry in that file for the host name of the server from which the event came, it sets the admin and address slots from that entry.

Figure 6-22 shows how the event attributes may appear if we send a fatal event from the server sapwas01. Notice that the address slot is populated with Bill's e-mail address and the admin slot is set to Bill.

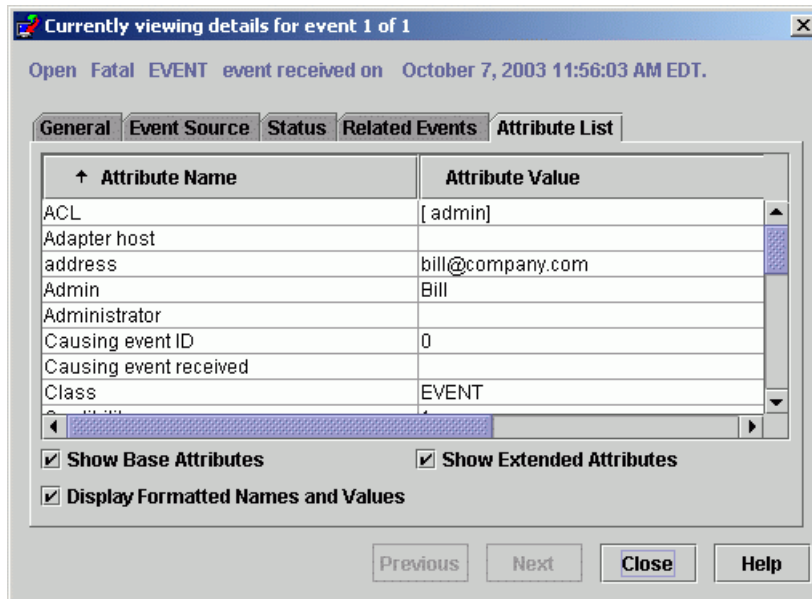


Figure 6-22 Event details after using a fact file for notification

Now we can modify the same script we used earlier to actually send the notifications. We change the name from `notify_bob.sh` to `notify_support.sh` because everybody is going to be notified from this script, not just Bob. Example 6-26 shows how the script should look.

#### Example 6-26 `notify_support.sh` script

```
#!/usr/bin/ksh
NOW=`date`
echo $NOW, $hostname, $class, $msg, $date >>/tmp/notify_bob.out

PATH=/bin:/usr/bin:/usr/ucb:/usr/lib
export PATH

sendmail -F TEC -t << __EOF__
To: $address
Subject: Notification from TEC about $hostname
Dear $admin
$msg happened on $hostname, $admin please fix this, or we will send Bob after
you.
__EOF__
```

We changed the To: line from Bob's e-mail address to \$address which is populated from our fact file based on the host name from which the event came. We also added the Dear \$admin line, which is also populated from our fact file.

#### **6.4.4 IBM Tivoli Monitoring**

IBM Tivoli Monitoring has two forms of notification:

- ▶ IBM Tivoli Monitoring Web Health Console
- ▶ Executing Tivoli Tasks upon an event situation

##### **Web Health Console**

IBM Tivoli Monitoring's main form of notification is the Web Health Console. The Web Health Console allow you to view the general health of a resource that you are monitoring.

You can use the Web Health Console to check, display, and analyze the status and health of any endpoint with profiles and resource models. Status reflects the state of the endpoint displayed on the Web Health Console, such as running or stopped. Health is a numeric value determined by resource model settings. You can also use the Web Health Console to work with real-time or historical data from an endpoint that is logged to the IBM Tivoli Monitoring database.

You can use the diagnostic and monitoring capabilities of the Web Health Console to perform targeted analysis of problems associated with individual endpoints when an event is sent to the Tivoli Enterprise Console. Use the online and historical data to follow up on specific problems with single endpoints. Figure 6-23 shows an example of the Web Health Console.

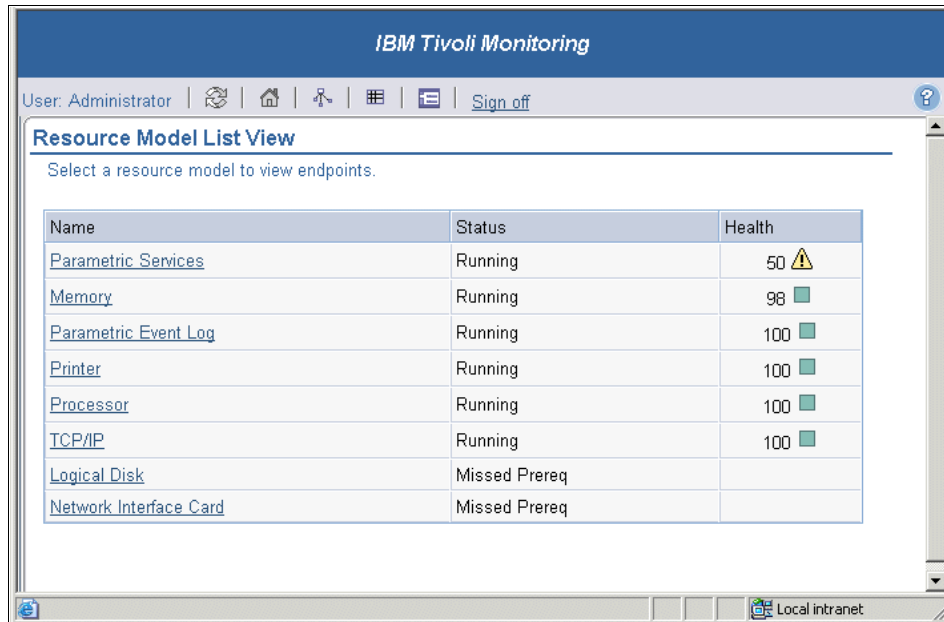


Figure 6-23 IBM Tivoli Monitoring Web Health Console Example

## Tasks

Built into IBM Tivoli Monitoring is the ability to run two Tivoli tasks to perform notifications:

- ▶ `dmae_mn_send_notice`: This task creates a Tivoli Notice that you can view via the Tivoli Desktop Notices window.
- ▶ `dmae_mn_send_email`: This task sends an e-mail to a specified e-mail address.

To enable one of these tasks within IBM Tivoli Monitoring, you must follow these steps:

1. Open a resource model via the Tivoli Desktop. Click the **Indications** button.
2. The Indications and Actions window opens. Select the indication about which you want to be notified and click the **Tasks** button.
3. The Tasks window (Figure 6-24) opens. Under Libraries, select **IBM Tivoli Monitoring Utility Tasks**. Under the Tasks pane, select the corresponding task that you desire either for a Tivoli notice or an e-mail. Then, click the **Select Task** button.

4. You now see a window to configure the task. The notice task asks for a notice group to send the notice to, as well as the priority. The e-mail task asks for the e-mail address or addresses where you want to send the notification.

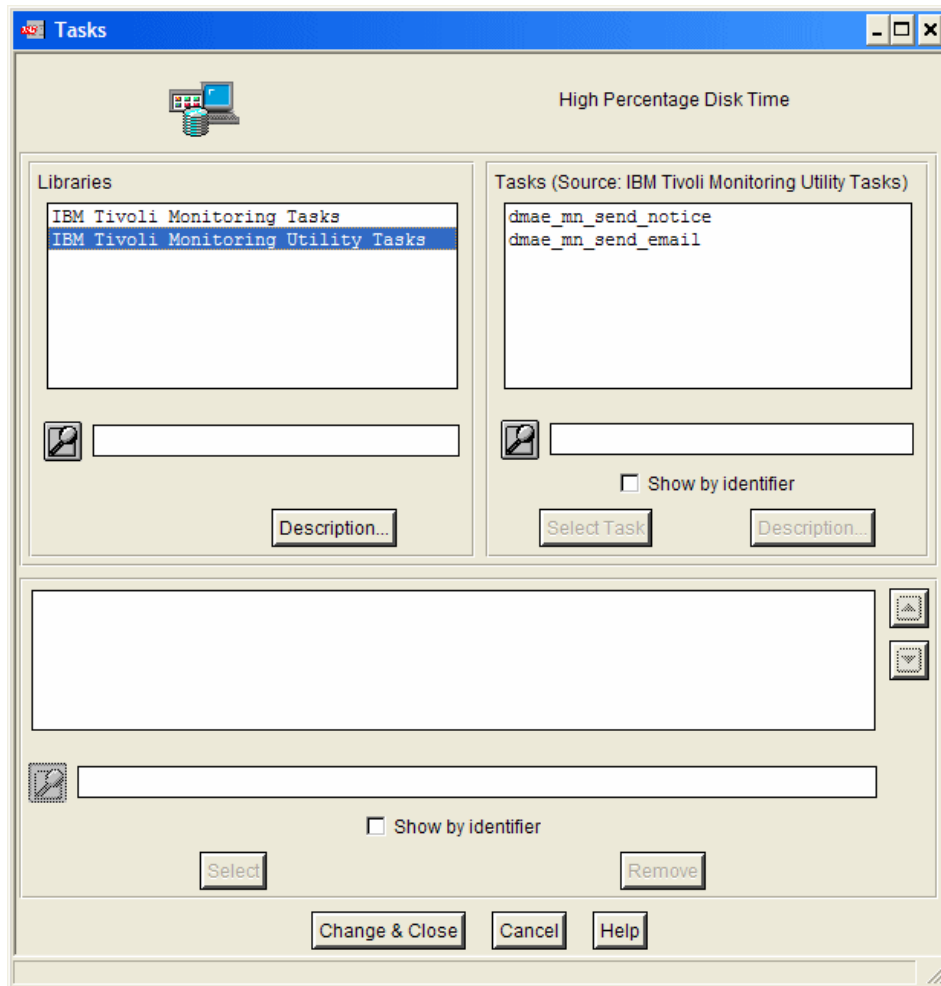


Figure 6-24 IBM Tivoli Monitoring resource model tasks notifications

## 6.5 Escalation

In this section, we discuss policies that you should define for proper handling of event escalation, and how severities are set and escalated using NetView and IBM Tivoli Enterprise Console.

# 6.5.1 Severities

Chapter 1, “Introduction to event management” on page 1, presents three types of escalation. The escalation type, increase the severities in response to a worsening condition or based on business impact. They also ensure the handling of problems by following a notification chain if they are not handled in a timely fashion.

For these types of escalation, policies must be clearly defined. This section presents some samples that demonstrate guidelines that we used in our lab case study.

## Severity mapping between tools

Often the severities employed by various software products differ. Administrators who set up monitoring often do not know which severities to use for events generated by their monitoring tools or the service-level implications of those severities.

Therefore, we recommend this *best practice*. To provide consistency in event handling, map the severities used by the different monitoring tools to each other and to service-level agreements (SLA).

This ensures that the event is assigned the proper severity when events are forwarded between event processors. It also ensures that administrators using different tools to manage problems respond with the appropriate sense of urgency.

Table 6-7 shows a sample mapping of severities between NetView, IBM Tivoli Enterprise Console, and the trouble-ticketing system. It also shows the relationship between severities and SLA response times.

Table 6-7 Severity mapping between IBM Tivoli NetView, Enterprise Console, trouble-ticketing systems

NetView severity	NetView trap color	IBM Tivoli Enterprise Console severity	IBM Tivoli Enterprise Console event colors (foreground on background)	Troubleticket severity	Service-level agreement
Indeterminate	Tan	Unknown	Black on blue	4	Respond within 2 days
Cleared	Tan	Harmless	Black on green	Close	
Warning	Yellow	Warning	Black on yellow	3	Respond within one day

NetView severity	NetView trap color	IBM Tivoli Enterprise Console severity	IBM Tivoli Enterprise Console event colors (foreground on background)	Trouble ticket severity	Service-level agreement
Minor	Green	Minor	Black on orange	2	Respond within four hours
Critical	Red	Critical	Black on red	1	Respond within two hours
Major	Orange	Fatal	White on black	1	Respond within one hours

## Setting severities

After an organization establishes which severities to use and when, the monitoring tools must be configured to set the appropriate values. This section discusses how NetView and IBM Tivoli Enterprise Console can set event severities.

### *Setting trap severity in NetView for UNIX*

Using our sample, suppose NetView receives a Router Down trap. The router is a key networking component whose failure can affect many users. Therefore, our SLA within the organization states that it should be fixed within one hour.

We must configure the trap in NetView with Major severity using the Event Configuration window. Follow these steps:

1. Issue the **xnmtrap** command. Or from the NetView console, select **Options → Event Configuration → Trap Customization: SNMP**.



2. The Event Configuration window (Figure 6-25) opens.
  - a. Under Enterprise Identification, select the enterprise that generates the trap. NetView generates the Router Down trap, so we choose **netView6000** as the enterprise.
  - b. Under Event Identification, select the trap itself from the list.
  - c. Click **Modify**.

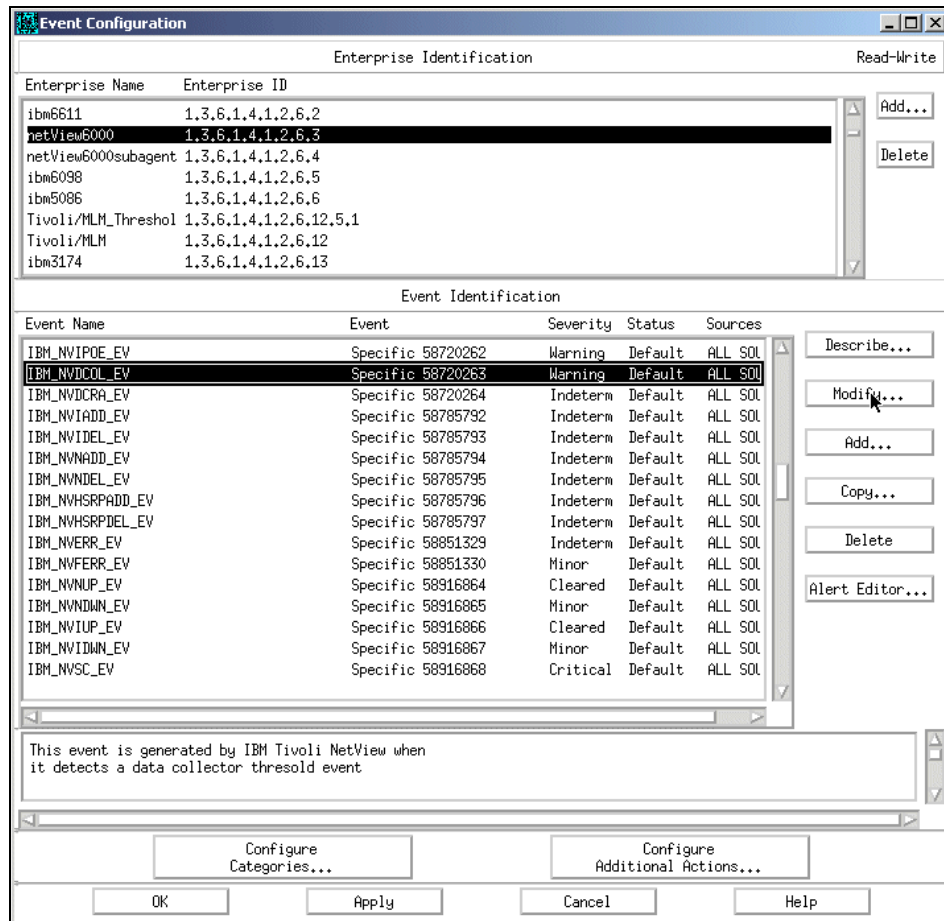


Figure 6-25 Event Configuration window in NetView

3. In the Modify Event window (Figure 6-26), you now see information about the trap. Select the appropriate severity from the list and click **OK**.

The screenshot shows the 'Modify Event' window with the following fields and options:

- Event Name:** ROUTDOWN\_EV
- Generic Trap:** Enterprise Specific (checkbox)
- Specific Trap Number:** 58916971
- Event Description:** This event is generated by IBM Tivoli NetView when it detects that a router is down. The data passed with the event are: 1) ID of application sending the event
- Event Sources (nodes):** (all sources (nodes) if list is empty)
- Source:** (empty text box)
- T/EC Event Class:** TEC\_ITS\_ROUTER\_STATUS
- Event Category:** Status Events (dropdown)
- Status:** Default Status (dropdown)
- Severity:** A dropdown menu is open showing options: Cleared, Indeterminate, Warning, Minor, Critical, and Major. 'Major' is selected.
- Source Character:** N (checkbox)
- Do Not Forward Trap:** (checkbox)
- Event Log Message:** \$3
- Popup Notification (Optional):** (empty text box)
- Command for Automatic Action (Optional):** (empty text box)
- Buttons:** OK, Reset, Cancel, Help

Figure 6-26 Setting the severity of a NetView trap

4. In the Event Configuration window, click **Apply** to force trapd to read the changed configuration.

Any router down traps now appear with Major severity in the NetView event displays, as shown in Figure 6-27.

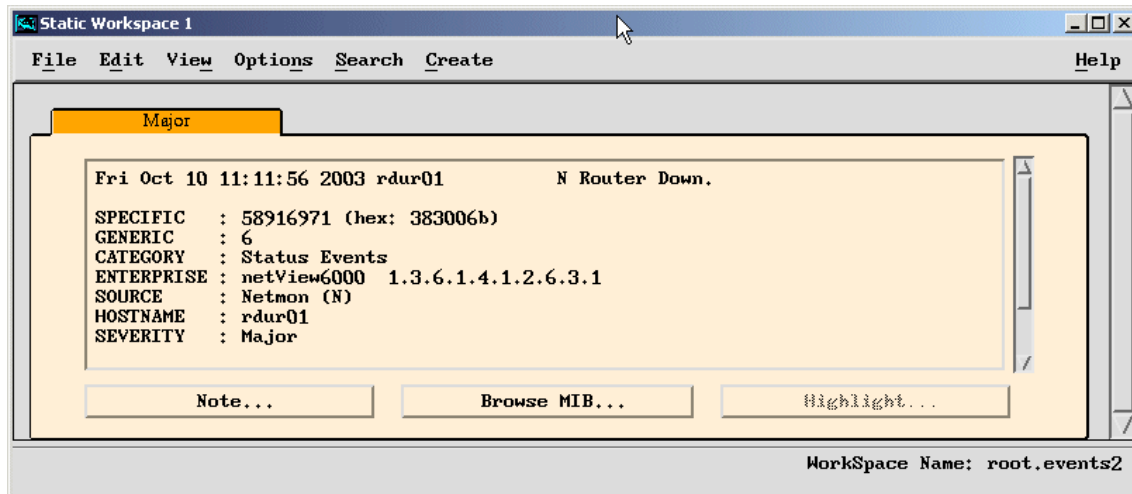


Figure 6-27 Router Down trap now shows Major severity

### **Setting trap severity in NetView for Windows**

In NetView for Windows, trap severities are set in the trap configuration window. Follow these steps:

1. From either the NetView main menu or Event Browser windows, select **Options → Trap Settings**, or execute `\usr\0V\bin\trap.exe`.

2. The Trap Settings window (Figure 6-28) opens. Under Select an enterprise, select the desired enterprise. Its associated traps are displayed in the Select a trap section of the window. Highlight the trap for which you want to set severity, and click **Properties**.

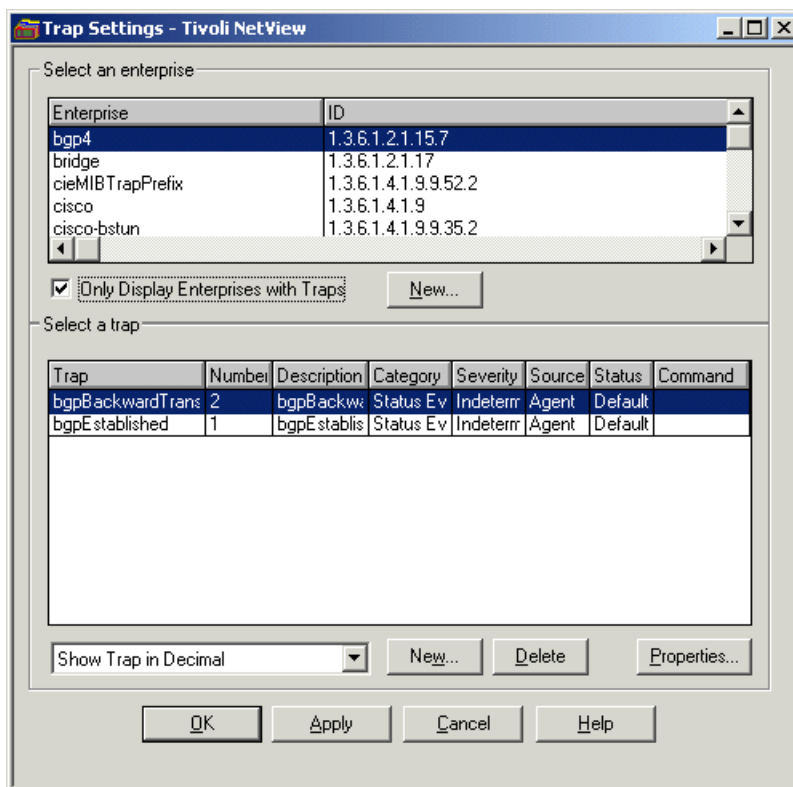


Figure 6-28 NetView for Windows trap configuration window

3. The Trap Properties window (Figure 6-29) opens. You can see the valid options under With Severity. Select the desired severity and click **OK** or **Apply** on this window for the change to take effect.
4. On the Trap Settings window, click **OK** or **Apply** for the change to take effect.

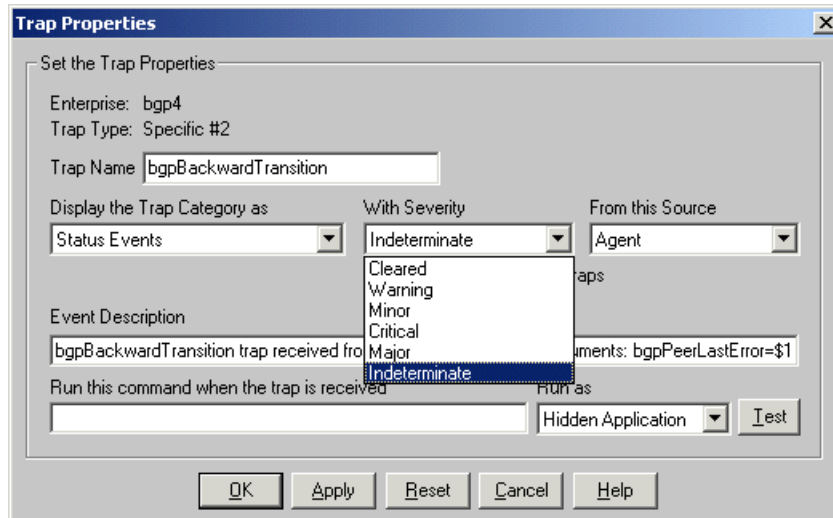


Figure 6-29 Setting trap severity in NetView for Windows

### Setting event severity in NetView for UNIX

From the Event Configuration window, it is possible to set the severity of the IBM Tivoli Enterprise Console event that is generated when the trap is forwarded to IBM Tivoli Enterprise Console using nvserverd.

1. When modifying the event, click **T/EC Slot Map**.
2. The Edit Trap Slot Map window (Figure 6-30) opens. If severity is not listed in the list of slot variables, click **Add**. For Slot Name, type severity. In the Slot Value field, type the desired value. Click **OK**.

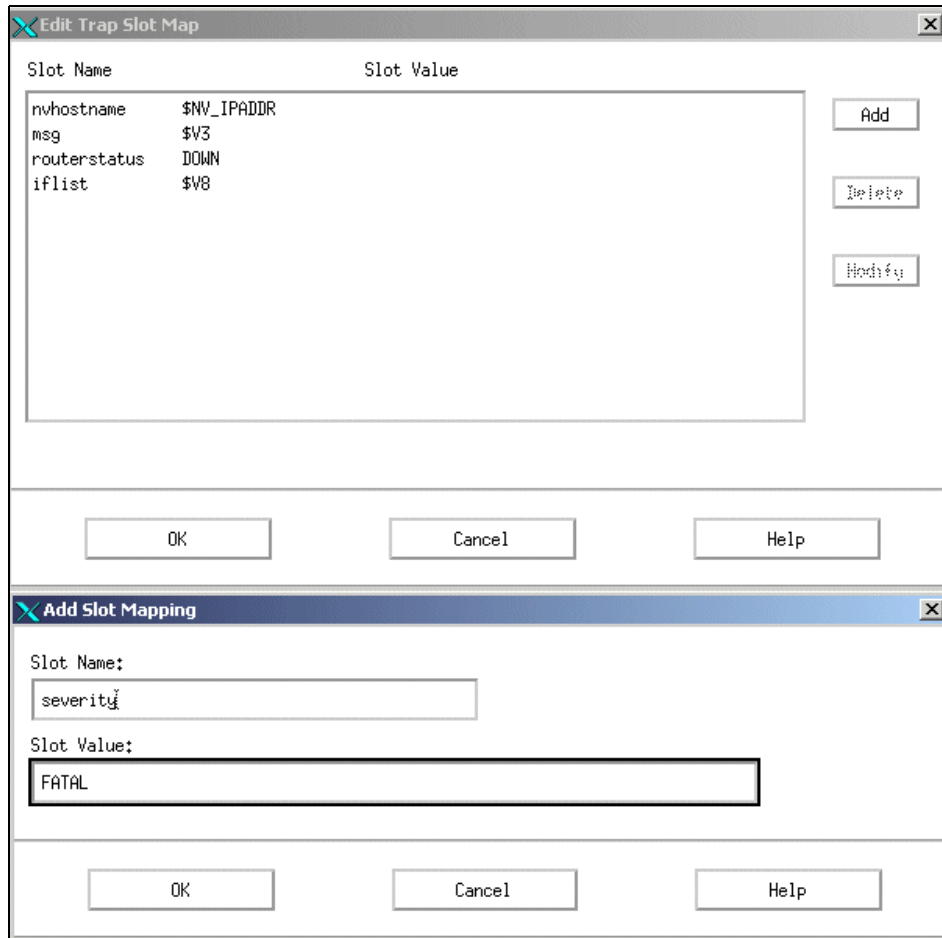


Figure 6-30 Setting IBM Tivoli Enterprise Console severity in NetView

3. Click **OK** two more times.
4. Click **Apply** for the setting to take effect.

When you enter the **wtdumpr1** command on the IBM Tivoli Enterprise Console event server, you see that the event arrived at IBM Tivoli Enterprise Console with severity FATAL as shown in Example 6-27.

*Example 6-27 wtdumpr1 output showing FATAL Router Down event*

---

```
1~2863~65538~1065810029(Oct 10 14:20:29 2003)
### EVENT ###
TEC_ITS_ROUTER_STATUS;source=nvserverd;nvhostname=9.24.106.154;category=2;msg="
Router Down.";date="10/10/03
14:22:09";routerstatus=DOWN;status=OPEN;adapter_host
=9.24.106.154;hostname=rdur01;origin=9.24.106.145;sub_source=N;iflist=['9.24.10
6.145', '9.24.106.130'];severity=FATAL;END

### END EVENT ###
PROCESSED
```

---

**Setting event severity in NetView for Windows**

You can set the severity of a trap to be forwarded to IBM Tivoli Enterprise Console using the NetView adapter by modifying the `tecad_nv6k.cds` file. The severity keyword and value are added to the MAP part of the trap definition, as shown in Example 6-28.

*Example 6-28 Setting severity in tecad\_nv6k.cds*

---

```
text omitted

# CLASS Env_Temp_Hot_CBT
# 'envTempHot' traps are mapped to 'envTemperature' OPEN/CRITICAL
CLASS ENV_TEMPERATURE_CBT
SELECT
    1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.52") ;
    2: $SPECIFIC = 282 ;
    3: ATTR(=,"boardIndex") ;
    MAP
        boardIndex = $V3 ;
        sub_origin = PRINTF("board %s", $V3) ;
        severity = CRITICAL ;
END

# CLASS Env_Temp_Normal_CBT
# 'envTempNormal' traps are mapped to 'envTemperature' CLOSED
CLASS ENV_TEMPERATURE_CBT
SELECT
```

```

1: ATTR(=,$ENTERPRISE), VALUE(PREFIX, "1.3.6.1.4.1.52") ;
2: $SPECIFIC = 284 ;
3: ATTR(=,"boardIndex") ;
MAP
boardIndex = $V3 ;
sub_origin = PRINTF("board %s", $V3) ;
severity = HARMLESS ;
END

```

text omitted

---

The severity is set to CRITICAL for Cabletron temperature hot environmental traps. It is set to HARMLESS for Cabletron temperature normal environmental events.

In this example, two different NetView traps (envTempHot and envTempNormal) are mapped to the same class, ENV\_TEMPERATURE\_CBT.

### ***Setting severity in IBM Tivoli Enterprise Console***

If the source does not generate an event with the desired severity, you can specify IBM Tivoli Enterprise Console to set it. There are two ways you can set severity in IBM Tivoli Enterprise Console.

The first method is to use the default settings from an IBM Tivoli Enterprise Console class definition. If an event is generated without specifying severity, IBM Tivoli Enterprise Console enters the severity based on the default value specified in the BAROC files. When the class definition itself does not specify a default, the value is inherited from a superclass to which the class belongs.

Out of the box, NetView enters information into its trapd.conf file to map traps to IBM Tivoli Enterprise Console classes, but does not supply severities for the events. Therefore, events received from NetView are initially assigned the default severity for the class. In our example, the IBM Tivoli Enterprise Console class, TEC\_ITS\_ROUTER\_STATUS, is defined in the netview.baroc file (Example 6-29) without a default severity. It is part of superclass TEC\_ITS\_L3\_STATUS, which specifies a default severity of WARNING.

If you do not add severities to the trap configuration definitions in NetView, all traps are sent to IBM Tivoli Enterprise Console as severity WARNING.



text omitted

```
TEC_CLASS: TEC_ITS_BASE ISA EVENT
DEFINES {

    # The default severity assigned to the event.
    severity: default = WARNING; # (from EVENT)

    # The default source assigned to the event.
    source: default = 'ITS'; # (from EVENT)

    # The default subsource assigned to the event.
    sub_source: default = 'N/A'; # (from EVENT)

    # The network node name associated with the event.
    # hostname: STRING (from EVENT)

    # The event textual description.
    # msg: STRING (from EVENT)

    # The NetView server hostname associated with the event.
    nvhostname: STRING;

    # An unique network id to identify management sites (placeholder).
    networkid: STRING;

    # This field always contains the node's IP address as viewed by NetView.
    hostaddr: STRING;

    # Category
    category: TEC_ITS_CategoryE, default = "undefined";

    # Slots for default event (that is, the raw trap data)
    nv_enterprise: STRING;
    nv_generic:    INT32;
    nv_specific:   INT32;
    nv_var1:       STRING;
    nv_var2:       STRING;
    nv_var3:       STRING;
    nv_var4:       STRING;
    nv_var5:       STRING;
    nv_var6:       STRING;
    nv_var7:       STRING;
    nv_var8:       STRING;
    nv_var9:       STRING;
    nv_var10:      STRING;
    nv_var11:      STRING;
```

```

        nv_var12:    STRING;
        nv_var13:    STRING;
        nv_var14:    STRING;
        nv_var15:    STRING;

    };
END

text omitted

TEC_CLASS: TEC_ITS_L3_STATUS ISA TEC_ITS_BASE
    DEFINES {

        # The default severity assigned to the event.
        severity: default = WARNING; # (from EVENT)

    };
END

text omitted

TEC_CLASS: TEC_ITS_ROUTER_STATUS ISA TEC_ITS_L3_STATUS
    DEFINES {

        # The status associated with the network router.
        routerstatus: TEC_ITS_RouterStatusE,
                        default = UP;

        # The list of comma separated IP addresses of network interfaces
        # attached to this router.
        iflist: LIST_OF STRING,
                default = [];

    };
END

text omitted

```

---

The second method is to use the default setting from the EVENT definition. When IBM Tivoli Enterprise Console receives an event that does not have a severity associated with it, it first looks at the BAROC definition for the IBM Tivoli Enterprise Console classes. Assuming that does not set a severity either, IBM Tivoli Enterprise Console assigns the WARNING severity, as defined in the EVENT definition in root.baroc file.

When processing a trap or event, it is possible to override its severity. Increasing the severity of an event is known as *escalation*. In 6.5.2, “Escalating events with NetView” on page 279, and “Escalating events with IBM Tivoli Enterprise

Console event server” on page 285, we discuss how to configure these products to perform escalation. Out of the box, NetView uses this method to modify the default severity of WARNING for a subset of events.

**Setting severity in trouble-ticketing system**

How trouble-ticketing systems set the severity of tickets differs based on the tool. Some may perform the mapping in the IBM Tivoli Enterprise Console event server. Others do it in the trouble ticketing software itself.

In our lab environment, we implemented Peregrine’s trouble-ticketing system and configured the hub IBM Tivoli Enterprise Console to open trouble tickets. The severity initially assigned to a ticket is assigned by a script that maps severity to a trouble ticket. This script is called by an IBM Tivoli Enterprise Console rule when processing events for which a ticket is desired.

An excerpt of the script is included in Example 6-30. It maps severities to priority codes as shown in Table 6-8.

*Table 6-8 IBM Tivoli Enterprise Console severity and Peregrine priority codes*

IBM Tivoli Enterprise Console severity	Peregrine priority code
FATAL	1
CRITICAL	2
MINOR	3
WARNING	4
HARMLESS	4

*Example 6-30 Peregrine sceventin.sh script*

```
#!/bin/sh
#####
#
# (C) COPYRIGHT Peregrine Systems, Inc. 1996, 1997
# Portions (C) COPYRIGHT Tivoli Systems, Inc. 1996, 1997
# All Rights Reserved
# Licensed Material - Property of Peregrine Systems, Inc.
#
#####
#
# This shell script may be used to perform expressions prior to mapping
# Currently it converts Tivoli severity to ServiceCenter priorityCode
# and converts environment names to a standard across Tivoli releases
#
```

```
#####

#
#       The standard directory and the /Plus override script
#
ETC=/etc/Tivoli

#
#       If this is NT, then etc is in a different location
#
if [ x"$OS\" = x"Windows_NT\" ] ; then
    ETC=$SystemRoot/system32/drivers/etc/Tivoli
fi

. $ETC/setup_env.sh

OSERV=`objcall 0.0.0 get_oserv`
export OSERV
INST_DIR=`objcall $OSERV query install_dir`
INST_DIR=`echo $INST_DIR | tr "\\\\" "/"`
export INST_DIR

PLUSDIR=$INST_DIR/generic_unix/TME/PLUS
LINKDIR=$PLUSDIR/LINK
PRODDIR=$PLUSDIR/SERVICECENTER
export PRODDIR

PATH=$LINKDIR:$PRODDIR:$PATH
export PATH

cd $PRODDIR

. ./scplustdirs.sh

EVENT_CLASS=${EVENT_CLASS:-$CLASS_NAME}
hostname=${hostname:-$HOSTNAME}
severity=${severity:-$SEVERITY}
msg=${msg:-$MSG}
source=${source:-$SOURCE}
sub_source=${sub_source:-$SUB_SOURCE}
origin=${origin:-$ORIGIN}
sub_origin=${sub_origin:-$SUB_ORIGIN}
date_event=${date_event:-$DATE_EVENT}
date_reception=${date_reception:-$DATE_RECEPTION}
server_handle=${server_handle:-$SRVR_HANDLE}
event_handle=${event_handle:-$HANDLE}
adapter_host=${adapter_host:-$ADAPTER_HOST}

# the next two lines are a fix to a problem in
```

```

# tivoli that date_reception is presented in hex
# rather than decimal

temp=~$SCPLUSHOME/bin/hextodec $date_reception`
date_reception=$temp

#end of fix. this code will be removed when pmr39520 is resolved

export EVENT_CLASS hostname severity msg source sub_source origin
export sub_origin date_event date_reception adapter_host event_handle
export server_handle

    if [ $severity = "FATAL" ]; then
        priorityCode="1"
    fi
    if [ $severity = "CRITICAL" ]; then
        priorityCode="2"
    fi
    if [ $severity = "MINOR" ]; then
        priorityCode="3"
    fi
    if [ $severity = "WARNING" ]; then
        priorityCode="4"
    fi
    if [ $severity = "HARMLESS" -o $severity = "UNKNOWN" ]; then
        priorityCode="4"
    fi

export priorityCode

$SCPLUSHOME/bin/sceventin >> $SCPLUSHOME/sctivoli.log

exit 0

```

---

## Best practice for setting severities

With so many options available to set severities, the question becomes: “Which is best?” When determining which method to use, keep in mind ease of use, capabilities of the event management tools, and processing overhead.

We suggest as a *best practice* that, in general, you set severity as close to the event source as possible.

Processing is required to format and generate the event at the source. The cycles required to add one additional field to the format are generally negligible.

This method is also easiest to debug. Using the severity mapping defined earlier, it is easy to determine what the severity of an event is along any path in the event

processor hierarchy. Many actions, such as forwarding events, opening trouble tickets, and escalating unhandled events, are predicated upon the event's initial severity. Therefore, debugging those actions becomes easier.

Obviously, if the event source is incapable of setting the event severity, it needs to be performed by an event processor in the hierarchy. Also, if the effort required to set the severity in the source is large, it may be easier to allow another event processor to perform the function.

## Notification chain

Another important event management policy deals with the chain of people to notify when an event is not addressed in a predefined time frame. The policy should include event types, time frames, and responsible personnel.

Here is a simple example of an escalation policy:

1. If an event is not acknowledged within half the time specified in the SLA to resolve that condition, increase its severity to the next level and notify the same people who were informed about the problem initially.
2. If an event is not resolved within the time specified in the SLA for that condition, increase its severity to the next level and notify the same people who were informed about the problem initially.
3. Repeat this procedure for each increased severity to which the event is assigned.
4. When the event is escalated to the highest defined severity, notify management of the team responsible for the event.

Using the example in “Severity mapping between tools” on page 263, assume a node down trap is given an initial severity of *minor* based on the importance of the server referenced. It is assigned to the server support staff responsible for the machine. The SLA states that the problem should be resolved within four hours.

If the event is not acknowledged within two hours, it is escalated to a CRITICAL status. If it is acknowledged within two hours but not resolved within four hours, the event is escalated to a CRITICAL status, and the server group is informed about the increased importance of handling the problem. If two more hours pass without a resolution, the problem is escalated to a FATAL severity, and the server team and their management are informed.

See “Escalation using `escalate.rls`” on page 286 for an example of how you can implement this using the supplied IBM Tivoli Enterprise Console rule.

## 6.5.2 Escalating events with NetView

Best practices dictate that you should perform escalation using IBM Tivoli Enterprise Console for worsening problems or using the trouble-ticketing system for problems that have not been addressed within a predefined time period. However, in environments where NetView is the only monitoring tool, it may be desirable to have it perform trap escalation. The NetView rule set is used to perform this function.

### Override ruleset node

One of the nodes available to the NetView ruleset editor is Override. This provides the capability to change the object status or severity assigned to a specific event and update applications that are registered to receive the output of the rule set. Typically, the Event Display application is registered to receive the output.

Figure 6-35 on page 283 contains the following relevant fields:

- ▶ **Status:** Specifies the new object status to be associated with this event. You can click No override if you do not want to change the status. The Event Display application updates the object status to this value.
- ▶ **Severity:** Specifies the new severity level to be used for this event. You can click No override if you do not want to change the severity level. A trap that is processed through this node is marked so that it is not handled by the default processing action specified for this rule.

### Business impact escalation rule

An example of business impact escalation is increasing the severity of traps based on device type. A rule set may be coded in NetView that uses a Query Database Field to determine the device type, and then uses an override, depending on the type.

NetView issues Node Down traps when switches and servers fail. By default, the traps are assigned a Minor severity. Assume that an organization decides to treat switch failures as more critical than other node outages. The rule set shown in Figure 6-31 accomplishes this.

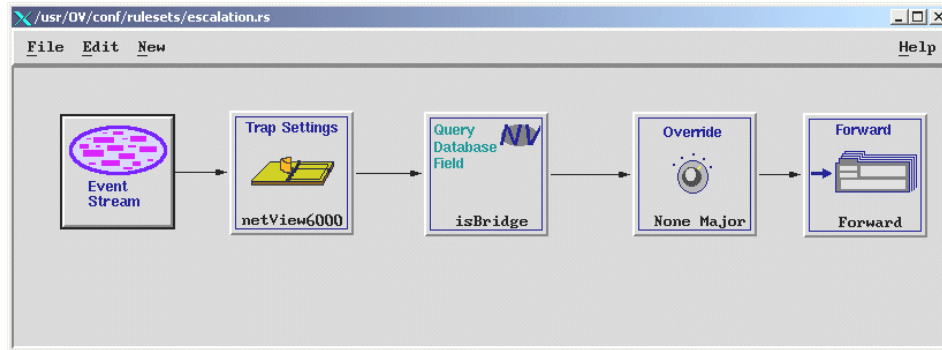


Figure 6-31 Escalation rule set in NetView

For the event stream to pass events, by default, follow these steps:

1. Right-click the **Event Stream node** and select **Edit**.
2. In the Ruleset window (Figure 6-32), click **Pass** and then click **OK**. This ensures that traps, other than the one in question, are passed by default.

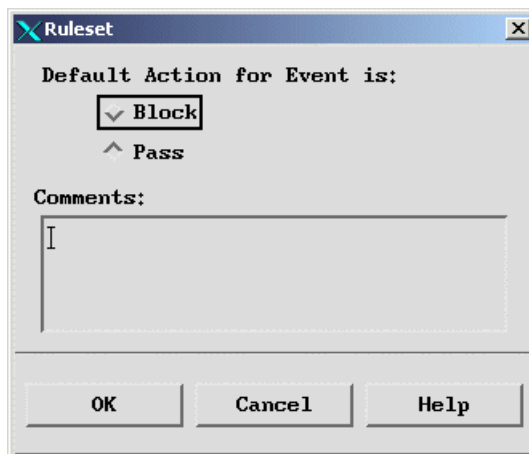


Figure 6-32 Event Stream default action

3. Add the Trap Settings node to check for the node down trap.
  - a. Right-click the **Trap Settings node** and select **Edit**.
  - b. In the Trap Settings window (Figure 6-33), choose the enterprise and trap, and click **OK**.

Upon matching, it sends the trap onto the Query Database Field. Note that traps that do not match are merely forwarded, as set by the default action for the event



stream. The Query Database Field checks to see if the isBridge field is set for the object. Switches have this field set.

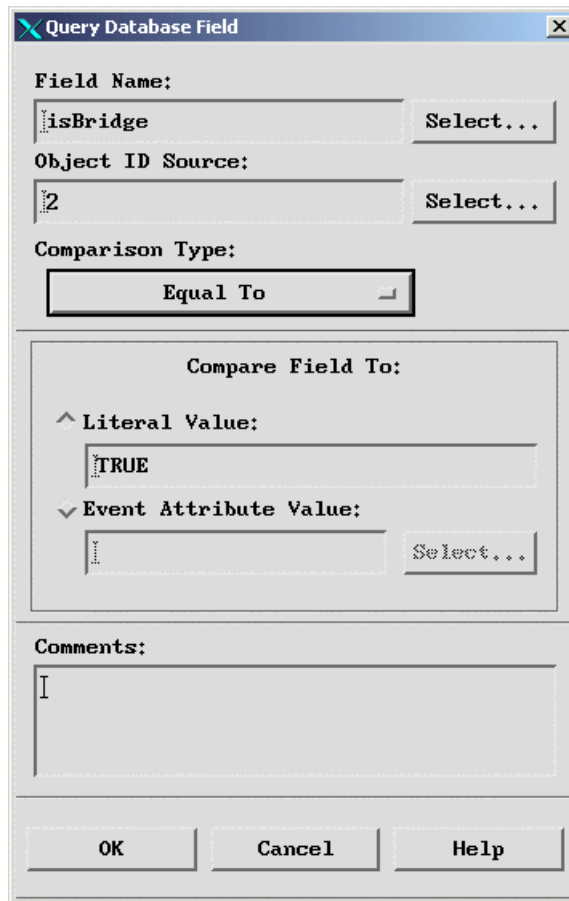
The image shows a 'Trap Settings' dialog box with the following sections:

- Enterprise Name:** netView6000
- Enterprise ID:** 1.3.6.1.4.1.2.6.3
- Event Name:** IBM\_NVNDWN\_EV
- Specific:** Specific 58916865
- Trap Description:** This event is generated by IBM Tivoli NetView when it detects a node is down. The data passed with the event are:
- Comparison Type:** Equal To
- Comments:** (Empty text area)

Buttons at the bottom: OK, Cancel, Help.

Figure 6-33 Trap setting for escalating a Node Down trap

Again, edit the node. Right-click the node and select **Edit**. As shown in the Query Database Field window (Figure 6-34), for Field Name, click the **Select...** button and select **isBridge**. The Object ID Source is set to 2. This indicates that the query should be performed for the object referenced in variable binding 2 (host name to which the trap applies) from the trap. See Appendix A in *Tivoli NetView for UNIX Administrator's Guide, Version 7.1*, SC31-8892, for a list of the variables passed in NetView internal traps.



The image shows a dialog box titled "Query Database Field". It contains several sections for configuring a query. The "Field Name:" section has a text box with "isBridge" and a "Select..." button. The "Object ID Source:" section has a text box with "2" and a "Select..." button. The "Comparison Type:" section has a dropdown menu set to "Equal To". The "Compare Field To:" section has two options: "Literal Value:" with a text box containing "TRUE", and "Event Attribute Value:" with a text box and a "Select..." button. The "Event Attribute Value:" option is currently selected. At the bottom, there is a "Comments:" section with a large text area. The dialog box has "OK", "Cancel", and "Help" buttons at the bottom.

Field Name:		
isBridge	Select...	
Object ID Source:		
2	Select...	
Comparison Type:		
Equal To		
Compare Field To:		
^ Literal Value:		
TRUE		
v Event Attribute Value:		
	Select...	
Comments:		
OK	Cancel	Help

Figure 6-34 Query Database Field settings for escalation

The Override node is edited and the severity for the trap is set to Major, as shown in Figure 6-35. No change was made to the object status.

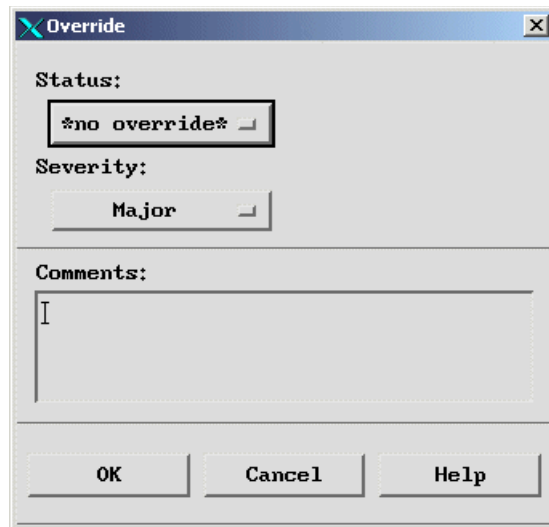


Figure 6-35 Override node used in escalation rule

To test the rule set, we created a dynamic event display window, as shown in Figure 6-36, and selected the escalation.rs rule for it. See Chapter 4, “Using Dynamic and Static Work Spaces”, in *Tivoli NetView for UNIX User's Guide for Beginners, Version 7.1*, SC31-8891, for information about how to create dynamic work spaces.

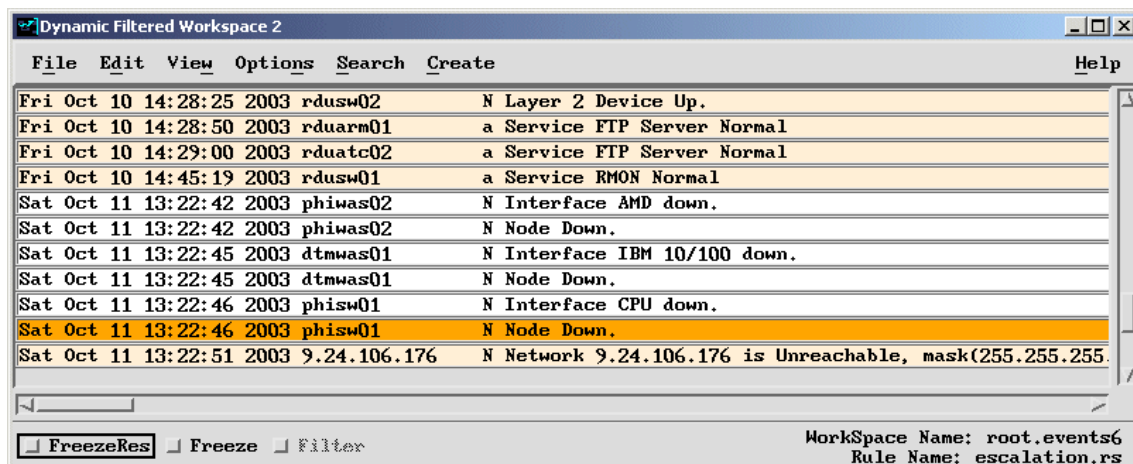


Figure 6-36 Dynamic filtered workspace for escalation rule set

The event now displays with Major severity, as shown in Figure 6-37.

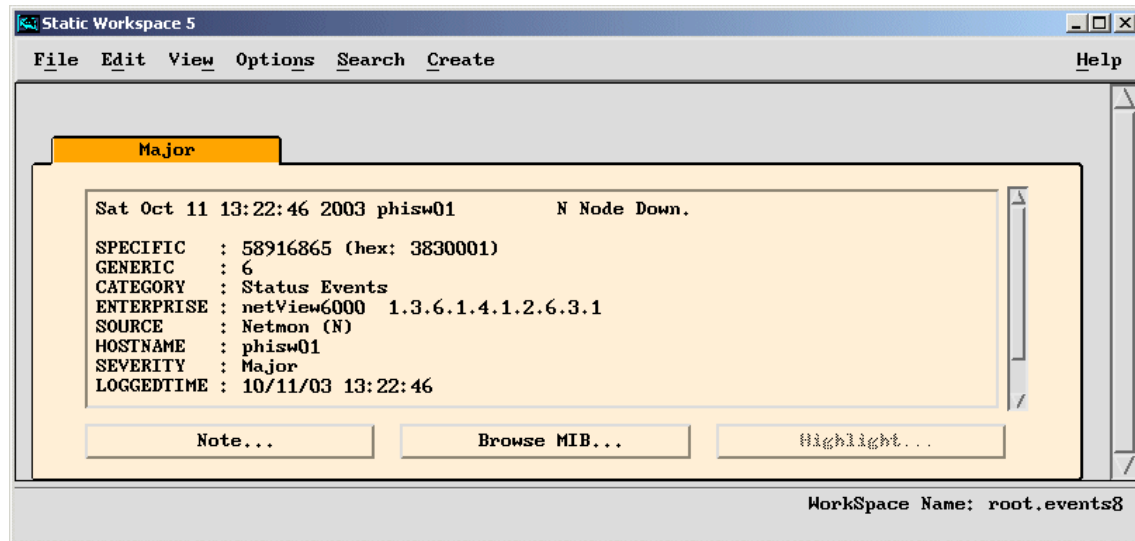


Figure 6-37 Node down with escalated (major) severity

## Worsening condition escalation

NetView can hold traps for a specified time to see if another event is received. It can also resolve an event upon receipt of another.

However, it is incapable of reprocessing a trap that has already been processed. Therefore, there is currently no way for NetView to escalate the severity of a trap to indicate a worsening condition.

## State correlation

The IBM Tivoli Enterprise Console state correlation gateway can handle escalating or worsening events that pertain only to criteria matching the XML rule and in the specified time period contained in the rule. This rule, or these rules, can specify a time period to wait for other events. From there, it can escalate the severity of the event before it is sent to the IBM Tivoli Enterprise Console event server (if another event is received that matches this define criteria). This is a one time only operation.

After the correlated event is sent to the IBM Tivoli Enterprise Console event server, the rule resets itself, and waits for another event, which matches its criteria. If the next event, which matches the criteria, represents a worsening condition, it is treated as a new event, and the correlation in the rule starts over.

We recommend that you use both the state correlation engine on the gateway and the IBM Tivoli Enterprise Console event server rule engine in conjunction with each other to ensure proper escalation of worsening conditions.

## Escalating events with IBM Tivoli Enterprise Console event server

The severity of events can be escalated using IBM Tivoli Enterprise Console rules. This section reviews several rules that are supplied with IBM Tivoli Enterprise Console and NetView. It explains how they are used to escalate the severity of events. Plus it describes the predicates that are available in IBM Tivoli Enterprise Console to code your own escalation rules.

### ***Escalation in netview.rls***

The default installation for NetView configures event forwarding, so that all traps sent to IBM Tivoli Enterprise Console from NetView have a WARNING severity. The rules in the netview.rls rule set adjust the severity of the events according to Table 6-9.

*Table 6-9 Rules from the netview.rls rule set*

Rule	Function
router_raise	Raises the severity of router down events to CRITICAL.
interface_lower	Lowers the severity of interface up events to HARMLESS.
isdn_lower	Lowers the severity of ISDN active events to HARMLESS.
snmp_lower	Lowers the severity of SNMP collect re-arm events to HARMLESS.
node_lower	Lowers the severity of node up events to HARMLESS.
router_lower	Lowers the severity of router up events to HARMLESS.
subnet_lower	Lowers the severity of subnet reachable events to HARMLESS.
interface_added_lower	Lowers the severity of interface added events to HARMLESS.
interface_managed_lower	Lowers the severity of interface managed events to HARMLESS.
node_added_lower	Lowers the severity of node added events to HARMLESS.
node_managed_lower	Lowers the severity of node managed events to HARMLESS.

Rule	Function
sa_status_lower	Lowers the severity of SA events with sastatus within ifUp, nodeUp and nodeResolved to HARMLESS.
l2_status_lower	Lowers the severity of Layer 2 node status up to HARMLESS.

While most of the rules set severity HARMLESS for clearing events, the router\_raise rule does a business impact escalation based on device type. Since the coders deemed the routers as more important network components, their failures were escalated to a severity of CRITICAL.

**Note:** We highly recommend that you set event severity in NetView using one of the approaches outlined in “Setting severities” on page 264 and disable the Severity Adjustment section of netview.rls. By sending the event with the desired severity, processing overhead is reduced in the IBM Tivoli Enterprise Console event server. That is several rules that must be checked for each network event are eliminated. Little processing is added to NetView using this approach. It already formats each event to send. The addition of one more slot variable in the event is negligible.

In addition, netview.rls defines a few predicates that are used in subsequent rules within the rule set, as shown in Table 6-10.

Table 6-10 netview.rls predicates

Predicate	Purpose
svc_impact_severity_escalation	Changes cause an event to fatal severity if service impact was reported by IBM Tivoli Monitoring. Otherwise set severity to critical unless it is already bumped to fatal by an earlier event. Returns the new severity in _result_sev.
higher_severity	Compares two severities and succeeds if the first is higher than the second.
severity_propagation	If the effectSeverity is higher than the causeSeverity, then it sets the severity of the cause event to the effect's severity.

### Escalation using escalate.rls

The escalation rule set contains rules that can increase the severity of events that are not handled within a specified period of time. When used along with the notification rule set, the escalation rules also trigger automatic e-mail or pager notification of event escalation.

The escalation rule set is not activated in the default rule base. To use this rule set, place it listed near the end of the rule\_sets file (following the correlation rule set, if that rule set is active) and activate it. In addition, the notification rule set (notify.rls) provides required support for e-mail notification. The notify.rls rule must be active for the escalation rules to trigger e-mail notification. See “Rule set sequencing and dependencies” on page 238 for more information about the required order for these rules.

The escalation rule set is preconfigured and ready to use. By default, this rule set is configured only to trigger e-mail or pager notification for FATAL events that require escalation. This function requires the notification rule set (notify.rls) also to be configured and active. Severities are not increased because FATAL events are already at their maximum severity. To escalate events with severities other than FATAL, customize the rule set by modifying the statements in the escalate\_parameters action of the escalate\_configure rule.

The following options are configurable:

- **Administrator name:** This is the administrator name to use when changing event severity. The default administrator name is escalate.rls and should be sufficient for most organizations. To change the administrator name, modify the statement that sets the \_escalate\_admin parameter as follows:

```
_escalate_admin = admin,
```

Here *admin* is the administrator name to use, enclosed in single quotation marks.

- **Escalation check frequency:** This is the frequency at which the escalation rules check the event cache for events that need to be escalated. The default frequency is every 60 seconds. To change this frequency, modify the statement that sets the \_escalate\_timer parameter as follows:

```
_escalate_timer = seconds,
```

Here, *seconds* is the length of time that you want to elapse between escalation checks.

- **Latency:** This indicates how far back in the event cache you want to search for events to escalate. The default is 30 days. To change the latency, modify the statement that sets the \_esc\_search\_time parameter as follows:

```
_escalate_search_time = seconds,
```

In this case, *seconds* is the number of seconds representing how far backward to search the cache for events.

- **Housekeeping frequency:** This specifies how frequently to remove references to escalated events that are no longer in the event cache. When an event is escalated, the rules assert an escalation fact in the knowledge base. The housekeeping rule periodically checks to ensure that each

escalation fact refers to an event that is still in the event cache. When an event is removed from the cache because of its age or because of space limitations, the housekeeping rule removes the associated escalation fact from the knowledge base. The default housekeeping frequency is 86400 seconds (24 hours). To change this frequency, modify the statement that sets the `_esc_housekeeping_timer` parameter as follows:

```
_esc_housekeeping_timer = seconds,
```

In this example, *seconds* is the length of time between housekeeping checks.

- **Whether to increase event severity:** Event severity may be automatically increased when an event is to be escalated. If this option is disabled (false), the escalation rules do not change event severity, but still trigger notification by the notification rules (if `notify.rls` is active). By default, this parameter is set to true. The syntax of the statement is:

```
_escalate_increase_severity = flag,
```

Here, *flag* represents either true or false.

- **Classes to escalate:** This indicates the IBM Tivoli Enterprise Console classes of events to be escalated. Code is supplied to list `TEC_Error` and `TEC_DB` events as those to be escalated. Uncomment this code and modify the `_escalate_class_list` parameter with the desired event classes:

```
rerecord(escalate_class_list,[
ev_classes
])
```

`ev_classes` is a list of event classes, each enclosed in single quotation marks, separated by commas. To apply the escalation rules to all classes, comment out this line and leave `_escalate_class_list` undefined.

- **Escalation time limits:** Amount of time events should be allowed to remain in ACK or OPEN status at each level of severity. For each level of severity at which you want escalation to take place, include the following statement:

```
assert(escalate_severity_timers(severity, open_ack_time, ack_close_time)),
```

Here *severity* is the severity level enclosed in single quotation marks. *open\_ack\_time* is the number of seconds to allow an event to remain open before escalation. And *ack\_close\_time* is the number of seconds to allow an event to remain acknowledged before escalation. A value of zero for `open_ack_time` or `ack_close_time` specifies no time limit. By default, the only severity level for which escalation time limits are defined is FATAL.

The following statement specifies that FATAL events are allowed to remain in OPEN state for 12 hours and in ACK status indefinitely. To change these time limits, modify the statement accordingly. To specify escalation for additional



severity levels, uncomment the corresponding statements and modify the time limits if necessary.

```
assert(escalate_severity_timers('FATAL', 43200, 0),
```

There are several rules within the rule set as summarized in Table 6-11.

Table 6-11 *notify.rls* rules

Rule	Purpose
escalate_configure	The settings in this rule govern the behavior of the escalation rules. When the event server initializes, it sends a TEC_Start event, which triggers this rule to load the settings into IBM Tivoli Enterprise Console's knowledge base. Customize this rule to configure the behavior of the escalation rules.
check_cache_for_escalation	<p>The check_cache_for_escalation rule runs upon receipt of the Escalate_event event, which is periodically generated by the escalate_old_events timer rule. When Escalate_event is received, the rule searches the event cache for any events that have remained in ACK or OPEN status longer than the allowed period of time. If a class list is defined using the escalate_class_list configuration parameter, this search is limited to the classes specified in that list.</p> <p>For each matching event, the rule first checks the knowledge base for a corresponding escalation fact, which indicates that the event is already escalated. If no escalation fact is found, a timer is set with a duration of one second to trigger immediate escalation by the escalate_specific_event timer rule. An escalation fact is then asserted in the knowledge base for the escalated event, and the received Escalate_event event is dropped.</p>
escalate_old_events	The escalate_old_events timer rule periodically generates Escalate_event events, which trigger the check_cache_for_escalation rule. The rule then resets the Escalate timer to trigger the next check. The duration of this timer is determined by the _escalate_timer parameter in the escalate_parameters configuration rule.

Rule	Purpose
escalate_specific_event	<p>The escalate_specific_event rule handles escalation. This rule runs upon expiration of any Escalate_open or Escalate_ack timer. This timer is set by the check_cache_for_escalation rule when an event is found that has remained in ACK or OPEN status too long. When the timer expires, the escalate_specific_event rule first checks to see if the event has been taken out of ACK or OPEN status since the timer was set. If this has happened, the rule retracts the associated escalation fact and then exits the rule. If the event is still in ACK or OPEN status, the rule takes one of the following actions:</p> <ul style="list-style-type: none"> <li>▶ If the _escalate_increase_severity parameter is set to true, the severity of the event increases, unless it is already FATAL, in which case it is reset to FATAL.</li> <li>▶ If the _escalate_increase_severity parameter is set to false, the severity of the event is reset to its current value.</li> </ul> <p>In either case, because the severity is reset, the change rules in the notification rule set are triggered, if that rule set is active.</p>
escalate_housekeeping	<p>The escalate_housekeeping rule runs upon expiration of the Escalate_housekeeping timer, which is set by the configuration rule based upon the duration specified by the _esc_housekeeping_timer parameter. When the timer expires, the escalate_housekeeping rule checks the event cache for each event for which an escalation fact exists in the knowledge base. If any escalated events are no longer in the event cache, the rule retracts the corresponding escalation facts from the knowledge base. It then resets the timer.</p>

For the purposes of our testing, we set the following configuration parameters to match our escalation policy defined in “Notification chain” on page 278. Since the policy applies to all events, we let the rule default to every event. It does not make sense to escalate HARMLESS events. If an event requires action, it should be supplied a different severity. Therefore, we left that configuration parameter unset.

- ▶ \_escalate\_increase\_severity = true
- ▶ assert(escalate\_severity\_timers('FATAL', 1800, 3600))
- ▶ assert(escalate\_severity\_timers('CRITICAL', 3600, 7200))
- ▶ assert(escalate\_severity\_timers('MINOR', 7200, 14400))
- ▶ assert(escalate\_severity\_timers('WARNING', 43200, 86400))
- ▶ assert(escalate\_severity\_timers('UNKNOWN', 86400, 172800))

By default, each severity is escalated as follows:

```
FATAL remains FATAL
CRITICAL becomes FATAL
MINOR becomes CRITICAL
```

WARNING becomes MINOR  
HARMLESS becomes WARNING  
UNKNOWN becomes HARMLESS

We commented out the escalation of HARMLESS and modified UNKNOWN to become WARNING. Example 6-31 shows the relevant lines from `escalate.rls`.

*Example 6-31 Severity escalation definitions from `escalate.rls`*

---

```
assert( next_level_of_severity('FATAL','FATAL')),  
assert( next_level_of_severity('CRITICAL','FATAL')),  
assert( next_level_of_severity('MINOR','CRITICAL')),  
assert( next_level_of_severity('WARNING','MINOR')),  
% assert( next_level_of_severity('HARMLESS','WARNING')),  
assert( next_level_of_severity('UNKNOWN','WARNING')),
```

---

Then we activated the rule set by issuing the following command:

```
wrb -imptgrule escalate -before notify EventServer genericTip006rRb
```

This places the rule toward the end of the rulebase, before the `notify`, as recommended. Then, we recompiled, reloaded, and recycled the rule base.

Next, we tested the rule by generating a **CRITICAL** event and not acknowledging it within the acknowledge time frame. As expected, the event becomes **FATAL**, as indicated by the **wtdumper** output.

We ran a second test, generating a **MINOR** event and not acknowledging it within the acknowledge time frame. As expected, it escalates to **CRITICAL** and then to **FATAL**.

In environments where the IBM Tivoli Enterprise Console is used for management, the `escalate.rls` rule works well to provide a best practices method of escalating events that are not handled. It can also be effectively used to escalate the severity of trouble tickets when the trouble ticketing interface supports updating tickets for event changes.

### ***Escalation using event thresholds.rls***

The main purpose of this rule is to perform throttling. This threshold capability of the rule is discussed in 6.2.2, “IBM Tivoli Enterprise Console duplicate detection and throttling” on page 212. A side function of the rule is to perform worsening condition escalation.

As supplied, the rule determines the number of events of the same severity received for the same host. When the number exceeds a specified threshold, it

upgrades the status of the event that passed the threshold according to Table 6-12.

*Table 6-12 Event escalation table*

Severity	Threshold	New severity	Frequency
MINOR	10 events within 60 seconds	CRITICAL	Once every 300 seconds (5 minutes)
CRITICAL	5 events within 60 seconds	FATAL	Once every 300 seconds (5 minutes)

Therefore, the eleventh event of severity MINOR within a minute is upgraded to CRITICAL, and its repeat count set to 10. This escalation takes place once during a five minute interval.

To test the rule, we wrote a script to generate a number of events of MINOR severity in succession. The **wtdumper** output (Example 6-32) shows the eleventh event. As expected, it has been upgraded to CRITICAL severity, and repeat count was incremented to 10.

*Example 6-32 The wtdumper output showing escalation based on threshold*

---

```
TEC_Error;  
    msg='Testing threshold rule';  
    msg_catalog='';  
    status=OPEN;  
    administrator='';  
    acl=[ admin];  
    severity=CRITICAL;  
    date='Oct 14 15:12:34 2003';  
    duration=0;  
    msg='Testing threshold rule';  
    msg_catalog='';  
    msg_index=0;  
    num_actions=0;  
    credibility=1;  
    repeat_count=10;  
    cause_date_reception=0;  
    cause_event_handle=0;  
END
```

---

Next, we tested by generating 13 CRITICAL events in succession. We expected the sixth event to be escalated to severity FATAL and have a repeat count of five. This worked as designed. The remaining events within the five minute time

interval were unaffected. After five minutes expired, we regenerated the thirteen events, and again, the sixth event was again the only one modified.

The `event_threshold.rls` rule set is not activate upon installation. Before activating, customize the rule to set meaningful criteria and thresholds. As supplied, the rule is an example of how thresholds and escalation can be performed using IBM Tivoli Enterprise Console rules.

To change the threshold values, modify the `create_threshold` predicate for the appropriate severity events. Example 6-33 shows the section of the rule set in which the processing values are set for CRITICAL events.

*Example 6-33 Excerpt from event\_threshold.rls: Set values for CRITICAL events*

---

```
create_threshold(critical_threshold,      % Name of threshold
                all_critical_search,      % Cache Search to use
                60,                       % Reception period (seconds)
                5,                        % Event threshold count
                300                       % Maximum reporting frequency
                ),
```

---

As with any IBM Tivoli Enterprise Console rule, keep performance in mind when modifying and using this rule. In general, scanning IBM Tivoli Enterprise Console cache for all instances of events that meet a given criteria can be processing intensive.

### ***Escalating severity using IBM Tivoli Enterprise Console rules***

There are several ways to update the severity of an event through IBM Tivoli Enterprise Console rules. A main difference in several of the methods is whether change rules are triggered by the predicate.

► Use `set_event_severity`.

This predicate sets the severity of the specified event by directly modifying the value of the severity attribute without issuing an internal change request that goes through the change rules. To trigger change rules, call the `place_change_request` predicate following the `set_event_severity` call, or use `change_event_severity` predicate. The syntax is:

```
set_event_severity(_event, new_severity)
```

Here, `_event` is a pointer to the event for which the severity is to be set and `new_severity` is the new event severity.

The following example shows predicate usage:

```
set_event_severity(_event, 'CRITICAL')
```

- Use `change_event_severity`.

This predicate places an internal request to change the severity of the specified event. This causes the change rules to evaluate the requested change before it is actually applied. The syntax is:

```
change_event_severity(_event, new_severity)
```

Here, *\_event* is a pointer to the event for which the severity is to be set and *new\_severity* is the new event severity.

The following example shows how to change the severity attribute of the event under analysis to CRITICAL:

```
change_event_severity(_event, 'CRITICAL')
```

- Use `bo_set_slotval`.

This method updates an event attribute value in the specified event with a new value. Unlike the `place_change_request` predicate, change rules are not evaluated in response to this action. Also, unlike `place_change_request`, `bo_set_slotval` does not automatically update the attribute value in the event database or the event consoles. Often these are updated automatically as part of other rule base activity, such as when the event is initially processed or during a change rule on that event. However, if you are using `bo_set_slotval` from a change rule on a different event than the current event, the update does not happen. To ensure that the attribute is updated everywhere, follow this up with a call to the `re_mark_as_modified` predicate.

The syntax is:

```
bo_set_slotval(_event, _attribute, _value)
```

Here, *\_attribute* is the attribute to update, *\_event* is a pointer to the pointer to the event to modify, and *\_value* is the new value to assign the attribute.

The following example shows how to update the severity attribute of the event under analysis to the value FATAL:

```
bo_set_slotval(_event, severity, 'FATAL')
```

- Use `place_change_request`.

This method requests a change to an attribute value. Change rules are triggered in response to the requested change. If there are no change rules in the rule base, the `bo_set_slotval` predicate is a more efficient choice to change an attribute value, because processing resources are not used to check the rule base for change rules. The syntax is:

```
place_change_request(_event, _attributename, _newattributevalue)
```

Here, *\_attributename* is the attribute to change, *\_event* is a pointer to the event containing the attribute to change, and *\_newattributevalue* is the value to assign the updated attribute.

The following example requests to change the severity attribute to a value of FATAL:

```
place_change_request(_event, severity, FATAL)
```

- ▶ Execute the Change\_Severity task supplied by IBM Tivoli Enterprise Console. The T/EC Tasks task library in the IBM Tivoli Enterprise Console Region in the Tivoli Management Region contains a Change\_Severity task.
- ▶ When correlating an escalation sequence, keep the first event, and escalate its severity, adding information to it if necessary. You must keep the first because it was reported, and event synchronization from trouble-ticketing system looks for it to close when trouble ticket is closed. It also keeps record of time failure first occurred this way.
  - An implication is that IBM Tivoli Enterprise Console event must record the trouble ticket number in a slot.
  - You must discuss ways to update one event with information from another.
- ▶ Use timers to escalate in IBM Tivoli Enterprise Console. This is bad because there is a limit on timers. We recommend that you escalate through the trouble-ticketing system.

## 6.6 Event synchronization

Event synchronization is an important component of event management, especially when working in a centralized event management world. Without event synchronization, you or your colleagues can be working on problems which are already solved, or you may miss problems that have escalated. This section discusses how the correlating IBM products handle event synchronization.

### 6.6.1 NetView and IBM Tivoli Enterprise Console

NetView and IBM Tivoli Enterprise Console handle event synchronization via rules. These rules are:

- ▶ **netview.rls**: Keeps IBM Tivoli Enterprise Console and NetView synchronized downward.

NetView.rls is the main rule that handles all communication with NetView. It mainly goes through the events that NetView sends and correlates them with events that have already passed through. It also sends traps back to NetView when an event is updated at IBM Tivoli Enterprise Console.

For example, if a router goes down and NetView picks it up, it sends a router down event to IBM Tivoli Enterprise Console. When that router comes back

up, NetView sends a router up event to IBM Tivoli Enterprise Console, which netview.rls correlates with the original router down and closes it.

If you get the router down event in IBM Tivoli Enterprise Console from NetView and then you acknowledge that event in IBM Tivoli Enterprise Console, IBM Tivoli Enterprise Console sends a trap back to NetView so that router's icon is displayed as *acknowledged* on the NetView console.

If you close that router on the IBM Tivoli Enterprise Console, IBM Tivoli Enterprise Console then sends a trap back to NetView. When NetView receives this trap, it polls the affected device to see if it is still up. However NetView does not send a new event back to IBM Tivoli Enterprise Console because of its state change nature. The main purpose of the netview.rls rule is for synchronization between IBM Tivoli Enterprise Console and the NetView console.

Upward synchronization is not as vital. NetView doesn't perform duplicate detection, so it doesn't need to update slot in IBM Tivoli Enterprise Console. NetView typically does state change monitoring, which necessitates sending a new event.

- Upward synchronization done by NetView sending an event of the same class, with status=close.

## 6.6.2 IBM Tivoli Enterprise Console gateway and IBM Tivoli Enterprise Console

IBM Tivoli Enterprise Console gateway and IBM Tivoli Enterprise Console event server event synchronization is handled automatically by the IBM Tivoli Enterprise Console product set. Event synchronization only occurs from the IBM Tivoli Enterprise Console gateway to the IBM Tivoli Enterprise Console event server. When state correlation is enabled on the IBM Tivoli Enterprise Console gateway, and the XML rule is in place, depending on how you wrote the XML rule, correlation occurs on the gateway. It is based on events that match your defined criteria and events received within a defined timing interval which match that same criteria. The events are correlated as you defined them. Then a summary event is sent to the defined IBM Tivoli Enterprise Console event server.

There is no data flow from the IBM Tivoli Enterprise Console event server to the IBM Tivoli Enterprise Console gateways. This is simply because the IBM Tivoli Enterprise Console state correlation gateways handle all event correlation for their managed event sources before sending data to the IBM Tivoli Enterprise Console event server. There is no need for downstream data from the IBM Tivoli Enterprise Console event server to the IBM Tivoli Enterprise Console gateways. Once state correlation takes place at the gateway, all subsequent events are handled independently. If you set a time interval for a specific correlation, and



that time interval is reached, it is reset and starts when the next matched event is received.

### 6.6.3 Multiple IBM Tivoli Enterprise Console servers

This section lists a few examples of when event synchronization is needed between IBM Tivoli Enterprise Console event server.

#### **CLOSED event synchronization from a low-level IBM Tivoli Enterprise Console event server**

One example of event synchronization between IBM Tivoli Enterprise Console servers is when you close an event at a lower level IBM Tivoli Enterprise Console event server. Automatic event synchronization is required to escalate this situation to the upper level IBM Tivoli Enterprise Console event server.

Synchronizing events between IBM Tivoli Enterprise Console servers can sometimes be a challenging task. You must make sure that, whenever you close an event on one server, that event is closed on all servers. This helps to avoid errors when correlating this event on all levels of your IBM Tivoli Enterprise Console servers.

IBM Tivoli Enterprise Console 3.9 comes with an out-of-the box rule that helps you forward an event from one IBM Tivoli Enterprise Console event server to another. This rule is called *forwarding.rls* and is not activated by default. A little customization of this rule is necessary for true bi-directional synchronization between your IBM Tivoli Enterprise Console servers. You also need to tell this rule what kind of events you want to forward.

You may be dealing with a hierarchy of IBM Tivoli Enterprise Console servers. In this case, a low-level IBM Tivoli Enterprise Console event server does most of your correlating. Then any duplicate detection and filtering that are not caught at the gateway are forwarded along with any remaining events to a high-level IBM Tivoli Enterprise Console event server. This high-level IBM Tivoli Enterprise Console event server is responsible for correlating all events from all sources.

Figure 6-38 shows the relationship between the event sources and the different levels of IBM Tivoli Enterprise Console servers. The lines that connect the two IBM Tivoli Enterprise Console servers, NetView and the trouble-ticketing system, are bidirectional.

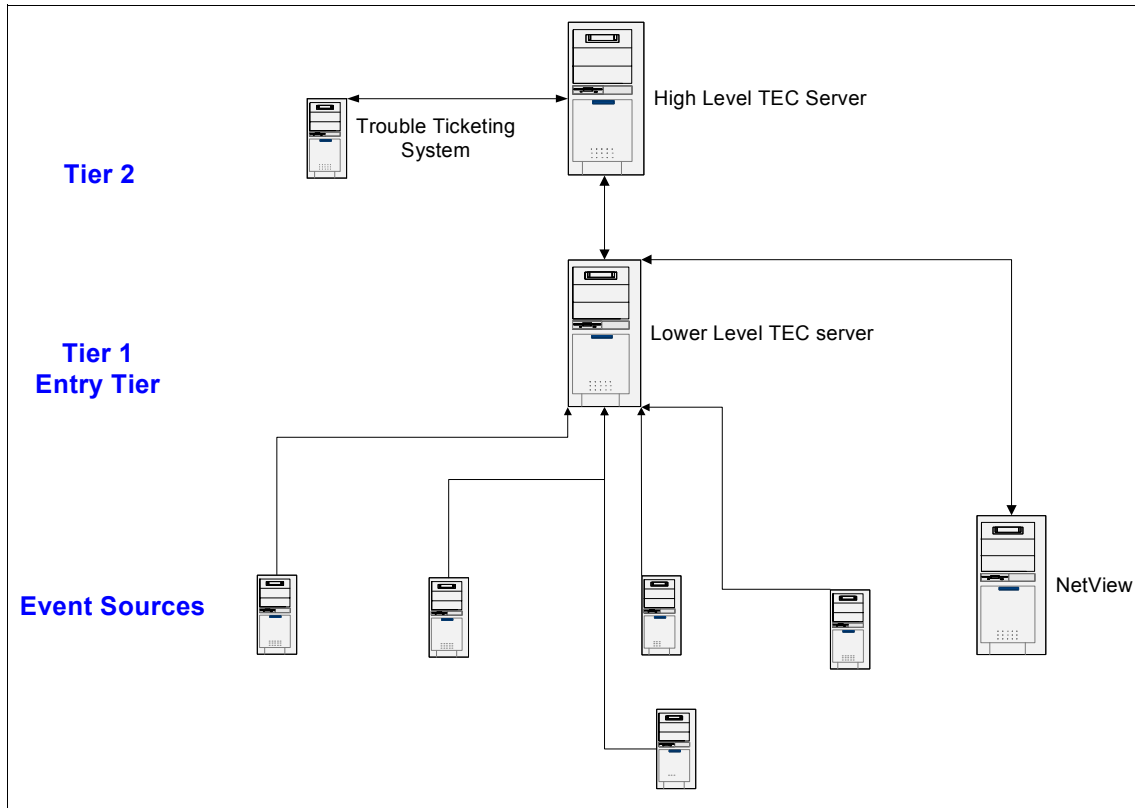


Figure 6-38 Relationship between IBM Tivoli Enterprise Console servers in a hierarchy

Now let's look at the forwarding.rls rule. The default action part of the forwarding.rls rule does not provide any rules that synchronize a closed event with other IBM Tivoli Enterprise Console servers. For our purposes, we modify this rule so that events are also forwarded to the other IBM Tivoli Enterprise Console servers when an event is closed.

Example 6-34 shows the first part of the forwarding.rls rule. This is the reception action section of the forwarding\_configure rule.

#### Example 6-34 Reception action of the forwarding.rls rule

```

create_event_criteria(ups_fatal_forwarding,      % event criteria name
                    'upsDischarged',            % class to filter on
                    yes,                        % fire on non-leaf only (yes/no)
                    [ ['severity', within, ['FATAL']] % criteria based on slots
                    ]
  
```

```

        ),
create_event_criteria(temp_alarm_forwarding, % event criteria name
                    'chassisAlarmOffTempAlarm',% class to filter on
                    yes, % fire on non-leaf only (yes/no)
                    [ ['severity', within, ['WARNING','FATAL']] ] % criteria based
on slots
                    ]
        ),

```

---

This is the section of this rule where you define the type of events to forward. You must also set the event server to where you will forward these events in the `tec_forward.conf` file that is located in the `TEC_RULES` directory. See 6.4.3, “Rules” on page 251, for more information about how you can place this information into a fact file.

There are two examples or forwarding rules by default: one for `ups_fatal_forwarding` and one for `create_event_criteria`. We use a test event class for our testing. We modify this section of the rule as outlined in Example 6-35.

---

*Example 6-35 Test event class within the forwarding.rls rule*

---

```

create_event_criteria(test_tec_event_forwarding,
                    'TEC_Test_Event',
                    yes,
                    [ ['severity', within, ['WARNING','FATAL']] ]
                    ]
        ),

```

---

We use the `TEC_Test_Event` class for this example. We forward only it if the severity is `WARNING` or `FATAL`. Example 6-36 shows the section of the rule where the actual forwarding takes place.

---

*Example 6-36 Forwarding section of the forwarding.rls rule*

---

```

rule: forwarding_events:
(
    event: _event of_class _class
    where [
    ],
    reception_action: forward_event:
    (

```

```

        % check if event matches any forwarding criteria
        recorded(event_forwarding_criteria, _criteria),
        check_event_criteria(_criteria, any, _event),

        % if found, forward it
        forward_event(_event)
    )
).

```

---

This rule works well for forwarding any event that we need. However, how do we handle a situation where we close an event and let the other event servers know that the event is closed? We do this by adding a change rule, as shown in Example 6-37.

#### *Example 6-37 Change rule for forwarding.rls*

---

```

change_rule: 'cause_closed_in_TEC':
(
    event: _event of_class _class
        where[
            ],
    attribute: status set_to 'CLOSED',
    action: forward_to_other_tec:(
        recorded(event_forwarding_criteria, _criteria),
        check_event_criteria(_criteria, any, _event),
        forward_event(_event)
    )
).

```

---

In this rule, we check any event that closes against our event criteria that we set. If an event's status changes to *Closed* and it matches the criteria, it forwards that event in a *Closed* status to the other IBM Tivoli Enterprise Console event server. After the other IBM Tivoli Enterprise Console event server receives this event, it correlates it with the original event and then closes that it.

### **Synchronizing a CLOSED event from a high-level IBM Tivoli Enterprise Console event server**

We also need to perform some customizing to ensure that any event that is closed on the higher level IBM Tivoli Enterprise Console is also closed on the lower level IBM Tivoli Enterprise Console. This is the opposite direction of what we discussed previously.

To do this, we use the forwarding.rls rule set and modify it for our purposes. We do not want to send any events other than update events to the lower level IBM Tivoli Enterprise Console from the higher one. Therefore, we add only our change rule to forwarding.rls and delete the rule for forwarding normal open events.

**Note:** Remember to change the tec\_forward.conf file to point to the lower level IBM Tivoli Enterprise Console. Also change the event criteria at the top of forwarding.rls to match for the type of events with which you are working.

Example 6-38 shows how our rule should look for adding event synchronization from a high-level IBM Tivoli Enterprise Console event server.

---

*Example 6-38 Rule for adding event synchronization*

---

```
rule: forwarding_configure:
(
  event: _event of_class 'TEC_Start'
  where [
  ],

  reception_action: forwarding_parameters:
  (
    create_event_criteria(test_tec_event_forwarding,
                        'TEC_Test_Event',
                        yes,
                        [ ['severity', within, ['WARNING','FATAL']] ]
                        ),
    record(event_forwarding_criteria, [test_tec_event_forwarding
                                     ])
  )
).

change_rule: 'cause_closed_in_TEC':
(
  event: _event of_class _class
        where[
        ],
  attribute: status set_to 'CLOSED',
  action: forward_to_other_tec:(
    recorded(event_forwarding_criteria, _criteria),
    check_event_criteria(_criteria, any, _event),
    forward_event(_event)
  )
)
```

)  
).

---

This is similar to our rule on the lower level IBM Tivoli Enterprise Console event server. The exception is that we do not have the rule to forward events. We are only concerned here with notifying the lower level IBM Tivoli Enterprise Console when an event is closed on the higher level IBM Tivoli Enterprise Console.

## 6.6.4 IBM Tivoli Enterprise Console and trouble ticketing

Another type of situation where you want to consider synchronization is for integration of an IBM Tivoli Enterprise Console and a trouble-ticketing system. IBM Tivoli Enterprise Console provides an out-of-the-box `troubleticket.rls` rule to help you do this. As with all of the out-of-the-box rules, some customization is necessary. We look at this rule and discuss the integration of this rule with Peregrine ServiceCenter.

The `troubleticket.rls` rule is provided as an example for integration with a trouble-ticketing system. It gives an example of how you can integrate your trouble-ticketing system with IBM Tivoli Enterprise Console. This is only an example.

Most trouble-ticketing systems have their own method of integration. Keep in mind that you can still use some of the rules from `troubleticket.rls` if necessary. We investigate the `troubleticket.rls` rule to show an example of how you can perform trouble-ticketing integration. We explain how to customize it to generate a log file with the information that may go to a trouble-ticketing system.

To start customizing the `troubleticket.rls`, you must specify on which events you want to open a ticket as shown in Example 6-39.

*Example 6-39 `troubleticket.rls` event class match*

---

```
% *****
% USER MUST ASSERT ALL RELEVANT CRITERIA HERE
% *****
assert_tt('TEC_Error','FATAL',_)
),
```

---

With this configuration, we sending all Fatal events with the class `TEC_Error`. We want to change this for our example to send Fatal events from the IBM Tivoli Enterprise Console class `TEC_Test_Event` from the host `rduatc01`. We modify this section as shown in Example 6-40.

*Example 6-40 troubleticket.rls showing TEC\_Test\_Event class match*

---

```
% *****
% USER MUST ASSERT ALL RELEVANT CRITERIA HERE
% *****
assert_tt('TEC_Test_Event','FATAL','rduatc02')
),
```

---

The troubleticket.rls rule calls a couple of classes that are not defined in the rule base. You must define the following classes with the values shown in Example 6-41 before the rule can work.

*Example 6-41 Class definitions for the troubleticket.rls rule*

---

```
TEC_CLASS:
  TT_Open_Event ISA EVENT
  DEFINES {
    severity: default = MINOR;
    ttserver_handle: STRING;
    ttdate_reception: STRING;
    ttevent_handle: STRING;
  };
END

TEC_CLASS:
  TT_Update_Event ISA EVENT
  DEFINES {
    severity: default = MINOR;
    ttserver_handle: STRING;
    ttdate_reception: STRING;
    ttevent_handle: STRING;
    slotvector: STRING;
  };
END

TEC_CLASS:
  TT_Close_Event ISA EVENT
  DEFINES {
    severity: default = MINOR;
    ttserver_handle: STRING;
    ttdate_reception: STRING;
    ttevent_handle: STRING;
  };
END
```

---

After we compile this rule, we run a script entitled TroubleTicket.sh based on the criteria we defined. Because we currently do not have an actual integration with a trouble-ticketing system, TroubleTicket.sh outputs to a file in /tmp. TroubleTicket.sh is located in the \$BINDIR/TME/TEC directory of your IBM Tivoli Enterprise Console event server.

After we receive an event with a matching event class, we should receive a file in /tmp. The file starts with *tt*, followed by a twelve-digit number, which is the *trouble ticket ID*, suffixed with the .log extension. The file that is generated in our environment is called *tt111065704782.log*. Example 6-42 shows an example of this file. This contents of the file are quite large because it includes all of the information that you may need to pass to a trouble-ticketing system.

*Example 6-42 Trouble ticketing log file from IBM Tivoli Enterprise Console*

---

```
Trouble Ticket ID:
tt111065704782
A trouble ticket has been opened on reception of this event from host rduatc02
The event has the following attributes:
BIM_PROLOG_DIR=/usr/local/Tivoli/bin/aix4-r1/TME/TEC
BUILD_INTERP=aix4-r1
DBDIR=/usr/local/Tivoli/db/rduatc02.db
DEST=/tmp/tt111065704782.log
DEST_DIR=/tmp
ERRNO=25
EVENT_CLASS=TEC_Test_Event
FCEDIT=/usr/bin/ed
IFS= '
'
INTERP=aix4-r1
LANG=en_US
LIBPATH=/usr/local/Tivoli/lib/aix4-r1:/usr/lib:/opt/inst/ibllib/aix4-r1:/usr/lib
LINENO=122
MAILCHECK=600
NLSPATH=/usr/local/Tivoli/msg_cat/%L/%N.cat:/usr/lib/nls/msg/%L/%N:/usr/lib/nls
/msg/%L/%N.cat:/usr/dt/nls/msg/%L/%N
.cat:/usr/dt/lib/nls/msg/%L/%N.cat:/usr/dt/lib/nls/msg/%l/%t/%c/%N.cat:/usr/dt/
lib/nls/msg/%l/%c/%N.cat:/usr/lib/nl
s/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
OPTIND=1
PATH=/usr/local/Tivoli/bin/aix4-r1/bin:/bin:/usr/bin
PPID=26564
PS2='> '
PS3='#? '
PS4='+ '
RANDOM=17231
```



```

SECONDS=0
SHELL=/usr/bin/sh
SLOTS='server_handle date_reception event_handle source sub_source origin
sub_origin hostname fqhostname adapter_ho
st date status administrator acl credibility severity msg msg_catalog msg_index
duration num_actions repeat_count c
ause_date_reception cause_event_handle server_path ttid'
TEC_BIN_DIR=/usr/local/Tivoli/bin/aix4-r1/TME/TEC
TEC_KB_DIR=tec/rb_dir
TEC_MASTER_PORT=35770
TEC_MASTER_START_TIMEOUT=300
TEC_RECV_BUFSIZE=500
TEC_RECV_LOG=YES
TEC_RULE_CACHE_CLEAN_FREQ=3600
TEC_RULE_CACHE_FULL_HISTORY=86400
TEC_RULE_CACHE_NON_CLOSED_HISTORY=15552000
TEC_RULE_CACHE_SIZE=1000
TISDIR=/usr/local/Tivoli/bin/aix4-r1/./generic
TMOUT=0
TZ=EST5EDT
WLOCALHOST=rduatc02
acl='\"[admin]\"'
adapter_host='\"'
administrator='\"'
cause_date_reception=0
cause_event_handle=0
class_name=TEC_Test_Event
credibility=0
date='\"Oct  9 09:10:57 2003\"'
date_reception=1065704782
duration=0
event_handle=1
flag=OPEN_TT
fqhostname=rduatc02
hostname=rduatc01
message='A trouble ticket has been opened on reception of this event from host
rduatc02'
msg=test22
msg_catalog='\"'
msg_index=0
num_actions=1
o_dispatch=94
origin=9.24.106.153
repeat_count=0
server_handle=1
server_path='\"[rduatc01 1 1065705057 1]\"'
severity=FATAL
source=wposte
status=OPEN

```



working on the problem stops and goes back to their normal tasks. If the problem happens again, then the person must go back and start all over again.

This can be a challenging situation, especially during off hours. You don't want to continually page someone throughout the night. It's best to have the matter investigated fully by the person while they are already looking into the problem. If the problem is truly resolved, then that person can make that determination. If the problem isn't fully resolved, then the person is already investigating it has a better chance of resolving the situation, without being paged throughout the night.

It is key to choose a trouble-ticketing system that has bi-directional communication between itself and the IBM Tivoli Enterprise Console event server. This is a best practice. If you don't have such a tool, you need to rely on manually closing tickets.

## 6.7 Trouble ticketing

This section explains how IBM products integrate with trouble-ticketing software and how to adhere to best practices in this environment.

### 6.7.1 NetView versus IBM Tivoli Enterprise Console

Both NetView and IBM Tivoli Enterprise Console have the ability to open trouble tickets, via rules. Although NetView can open tickets directly from its rules set, it is usually best to open only tickets automatically from one place.

When you use IBM Tivoli software, the one place is IBM Tivoli Enterprise Console. This is because it is easier to correlate and synchronize events between IBM Tivoli Enterprise Console and a trouble-ticketing system if all of the event are going through IBM Tivoli Enterprise Console. All of the events go through the same rule set before they cut a ticket instead of one or two different rule sets, which may not be in synchronization.

### 6.7.2 IBM Tivoli Enterprise Console

You can directly open trouble tickets from within IBM Tivoli Enterprise Console using rules. In this section, we discuss the how to use the rules that are necessary to open trouble tickets.

#### **The supplied rule set: troubleticket.rls**

In 6.6.4, "IBM Tivoli Enterprise Console and trouble ticketing" on page 302, we discuss how to customize troubleticket.rls to show an example of how to create a

ticket. In this section, we show how to use this capability further when dealing with closing and opening tickets and events.

### **Bidirectional update capability**

The trouble-ticketing system for Peregrine's Service Center has a Plus module for integration with IBM Tivoli Enterprise Console. During the installation of the Plus module, you are asked if you want to replace the TroubleTicket.sh script. Make sure that you select *yes*. The Plus module installation replaces the script, but generates a log file first in /tmp, with a script that sends the event to the Service Center.

We can still use the troubleticket.rls file that we customized in 6.6.4, "IBM Tivoli Enterprise Console and trouble ticketing" on page 302, to define the criteria of which events to send to IBM Tivoli Enterprise Console. The troubleticket.rls rule runs the TroubleTicket.sh script when that criteria is met.

After the installation of the Plus module, you must modify a few rules that are provided with it for true bidirectional trouble ticketing integration. To understand what is needed to achieve this integration, we look first at how the events are linked to each other between the Service Center and IBM Tivoli Enterprise Console. When an event meets the criteria in the troubleticket.sh script, it runs TroubleTicket.sh. After it is modified by the Plus module, the installation generates an event of the SCpmOpen class. Although it is not a best practice to generate a separate event for every ticket, this is how the Plus module for the Service Center works on its event synchronization. We can clean this up later in the rule set.

The SCpmOpen event triggers the communication between IBM Tivoli Enterprise Console and the Service Center based on maps that are stored in the SCPlus directory on the IBM Tivoli Enterprise Console event server. If we fired an event with our TEC\_Test\_Event class, which is still configured to send a trouble ticket in troubleticket.rls, we should receive an SCpmOpen event, such as the event shown in Figure 6-40.

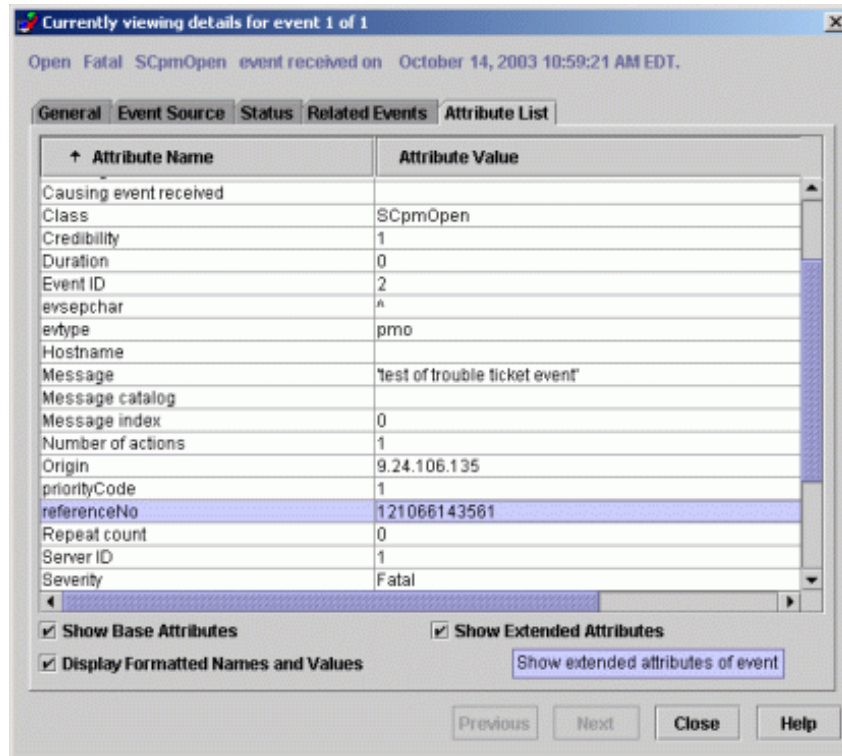


Figure 6-40 SCpmOpen event details

Notice the referenceNo slot. This number matches the referenceNo slot in our original event. We use this later to correlate back and forth between IBM Tivoli Enterprise Console and the Service Center. We must modify the TroubleTicket.sh script to ensure that the referenceNo slot from the original event is the same in SCpmOpen event. In the TroubleTicket.sh script, the following command is run:

```
wpostemsg -m "$msg" severity=$severity category=example SCpmOpen ServiceCenter
```

We must modify this command as follows that the referenceNo slot is passed on:

```
wpostemsg -m "$msg" severity=$severity referenceNo=$referenceNo
category=example SCpmOpen ServiceCenter
```

After the SCpmOpen event is sent to the Service Center, the Service Center creates a ticket and sends an event back to IBM Tivoli Enterprise Console with the SCpmOpened class. This event has the trouble-ticket number in the number slot, as shown in Figure 6-41.

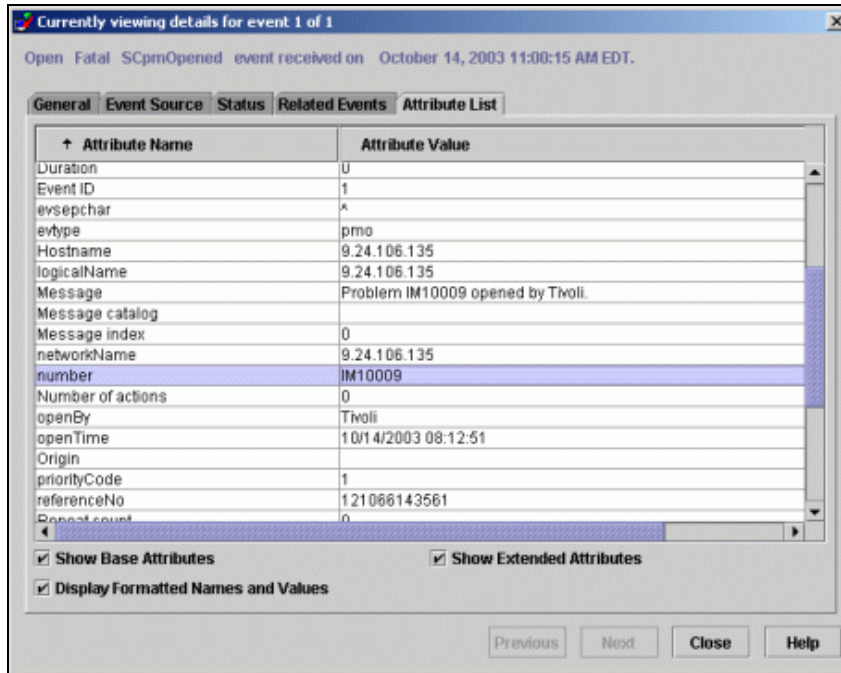


Figure 6-41 Event details showing the trouble ticket number

Notice that the referenceNo slot matches the referenceNo in our SCpmOpen event and the original event that caused it.

The default rules supplied by the Service Center Plus module are outdated and do not do a good job of correlating between the SCpm events and the originating event. Therefore, we must first add the slots number and referenceNo to the root.baroc EVENT class as a string. This ensures that every event that comes through IBM Tivoli Enterprise Console has these slots for correlation later with the Service Center.

Next we need to modify scenter.rls. This is the rule that is supplied with the Service Center Plus module for handling Service Center events. First, we must copy the scenter.rls file to scenter.rls.orig. Then delete the scenter.rls file. Create a new scenter.rls file and populate it from the following rules. We use only certain rules from scenter.rls, so doing this makes it easier to keep them straight.

Now make sure that the referenceNo slot is set for each event. We do this by adding the rule shown in Example 6-43 to our blank scenter.rls rule set.

---

*Example 6-43 Rule to populate the referenceNo slot*

---

```
rule: ensure_refno: (  
    event: _event of_class within _class  
        where [ referenceNo: equals '',  
                date_reception: _dr,  
                server_handle: _sh,  
                event_handle: _eh ],  
    reception_action: set_refno: (  
        sprintf( _refno, '%d%d%lu', [ _sh, _eh, _dr ] ),  
        bo_set_slotval( _event, 'referenceNo', _refno ),  
        commit_action  
    )  
)
```

---

We want the SCpmOpen events that are generated to create a ticket. Therefore, we add the rule outlined in Example 6-44 to run the Service Center script that sends the information to the Service Center.

---

*Example 6-44 Rule to execute the Service Center script*

---

```
rule: 'create_SC_Event': (  
  
    description:  
        'Create ServiceCenter Event from Peregrine-defined TEC Event',  
  
    event: _event of_class within [ 'SCpmOpen', 'SCtestOpen',  
                                     'SCpmUpdate', 'SCtestUpdate',  
                                     'SCpmClose', 'SCtestClose' ] ,  
  
    reception_action: run_sceventin: (  
        (exec_program( _event, '/usr/SCPlus/lib/sceventin.sh', '', [], 'YES' )),  
        drop_received_event  
    )  
)
```

---

Notice how the drop\_recieved\_event is at the end of this rule. This helps with cleanup. Now that the SCpmOpen event has created a ticket, we no longer need it, so we can drop it.

Now we want to place the Service Center trouble-ticket number that comes in the SCpmOpened event into our original event. We do this by adding the rule in Example 6-45 to scenter.rls, after the create\_SC\_event rule.

*Example 6-45 Rule to retrieve the SC trouble ticket number*

---

```
rule: 'correlate_PM_number_with_event':(
    event: _evt of_class 'SCpmOpened'
        where [ referenceNo: _refNo,
                number:      _PM_No
              ],
    action: find_orig_and_fill_PM_No:(
        /* first, find the original SC Open event */
        first_instance(
            event: _orig_evt of_class _class outside ['SCpmOpened']
            where [ referenceNo: equals _refNo,
                    status:      within [ 'OPEN', 'ACK' ]
                  ]
        ),
        /* now fill "number" slot with Problem Ticket # */
        bo_set_slotval( _orig_evt, 'number', _PM_No ),
        re_mark_as_modified(_orig_evt, _),
        drop_recieved_event
    )
).
```

---

This rule acts when the SCpmOpened event comes in and populates the number slot on the original event. It finds the original event based on the referenceNo slot. Notice how again we drop the SCpmOpened event. We already pulled the information we need from it, so we can drop it. This keeps necessary events from lingering. Now our Service Center trouble-ticket number is in our number slot, as shown in Figure 6-42.



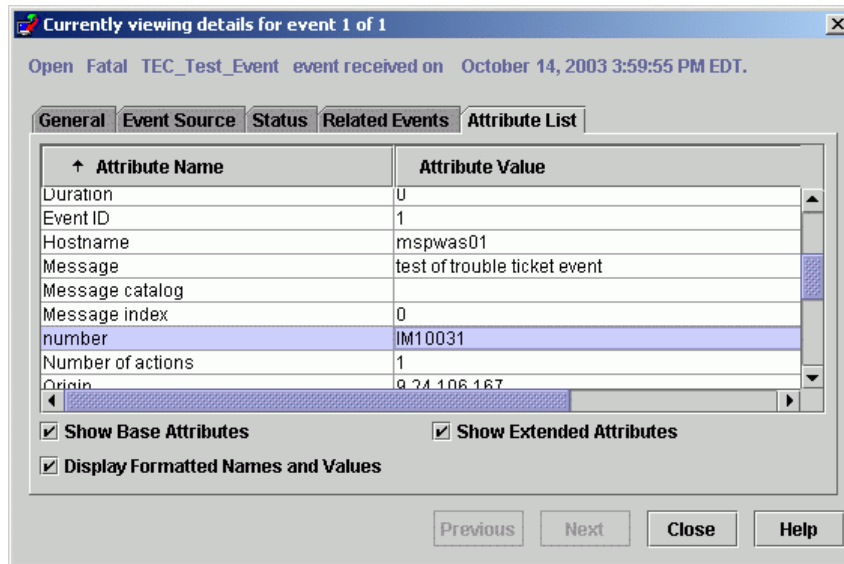


Figure 6-42 IBM Tivoli Enterprise Console event detail with ticket number in event

Now that we have our information in our event we can acknowledge an event when we receive an SCpmOpened event. We also close the IBM Tivoli Enterprise Console event when the Service Center trouble ticket is closed.

First, we add the rule shown in Example 6-46 to our scenter.rls file.

*Example 6-46 Rule to set the status of original event to acknowledges*

```
rule: sc_ack_event:
(
  event: _event of_class 'SCpmOpened'

    where [
      referenceNo: _refNo],

reception_action:
(
  all_instances(
    event: _tiv_event of_class _class outside ['SCpmOpened']
    where [status: equals 'OPEN',
      referenceNo: equals _refNo]
  ),
  set_event_status(_tiv_event,'ACK'),
  drop_received_event,
```

```

        commit_action
    )
).

```

---

This rule sets the status of the original event to acknowledged. While it is an *acknowledged* status, you may want to think about placing some duplicate detection on events that are in *acknowledged* status. This prevents multiple tickets from being opened for the same issue.

Now we can add a rule so that when an operator or analyst closes the Service Center ticket associated with an IBM Tivoli Enterprise Console event, then that event is also closed. When a ticket is closed in the Service Center, you can configure it to send an event to IBM Tivoli Enterprise Console that lets IBM Tivoli Enterprise Console know the ticket is closed. These events come into IBM Tivoli Enterprise Console with the SCpmClosed class. We can add the rule shown in Example 6-47 to close our original event.

---

*Example 6-47 Rule to close original event*

---

```

rule: close_event:
(
    event: _event of_class 'SCpmClosed'

        where [
            referenceNo: _refNo],
    reception_action:
    (
        first_instance(
            event: _tiv_event of_class _class
            where [status: equals 'ACK',
                referenceNo: equals _refNo]
        ),
        set_event_status(_tiv_event,'CLOSED'),
        drop_received_event,
        commit_action
    )
).

```

---

Once again, the drop\_recieved\_event drops the SCpmClosed events for cleanup. Now we have bidirectional integration with Peregrine ServiceCenter.

## Handling related events

You may be concerned with tracking events from the same problem that come in after you open a trouble ticket. One way to manage this is in the supplied `troubleticket.rls`.

If the flag *called* `_assoc_flag` is set to on, it calls `TroubleTicket.sh` to send only an update event and not to create a new ticket. This is for any events that are duplicates of the original event that came in.

## 6.8 Maintenance mode

There are a many things to consider when it comes to maintenance modes. This section attempts to describe how the corresponding IBM products handle this type of situation.

### 6.8.1 NetView

NetView tracks the status of the devices it manages. It generates traps when it detects changes in the condition of those devices. It also receives and processes traps sent to it by SNMP-capable devices that have NetView defined as one of their trap receivers. When discussing maintenance mode, it is important to consider both types of traps.

Another consideration is how the NetView product is used. Consider the example where all events within an organization are managed from IBM Tivoli Enterprise Console, and no-one uses the NetView console for monitoring traps. In this case, it may be sufficient to handle maintenance mode at the IBM Tivoli Enterprise Console level and allow NetView to continue processing as normal. See 6.8.2, “IBM Tivoli Enterprise Console” on page 328, for information about handling event from devices in maintenance mode at the IBM Tivoli Enterprise Console event server. Likewise, if the NetView user who monitors the console is the same person who is performing the maintenance, it may not be necessary to take any special action for the events.

We recommend that you use IBM Tivoli Enterprise Console to handle maintenance mode when possible. Allow NetView to process traps as normal. NetView and IBM Tivoli Switch Analyzer's correlation capabilities can be used to handle events generated by those products, performing root cause analysis to suppress several unnecessary events. The device in maintenance mode is most likely identified as the root cause of the problems. A trap reporting its status is forwarded to other event processors. Assuming the processors were informed that the network device is in maintenance mode through their normal means,

they properly handle both the root cause event and any unsolicited traps generated by the device itself.

An organization that does not have a higher level event processor may want to implement a method of handling traps from devices in maintenance mode at the NetView server. The same applies to organizations with users who are actively monitoring events using NetView. Several NetView features can be implemented to accomplish this.

**Unmanaging SNMP devices in maintenance mode**

The NetView object database contains global information for each object that NetView has discovered. Two of the fields it maintains for an object are OVW Maps Exists and OVW Maps Managed. These fields contain integers showing the numbers of maps on which the object is located and the number on which it is managed respectively. The information stored in the object database can be displayed using the **ovobjprint** command. In Example 6-48, the information about device sapsw01 was displayed using the command:

```
ovobjprint -s sapsw01
```

*Example 6-48 Information from NetView's object database*

OBJECTID

SELECTION NAME

OBJECT: 543

FIELD ID	FIELD NAME	FIELD VALUE
10	Selection Name	"sapsw01"
11	IP Hostname	"sapsw01"
14	OVW Maps Exists	2
15	OVW Maps Managed	2
20	IP Status	Normal (2)
23	isIPRouter	FALSE
35	vendor	cisco Systems (12)
47	isNode	TRUE
50	isConnector	TRUE
51	isBridge	TRUE
52	isRouter	FALSE
53	isHub	TRUE
54	isRepeater	TRUE
75	isIP	TRUE
94	isSNMPSupported	TRUE
96	SNMP sysDescr	"Cisco Systems Catalyst
1900,V9.00.06	"	
97	SNMP sysLocation	" "

98	SNMP sysContact	""	
99	SNMP sysObjectID	"1.3.6.1.4.1.9.5.31"	
100	SNMPAgent	Cisco Switch(372)	
106	SNMP ipAddress	"9.24.106.164"	
107	isMLM	FALSE	
108	isSYSMON	FALSE	
109	isSIA	FALSE	
110	isManager	FALSE	
112	isSLM	FALSE	
113	isSIAOS2	FALSE	
145	TopM Interface Count	1	
151	TopM Interface List	"CPU	Up
9.24.106.164	255.255.255.240 0x003080547D40	ethernet csmacd	"
233	XXMAP Protocol List	"IP"	
233	XXMAP Protocol List	"IP"	
521	IP Name	"sapsw01"	
590	default IP Symbol List	67	
		68	
695	isRMON	TRUE	
5192	jackiemap IP Symbol List		60
		61	

---

When netmon detects a status change in the network, it calls **ovtopmd**, which dispatches **ovwdb** to update fields for the appropriate managed objects in the object database. If the OVW Maps Managed field is 0 for an object, NetView is not managing it. Therefore, it does not update the IP Status field in the object database or generate any traps for the device.

Unmanaging a network device that is undergoing maintenance prevents unnecessary traps for the device from being generated by NetView and IBM Tivoli Switch Analyzer and being forwarded to other event processors. This can be accomplished by unmanaging the device on *all* maps. This does not suppress unsolicited traps generated by the device if that machine defines NetView as one of its trap receivers.

There are several ways to unmanage the device:

- Manually manage and unmanage the device through the NetView native or Web consoles.

This relies on the administrator to remember to unmanage the device at the start of the maintenance window and to re-manage it afterwards.

- Execute the **nvmaputi1** utility on the NetView machine.

This utility provides a command line method of executing some of the functions that are normally performed through the GUI. It can be used to both manage and unmanage devices using the syntax shown in Table 6-13.

Table 6-13 Syntax to manage and unmanage devices

Function	Command
Manage node	<code>nvmaputil.sh --manage-node <i>hostname or IP_address</i> --mapname <i>map_name</i></code>
Unmanage node	<code>nvmaputil.sh --manage-node <i>hostname or IP_address</i> --mapname <i>map_name</i></code>
Manage interface	<code>nvmaputil.sh --manage-interface <i>IP_address</i> --mapname <i>map_name</i></code>
Unmanage interface	<code>nvmaputil.sh --unmanage-interface <i>IP_address</i> --mapname <i>map_name</i></code>

The utility is located in the `/usr/OV/bin` directory for UNIX and in `\urs\OV\bin` for Windows. Notice that *two* dashes precede the argument names, not just one dash.

Successful execution of the commands requires the map to be opened in read/write mode. Attempts to run the utility without the map open result in the error message `There is no read/write map opened whose map name is "mapname"`.

You must run this utility on the NetView machine. You can run it manually by typing the appropriate command at a command prompt on the machine. A trap can be generated to signal the start or termination of maintenance mode for a machine. A NetView rule can respond to it automatically by executing the utility. Or the administrator can execute a Tivoli task that runs the utility, and select the NetView box as the task subscriber.

For more information about **nvmaputil**, see *Release Notes for NetView for UNIX, Version 7.1.2* on the Web at:

<http://www-1.ibm.com/support/docview.wss?uid=swg21063303>

Also see *IBM Tivoli NetView for UNIX 7.1.3 Release Notes*, GI11-0927.

This solution has several drawbacks:

- The device must be unmanaged on all maps.
  - Maps must be open in read/write mode for successful execution of the utility.
  - It does not handle unsolicited traps sent from the device.
- Update `\usr\OV\conf\offperiods.conf` (NetView for Windows only).

NetView for Windows can be configured to disable status, configuration, and discovery polling for weekly off periods. This function can be used to temporarily stop polling of devices in maintenance mode.

To schedule off periods for polling, select **Options** → **Polling**. In the Polling Options window, select **Polling Off Periods** and click the **Edit** button to edit the default file `\usr\ov\conf\offperiods.conf`. The format of this file is:

*IPaddress StartDay StartTime EndDay EndTime PollTypes*

Note the following explanation:

<b>IPAddress</b>	Specifies the IP address, expressed in numerical dot notation, of the node or nodes for which polling should be disabled. IP address ranges and a wildcard character (*) can be used to specify multiple nodes.
<b>StartDay</b>	Specifies the day of the week to start the polling off period. Specify the day by the first three letters, such as Sun, Mon, Tue, and so on. This field is not case-sensitive.
<b>StartTime</b>	Specifies the time of day to start the polling off period. Specify the time in 24-hour time (colon separated).
<b>EndDay</b>	Specifies the day of the week to end the off period. Specify the day as described for StartDay.
<b>EndTime</b>	Specifies the time of day to end the polling off period. Specify the time as described for StartTime.
<b>PollTypes</b>	Specifies the poll types to disable. Enter one or more of the values p, c, or d to specify ping, configuration, and discovery, respectively.

For example, add entries in the `\usr\ov\conf\offperiods.conf` file similarly to what is shown here. The following entry turns off all polling for one hour maintenance window for device 16.21.14.140:

```
16.21.144.140 Fri 23:00 Fri 23:59 pcd
```

The following entry disables all polling from 11 p.m. Friday to 6 a.m. Monday for every device in the IP address range

```
16.21.[140-150].* Fri 23:00 Mon 6:00 pcd
```

After editing the `\usr\ov\conf\offperiods.conf` file, in the Polling Options window, click **Apply** or **OK**. The changes take effect immediately without stopping the daemons.

Using this method to handle traps from devices in maintenance mode is easy to implement. The administrator can run a command or BAT file to append the necessary entry to the `offperiods.conf` file at the start of maintenance. The administrator can run another command to remove it afterwards. This solution, unlike the `nvmaputil` method described earlier, does not require open maps for execution. However, it also cannot handle unsolicited traps sent from the device.

Therefore, as stated earlier, we recommend that you use IBM Tivoli Enterprise Console to handle maintenance mode for networking devices. Organizations that still want to unmanage the network devices being maintained should automate the process as much as possible. Administrators may not remember to

unmanage or re-manage network devices, particularly after off-hours maintenance windows.

## **Handling maintenance mode through NetView rules**

If the NetView console is routinely monitored for network problems, an organization may want to account for maintenance mode in NetView. There are a few ways to handle the traps from devices undergoing maintenance:

- ▶ Block the traps. This prevents the traps from displaying on the console or from being processed by applications that are registered to receive them. The Block event display ruleset node can be used in a NetView rule to block the traps. See 6.1.1, “Filtering and forwarding with NetView” on page 174, for more information about this ruleset node.
- ▶ Flag the traps to indicate that they were received during the device’s maintenance mode. NetView does not provide the capability to redo traps through its rules. Therefore, it is not possible to unflag the traps after maintenance mode ends. The Override ruleset node in a NetView rule can change the status or severity of the traps to flag them as from a device in maintenance mode.
- ▶ Hide the traps from view. A filter can be used with the NetView event console to prevent the traps from displaying.

The three approaches all rely on properly identifying the trap as referencing a device in maintenance mode. Because NetView is a versatile product, you can choose from dozens of methods to make this determination. Consider maintainability and performance when designing and implementing a maintenance mode solution.

In keeping with the best practices discussed in 2.10.2, “Handling events from a system in maintenance mode” on page 74, we decided for our case study to flag the traps rather than to discard them. This ensures that, if there is a legitimate problem with the device being maintained, it is reported. The console user can optionally choose to filter the traps from the event display, if desired.

These are the steps we used to implement our solution.

### ***Added a field to the NetView database***

The NetView object database is implemented as a stand-alone module that works in conjunction with the rest of NetView. Entries in the NetView object database persist across NetView sessions. Fields and objects created in one NetView session are available to all other applications in all NetView sessions.

Information about which devices are in maintenance mode is needed by all users, applications, and sessions of NetView. Therefore, we chose to record this



information in the NetView object database. This also provides an easy method of creating a NetView smartset for objects in maintenance mode.

Such fields as `corrstat1` can be used for this purpose. However, we decided to define a new field for maintenance mode in case the `corrstat` fields were already used by other NetView rules.

To separate our customization from the one supplied with NetView and third-party applications, we created a separate file, `/usr/OV/fields/C/lab_fields`, to define the field. Example 6-49 shows the contents of the file.

---

*Example 6-49 Contents of the lab\_fields file*

---

```
/* ****
* Fields added for case study
* **** */

Field "Object_mode" {
    Type    StringType;
    Flags   Locate, General;
}
```

---

In our example, we specified `Locate` and `General` as flags. The `Locate` flag causes the field name to appear in the window that opens when you select `Locate → By Attribute` when users attempt to locate an object.

Use care in setting this field. If you do not specify the `Locate` flag, users cannot locate an object based on your field. Indiscriminate use of the `locate` field results in an overabundance of `locate` entries. This makes the window more difficult to use effectively. Set this flag only if users need to locate objects through this field. This flag cannot be set together with the `list` flag.

Fields with the `General` flag appear in a special `General Attributes` window associated with every object. This is for fields that are not application-specific and do not appear in any application-specific window. The `vendor` field is a good example of a general field.

Then we added the field to the database by issuing the following command:

```
ovw -fields
```

This reads the files in `/usr/OV/fields/C`, and either adds the fields defined in the files to the object database or verifies that they already exist. The output of the command is lengthy. A portion of it is listed in Example 6-50. The last line notes the addition of our field to the database.

*Example 6-50 Portion of output from the ovr -fields command*

---

```
/usr/ov/fields/C/xxmap_fields: Verified Enumeration field "XXMAP Layout
Algorithm"
    Verified enumeration value "None" (1)
    Verified enumeration value "User Defined" (2)
    Verified enumeration value "Point to Point" (3)
    Verified enumeration value "Bus" (4)
    Verified enumeration value "Star" (5)
    Verified enumeration value "Spoked Ring" (6)
    Verified enumeration value "Row Column" (7)
    Verified enumeration value "Point to Point Ring" (8)
    Verified enumeration value "Tree" (9)
/usr/ov/fields/C/xxmap_fields: Verified Integer field "XXMAP Flush Trigger"
/usr/ov/fields/C/xxmap_fields: Verified Integer field "XXMAP Flush Timeout"
/usr/ov/fields/C/zos_fields: Verified Boolean field "isZOS"
/usr/ov/fields/C/zos_fields: Verified String field "MVS SystemName"
/usr/ov/fields/C/zos_fields: Verified String field "MVS SysplexName"
/usr/ov/fields/C/zos_fields: Verified String field "MVS TcpiProcName"
/usr/ov/fields/C/wteuiap.fields: Verified String field "Software Status"
/usr/ov/fields/C/wteuiap.fields: Verified String field "wttest_field"
/usr/ov/fields/C/wteuiap.fields: Verified String field "WTMergeId"
/usr/ov/fields/C/wteuiap.fields: Verified Integer field "WTint"
/usr/ov/fields/C/wteuiap.fields: Verified String field "WTstring"
/usr/ov/fields/C/lab_fields: Created String field "Object_mode"
```

---

A field cannot be used until it is associated with an object. We used the **nvdimport** command to add the fields to all nodes in the database. The input file used associates the Normal object\_mode attribute to each node (see Example 6-51). The first line of the file lists the field names to which the data applies. The first field is the selection name, and the second is Object\_mode. Subsequent lines contain node names followed by the Normal mode. The data on each line is comma delimited. There are no spaces around the data.

*Example 6-51 The fields.data file used to associate object mode with object*

---

```
Selection Name,Object_mode
dtmaas01.dtm.ibmitso.com,Normal
dtmsw01.dtm.ibmitso.com,Normal
dtmsw02.dtm.ibmitso.com,Normal
dtmwas01.dtm.ibmitso.com,Normal
mspsw01.msp.ibmitso.com,Normal
mspas01.msp.ibmitso.com,Normal
phisw01.phi.ibmitso.com,Normal
phiwas01.phi.ibmitso.com,Normal
phiwas02.phi.ibmitso.com,Normal
rduanw01.rdu.ibmitso.com,Normal
```

rduarm01.rdu.ibmitso.com,Normal  
rduatc01.rdu.ibmitso.com,Normal  
rduatc02.rdu.ibmitso.com,Normal  
rduatf01.rdu.ibmitso.com,Normal  
rdur01.rdu.ibmitso.com,Normal  
rdur02.rdu.ibmitso.com,Normal  
rdusw01.rdu.ibmitso.com,Normal  
rdusw02.rdu.ibmitso.com,Normal  
rduwws01.rdu.ibmitso.com,Normal  
sapsw01.sap.ibmitso.com,Normal  
sapwas02.sap.ibmitso.com,Normal

---

The fields were added to the database using this command:

`/usr/OV/bin/nvdbimport -f /usr/OV/custom/scripts/fields.data`

A subsequent listing of the object shows the new field and its Normal value. It is the third field from the bottom in Example 6-52.

*Example 6-52 The ovobjprint output showing the new Object\_mode field*

---

OBJECTID	SELECTION NAME		
OBJECT: 289			
	FIELD ID	FIELD NAME	FIELD VALUE
	10	Selection Name	
		"dtmwas01.dtm.ibmitso.co	
		m"	
	11	IP Hostname	
		"dtmwas01.dtm.ibmitso.co	
		m"	
	14	OVW Maps Exists	1
	15	OVW Maps Managed	1
	20	IP Status	Critical(4)
	23	isIPRouter	FALSE
	35	vendor	Microsoft(22)
	47	isNode	TRUE
	49	isComputer	TRUE
	50	isConnector	FALSE
	51	isBridge	FALSE
	52	isRouter	FALSE
	53	isHub	FALSE
	56	isPC	TRUE
	75	isIP	TRUE
	94	isSNMPSupported	TRUE

95	isSNMPProxied	FALSE
96	SNMP sysDescr	"Hardware: x86 Family 6 Model 8 Stepping 3 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.0 (Build 2195 Uniprocessor Free)"
97	SNMP sysLocation	" "
98	SNMP sysContact	" "
99	SNMP sysObjectID	"1.3.6.1.4.1.311.1.1.3.1.2"
100	SNMPAgent	Microsoft Windows NT 4.0 (321)
106	SNMP ipAddress	"9.24.106.185"
107	isMLM	FALSE
108	isSYSMON	FALSE
109	isSIA	FALSE
110	isManager	FALSE
112	isSLM	FALSE
113	isSIAOS2	FALSE
145	TopM Interface Count	1
151	TopM Interface List	"IBM 10/100 Down"
9		
.24.106.185	255.255.255.240 0x0004AC98D22B ethernet csmacd	" "
233	XXMAP Protocol List	"IP"
266	Object_mode	"Normal"
292	IP Name	
"dtmwas01.dtm.ibmitso.com"		
378	default IP Symbol List	47

---

We need a method to change the field when a node goes into maintenance mode. We wrote a script to run from the NetView machine to do this. The script (Example 6-53) accepts two parameters. The first tells whether to start or stop maintenance. The second supplies the object on which to perform maintenance. You can run this script from the command line on the NetView machine. Or you can modify and set it up to execute from the object menu on the NetView map.

*Example 6-53 Script to change Object\_mode to Maintenance or Normal*

---

```
#!/bin/ksh
#
# This script sets the Object_mode for a node to Maintenance or Normal
# depending upon whether the object is being put into or removed from
# maintenance mode
#
# Syntax: /usr/OV/custom/scripts/maintmode.sh start <nodename>
#         /usr/OV/custom/scripts/maintmode.sh stop <nodename>
#
```

```

# Checks for correct number of parameters
#
if (( $# != 2 ));
then
    echo "Syntax: /usr/OV/custom/scripts/maintmode.sh start|stop <nodename>"
    exit
fi
#
# Creates a temporary file, which is used as input to the nvdbimport command
#
echo "Selection Name,Object_mode" > /tmp/$$file
case $1 in
start) echo $2,Maintenance >> /tmp/$$file;;
stop) echo $2,Normal >> /tmp/$$file;;
*) echo "Syntax: /usr/OV/custom/scripts/maintmode.sh start|stop <nodename>";;
esac
#
# Runs the nvdbimport command to change the mode of the object
#
/usr/OV/bin/nvdbimport -f /tmp/$$file
rm /tmp/$$file
exit

```

---

Next, we coded a rule set to check the Object\_mode field for all traps received. If the object is in maintenance mode, the trap is set to an indeterminate severity. To perform this function, the Query Database Field node was used in the rule set. The ruleset name was activated for a user's dynamic event workspace. The rule set is shown in Figure 6-43.

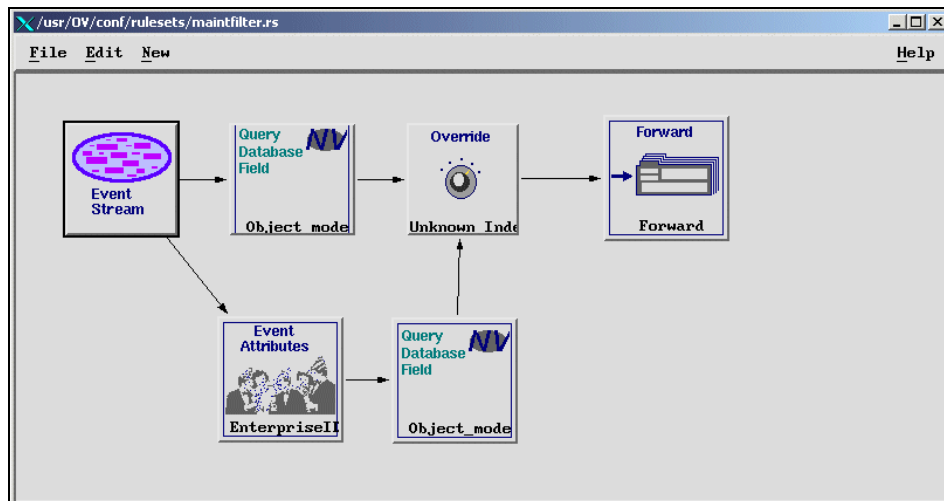


Figure 6-43 Rule set to change traps to indeterminate for objects in maintenance mode

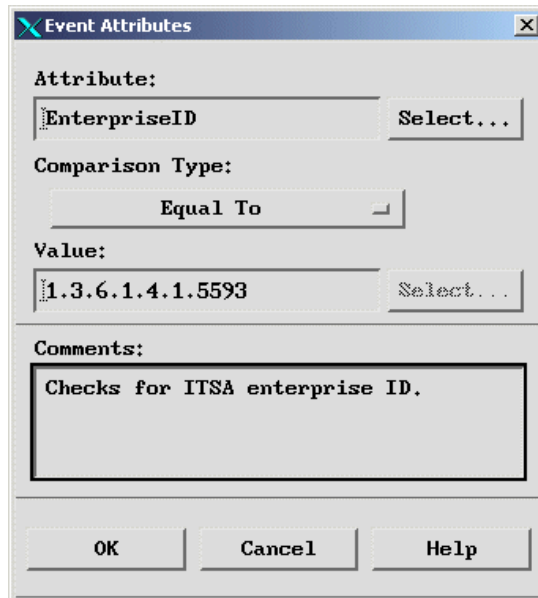
Query Database Field compares the Object\_mode field from the object database to fields in the trap. The top comparison checks against NVATTR\_2 in the trap, which is where NetView populates the host name. The bottom Query Database Field node checks NVATTR\_7 in which IBM Tivoli Switch Analyzer populates the host name. A sample setup for the node is shown in Figure 6-44.

The screenshot shows a dialog box titled "Query Database Field". It contains the following fields and controls:

- Field Name:** A text box containing "Object\_mode" and a "Select..." button.
- Object ID Source:** A text box containing "2" and a "Select..." button.
- Comparison Type:** A dropdown menu set to "Equal To".
- Compare Field To:** A section with two options:
  - Literal Value:** A text box containing "Maintenance".
  - Event Attribute Value:** A text box and a "Select..." button.
- Comments:** A large text area.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

Figure 6-44 Query Database Field node settings

The Event Attributes node checks for IBM Tivoli Switch Analyzer's enterprise ID, shown in Figure 6-45.



The 'Event Attributes' dialog box is shown with the following settings:

- Attribute:** EnterpriseID (with a 'Select...' button)
- Comparison Type:** Equal To (dropdown menu)
- Value:** 1.3.6.1.4.1.5593 (with a 'Select...' button)
- Comments:** Checks for ITSA enterprise ID. (text area)
- Buttons:** OK, Cancel, Help

Figure 6-45 Event Attributes node settings

Finally, as shown in Figure 6-46, the Override node sets the trap severity to indeterminate.



The 'Override' dialog box is shown with the following settings:

- Status:** Unknown (dropdown menu)
- Severity:** Indeterminate (dropdown menu)
- Comments:** Sets severity of trap to indeterminate. (text area)
- Buttons:** OK, Cancel, Help

Figure 6-46 Override node settings

The filter is activated when creating a dynamic event workspace by selecting the ruleset name from the supplied list as shown in Figure 6-47.

The screenshot shows a dialog box titled "Query Database Field". It contains several input fields and buttons. The "Field Name:" field has "Object\_mode" entered. The "Object ID Source:" field has "2" entered. The "Comparison Type:" is set to "Equal To". Under the "Compare Field To:" section, the "Literal Value:" is "Maintenance" and the "Event Attribute Value:" is empty. The "Comments:" field is empty. At the bottom, there are "OK", "Cancel", and "Help" buttons. The "OK" button is highlighted with a mouse cursor.

Figure 6-47 Setting the Query Database node to check for maintenance mode

## 6.8.2 IBM Tivoli Enterprise Console

Rules used for handling events for systems in maintenance mode are supplied with the IBM Tivoli Enterprise Console product. In addition, the IBM Tivoli Enterprise Console administrator can write custom rules to perform similar processing.

This section describes the supplied maintenance rules in detail. It explains how to initiate maintenance, customize the rules, and modify them to meet the needs of the organization. A custom rule for long-term maintenance is presented to handle problems whose resolution cannot be immediately implemented.



## Overview of maintenance\_mode.rls

The maintenance mode rule set provides automated event processing to indicate that a monitored system is being placed into maintenance mode for a specified period of time. As supplied, the rule can be used to discard or close events that occur from systems in maintenance mode.

### Configuring the rules

Upon installation of IBM Tivoli Enterprise Console, the maintenance\_mode.rls rule set is loaded into the current rule base and activated. As described in “Rule set sequencing and dependencies” on page 238, the maintenance\_mode rule set is placed near the beginning of the rule\_sets file to avoid unnecessary processing of events sent from systems in maintenance mode.

It is preconfigured with parameters that govern its execution, including automatically closing events received for systems in maintenance mode. These settings may be changed as described later. The rule base must be recompiled and reloaded for the changes to take effect.

To customize this rule set, modify the appropriate statements in the maintenance\_mode\_configure configuration rule. The following options are configurable:

- **Latency:** Time period in seconds to keep maintenance facts in the knowledge base after the corresponding maintenance window has ended.

This parameter allows for processing of events that were sent during a maintenance window but arrive late. The default latency is one hour. To change this interval, modify the statement that sets the `_over_time` variable as follows:

```
_over_time = otime
```

`otime` is the length of time (in seconds) that you want to keep maintenance facts in the knowledge base after maintenance has ended.

- **Maintenance event severity:** Severity assigned to TEC\_Maintenance events generated by the maintenance mode rules.

These events are generated when a maintenance window has ended. The default severity is HARMLESS. To change the severity of generated TEC\_Maintenance events, modify the statement that sets the `_severity` variable as follows:

```
_severity = msev
```

`msev` is a valid severity for the TEC\_Maintenance event class.

- **Administrator name:** Name to use when automatically acknowledging or closing received TEC\_Maintenance events.

The default administrator name is `maintenance_mode.rls`. To change the administrator name, modify the statement that sets the `_maint_admin` variable as follows:

```
_maint_admin = admin
```

*admin* is the administrator name to use.

- **Fact file name:** Name of the file to use to store maintenance facts in the knowledge base.

Specify either an absolute location for the file or the file name only to create the file in the `$DBDIR` directory. The default file name is `maintenance_mode.pro`. Its default location is `$DBDIR` on the IBM Tivoli Enterprise Console event server. To change the file name, modify the statement that sets the `_maintenance_file` variable as follows:

```
_maintenance_file = filename
```

*filename* is the name of the fact file that you want to use. It optionally includes a relative or absolute path and is enclosed in single quotation marks.

The fact file contains such entries as:

```
maintenance(rdur01,ON,0x3f8b00bc,3600)
```

This states that maintenance mode is “ON” for system `rdur01`, for a duration of 3600 seconds, or 60 minutes. The start time for maintenance mode is recorded in the hex form of the epoch time. Hex `3f8b00bc` is decimal 1066074300, which is the epoch time for 13 October 2003 at 3:45 p.m.

- **Event handling:** Specifies whether to close or drop events received from a system in maintenance mode.

The default action is to close them. To change this behavior, modify the statement that sets the `_maint_action` variable as follows:

```
_maint_action = action
```

*action* is either `CLOSE` or `DROP`.

### ***Initiating maintenance mode***

The initiation of maintenance mode is indicated by an event of the `TEC_Maintenance` class whose `current_mode` attribute is equal to `ON`. The product offers two methods to generate this event:

- Execute the `$BINDIR/TME/TEC/scripts/wstartmaint.sh` script.

**Note:** This script should be executed on the IBM Tivoli Enterprise Console event server. In our testing, attempts to run the script on other hosts failed.

The syntax of this command is:

```
wstartmaint.sh host duration "owner info" [ start time ]
```

Note the following explanation:

**host**            The fully qualified host name  
**duration**       The length of the maintenance window in minutes  
**owner info**     Information about the person who started the maintenance window  
**start time**     The maintenance start time as YYYY MM DD HH MIN SS

Here's an example of the command:

```
./wstartmaint.sh dtmwas01 15 "Jackie" 2003 10 13 13 30 00
```

Notice that there is a space between each element in the start time field.

Specifying the start time is optional. See “Processing logic” on page 333 for an explanation about how start time is used.

- Use the Start\_Maintenance task. The supplied task is located in IBM Tivoli Enterprise Console Region →T/EC Tasks on the Tivoli desktop.

**Note:** You should run this task on the IBM Tivoli Enterprise Console managed node. In our testing, attempts to run it on the IBM Tivoli Enterprise Console endpoint or on other endpoints or managed nodes failed.

When the task is run by an administrator with the proper authority (super, senior, administrator, or user), a window is displayed to enter the maintenance mode information. Enter the appropriate values:

- a. Choose the event server from the supplied list.
- b. Enter the fully qualified host name of the machine to place in maintenance mode.
- c. Supply text to indicate the administrator handling the maintenance.
- d. Enter the duration of the maintenance.

- e. If the maintenance mode should begin immediately, leave the Time to Start Maintenance field blank. Otherwise, specify the start time in the format shown at the field prompt, as shown in Figure 6-48.

Start\_Maintenance

Configure Task Arguments

Configure Start Maintenance Mode on a Host from T/EC Tasks

Event Server's Name Choose ... EventServer

hostname dtmwas01.itso.ral.ibm.com

Maintenance Owner Information Unlucky Administrator

Maintenance Duration (Minutes) 30

Time To Start Maintenance (yyyy mm dd hh mm ss) 2003 10 13 23 59 00

Set & Execute Save ... Cancel Task Description ...

Figure 6-48 Starting maintenance mode with supplied IBM Tivoli Enterprise Console task

Upon successful execution of the task, you see a window similar to the one shown in Figure 6-49.

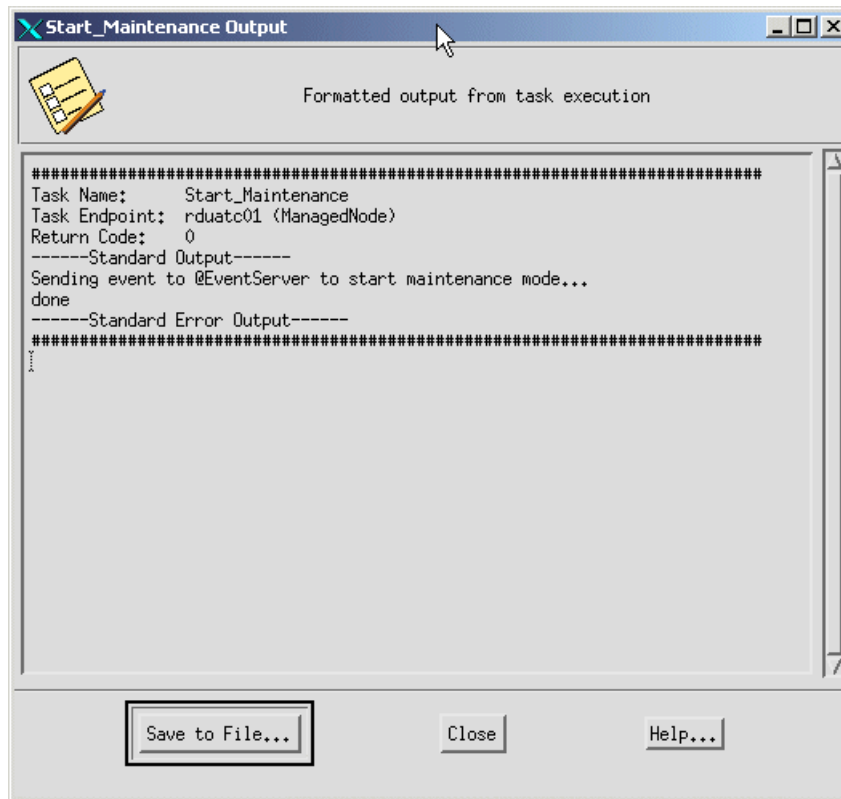


Figure 6-49 Output from IBM Tivoli Enterprise Console Start\_Maintenance task

### Processing logic

When the start maintenance event arrives, the rules record a maintenance fact in the knowledge base. This fact records the following information:

- ▶ The fully qualified host name of the system being placed in maintenance mode  
If the fqhostname attribute of the TEC\_Maintenance event is equal to a single asterisk (\*), this indicates that all monitored systems (except the event server) are being placed in maintenance mode.
- ▶ The time maintenance started or is scheduled to start  
If no start time is specified, the current time is assumed.
- ▶ The maximum allowed duration of the maintenance window (the period of time during which the system is in maintenance mode)

If the start time for the maintenance window is the current time (or a time already in the past), this indicates that the specified system is being placed in maintenance mode immediately. If the start time is in the future, this indicates that the system is scheduled for maintenance at some time in the future. In either case, the msg attribute of the generated TEC\_Maintenance event indicates that a maintenance window has begun or has been scheduled.

During the maintenance window, any events received from the system (other than TEC\_Maintenance events) are ignored. These events are either closed or dropped, depending on how the rule set is configured. When the maximum allowed duration of the maintenance window has elapsed (indicated by the maintenance timer), the monitored system is no longer considered in maintenance mode. Then a message is sent to the console indicating that the maintenance window has ended.

**Important:** The proper functioning of maintenance\_mode.rls relies on the fqhostname slot variable in an event that contains the correct fully-qualified host name. You must either ensure that events are sent with this slot variable populated or write an IBM Tivoli Enterprise Console rule to populate the value. IBM Tivoli Monitoring V5.1.1 Fix Pack 5 will use this variable. Be sure to install it when it becomes available. NetView automatically populates fqhostname if it knows devices by their fully-qualified host names.

### ***Terminating maintenance mode***

Maintenance mode is terminated when a TEC\_Maintenance event is received with current\_mode equal to OFF. Reception of this event informs IBM Tivoli Enterprise Console that either a system has finished maintenance or a scheduled maintenance window has been canceled.

There are several ways in which to generate this event:

- ▶ When the maximum duration of a maintenance window has elapsed
- ▶ Using the wstopmaint.sh script about the monitored system

Specific handling of this event depends upon the value of the start\_time attribute:

- ▶ If the start time matches a maintenance window that is currently in progress, the maintenance window is immediately ended.
- ▶ If the start time matches a maintenance window that has not yet started, the future maintenance window is canceled.
- ▶ If the start time is not specified, all current and future maintenance windows for the system are canceled.

After a maintenance window ends (or is canceled), the relevant maintenance fact remains in the knowledge base for a configurable period of time to allow for

processing of events that arrive late. After this time period elapses, any maintenance fact related to a maintenance window that has ended is retracted from the knowledge base.

### **Rules**

The maintenance\_mode.rls rule set contains several rules. Table 6-14 outlines the functions of these rules.

*Table 6-14 Rules contained in the maintenance\_mode.rls rule set*

Type	Rule	Purpose
Startup rule	maintenance_mode_configure	The maintenance_mode_configure rule is a configuration rule that runs upon receipt of the TEC_Start event, which is sent during initialization of the event server. By customizing this rule, you can configure the behavior of the maintenance_mode.rls rule set. In addition to setting global parameters for the maintenance mode rules, the maintenance_mode_configure rule restarts the maintenance timers for any pending or ongoing maintenance windows that were interrupted by a restart of the event server.
Maintenance rules	maintenance_received	<p>The maintenance_received rule manages scheduled maintenance windows for monitored systems. When a TEC_Maintenance event is received with the attribute current_mode set to ON, this rule records a maintenance fact that specifies the start time and duration for the maintenance window. If the specified duration is 0, this rule generates an error message and closes the received TEC_Maintenance event without taking any additional action. If the start time for the maintenance window is in the future, this rule starts a timer that expires when it is time for the maintenance window to start.</p> <p>When a TEC_Maintenance event is received with current_mode set to OFF, this rule searches the event cache for a matching TEC_Maintenance event specifying the same system and the same start time. Any matching event is closed. Any maintenance window currently in progress for the system is canceled. If no start time is specified by the received event TEC_Maintenance, all current and future maintenance windows for the specified system are canceled.</p>

Type	Rule	Purpose
Maintenance rules	check_maintenance_mode	<p>The check_maintenance_mode rule runs upon receipt of an event. It checks to see if the system that generated the event is currently in maintenance mode. That is, the start time of a maintenance window has passed, but the duration of the maintenance window is not yet elapsed. If the system is in maintenance mode, the received event is closed or dropped, depending upon the configuration, without further action.</p> <p><b>Note:</b> This rule does not drop or close events from the event server. The event server cannot be placed in maintenance mode.</p>
Timer rules	check_maintenance_timeout	<p>The check_maintenance_timeout timer rule checks periodically to determine whether any system has been in maintenance mode longer than the maximum allowed period of time for the maintenance window. If the maximum duration of the maintenance window has elapsed and the system is still in maintenance mode, this rule ends the maintenance window and generates a message indicating that this happened. In addition, it generates a TEC_Maintenance event with current_mode set to OFF.</p>
Timer rules	start_maintenance_timer	<p>The start_maintenance_timer timer rule sees if the start time for any scheduled maintenance window has arrived. This rule is triggered by the expiration of a maintenance timer set by the maintenance_received rule. This timer is set upon receipt of a TEC_Maintenance event specifying a maintenance window in the future. If any system is scheduled to start a maintenance window, the start_maintenance_timer rule generates a message that the specified system has entered maintenance mode. It starts a timer that expires when the maximum duration of the maintenance window has elapsed.</p>



Type	Rule	Purpose
Timer rules	check_overtime_timer	The check_overtime_timer timer rule sees if it is time to retract any maintenance facts from the knowledge base. Maintenance facts are kept in the knowledge base for a configurable period of time after the maintenance window ends to allow for handling of related events that arrive late. This interval is determined by the _over_time parameter in the maintenance_mode_configure configuration rule. When the specified amount of time has elapsed since the end of the maintenance window, the check_overtime_timer rule retracts the relevant maintenance fact from the knowledge base.

### ***Open Maintenance event group***

IBM Tivoli Enterprise Console V3.9 comes with predefined event groups. One is *Open Maintenance*. Events in this group are used to inform an operator that a specific system is in maintenance mode. When a system is in maintenance mode, events from that system cannot be processed.

Events in this event group can be in open state or acknowledged state. Events in open state indicate a scheduled maintenance time for the system identified in the event. Events in acknowledged state indicate that the system identified in the event is within the scheduled maintenance time.

### ***Testing maintenance\_mode.rls***

Several test cases were used in the lab to test the maintenance\_mode.rls rule set. The expected and obtained results were compared.

### ***Test cases***

To test the functionality of maintenance\_mode.rls in the lab, we developed a series of test cases and documented the expected outcome. Areas we focused on included updates to the knowledge base, generation and correlation of maintenance events, and dropping or closing of events from a machine in maintenance mode. Table 6-15 summarizes the results of the test.

*Table 6-15 Test case results*

Test	Expected	Actual
Generate maintenance mode event starting immediately.	Update to knowledge base	Updated knowledge base
Send an event from same host name as the maintenance mode event	Event to be closed	Event was closed

Test	Expected	Actual
Generate the maintenance mode stop event	Knowledge base update and previous host name in maintenance mode taken out	Knowledge base was updated and host name was taken out
Send an event from the same host name after the maintenance mode stop event	Event would appear in open status as normal	Event appeared as normal
Generate the maintenance mode start event for five minutes in the future	Update to knowledge base	Knowledge base updated
Send an event before the five minutes specified	Event would appear in open status as normal	Event appeared as normal
Send an event after the five minutes specified	Event to be in closed status	Event appeared in a closed status
Generate a maintenance start event for a five minute duration	Update to knowledge base	Knowledge base was updated
Send an event after the five minute duration	Event would appear in open status as normal	Event appeared as normal

### ***Best practices customization***

To meet best practices, modify the supplied maintenance rule so that events received from devices in maintenance mode are placed in an acknowledged or user status, rather than closed or dropped. When the device comes out of maintenance, either by posting an event or by the duration expiring, reprocess the events and report any outstanding events as problems.

## **6.9 Automation**

As discussed in 2.11.2, “Automation implementation considerations” on page 80, you must keep in mind several things when selecting the best tool for executing an automated action. This section describes the features of various Tivoli products that help to perform automation. It also provides guidelines on when and how to use these products.

### **6.9.1 Using NetView for automation**

NetView is an SNMP-based application used to manage IP networks and devices. It can issue SNMP commands to track the status of network devices and to perform actions upon them. By integrating with IBM Tivoli Switch Analyzer and

third-party management applications, NetView extends its management capabilities to layer 2 devices. It enhances its automation functionality with the commands supplied by third-party software.

NetView is an ideal tool to use to perform automation on networking devices by using SNMP commands and those supplied by integrated management applications. It can also be used to trigger the automated actions in response to an event or trap.

At this point, we must differentiate between executing automated actions and triggering them. *Executing automated actions* implies running a command or series of commands. *Triggering automated actions* responds to an event and sets off a course of actions that culminates in the execution of the automated action. The machine that triggers the action may not be the one to execute it. For example, NetView may send a trap to the Tivoli Enterprise Console as an event. IBM Tivoli Enterprise Console may perform some correlation and determine an action that is required for the event. IBM Tivoli Enterprise Console may run a Tivoli Framework task against the NetView machine, causing it to perform an SNMP set command on a networking device. In this case, IBM Tivoli Enterprise Console triggers the automated action in response to the event, but the NetView machine actually performs it.

In general, you execute the required commands as close as possible to the device requiring action. Keeping the path between the management station and the device short minimizes the likelihood that the command, or its response, will be lost or that the communications path will be unavailable. For most networking devices, the NetView management server is the point closest to the device at which automated actions may be executed.

Also, trigger the action from the closest point at which all necessary events are available for correlation. This ensures that an accurate decision can be made concerning the need to take action for the events. If the determination to automate can be made at NetView, trigger the actions from NetView. Otherwise, initiate them from IBM Tivoli Enterprise Console or another event processor at which all relevant events are received.

There are several ways to use NetView to execute and trigger automation. These are covered in the following section.

### **Executing commands from trapd.conf**

If an action is desired every time a trap is received, NetView can be configured to respond by placing the command in the `/usr/OV/conf/C/trapd.conf` file. This approach is often used to execute commands that run quickly and scripts that issue user defined SNMP traps. For example, sometimes business impact managers rely on additional information that is not present in the original traps

such as device type. Typically, the traps of interest are defined to execute a script that determines the additional data required, combines it with information from the original trap, and issues a second trap. The new trap, when processed, is the one forwarded to the business impact manager.

### **Configuring command execution from trapd.conf**

Add entries to the /usr/OV/conf/C/trapd.conf file by using the **xnmtrap** command.

Or from the NetView console, select **Options** → **Event Configuration** → **Trap Customization: SNMP**. This opens the NetView Event Configuration window (Figure 6-50). In the top portion of the window, highlight the enterprise which generates the trap. Then in the lower part of the window, select the trap itself and click **Modify**.

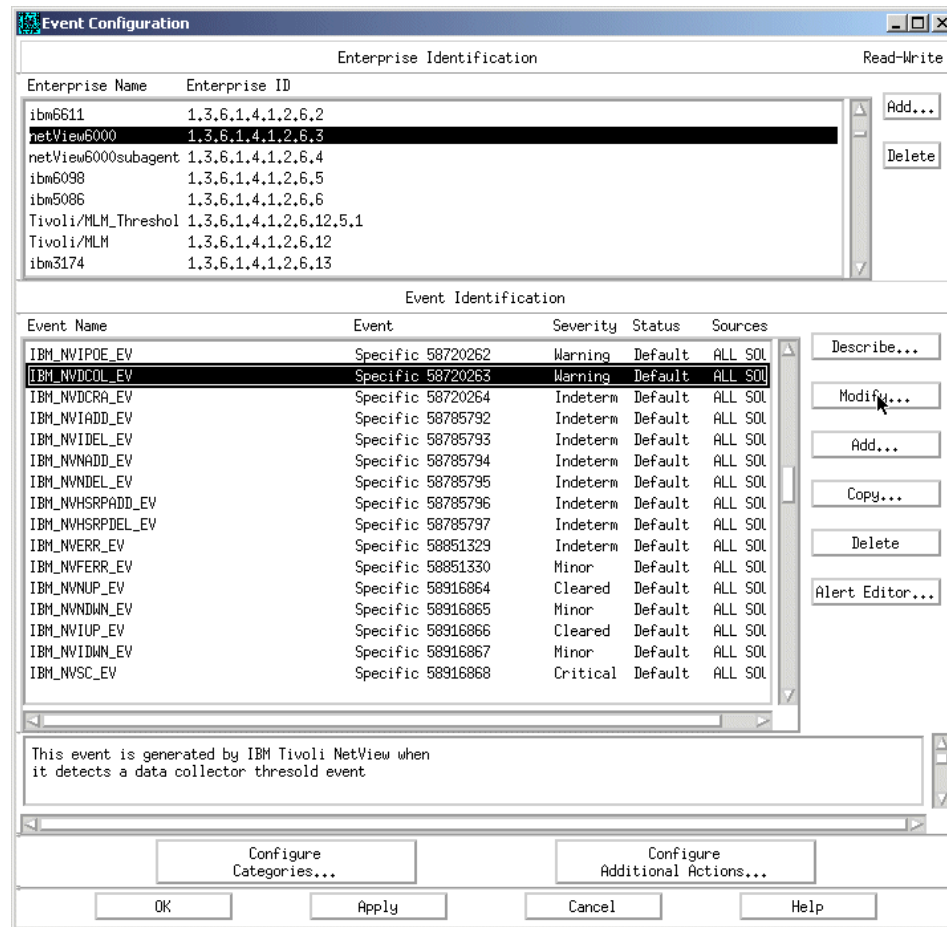


Figure 6-50 NetView Event Configuration window

In the Command for Automatic Action (Optional) box at the bottom of the window, type the command to execute. Click **OK** and then **Apply**. This tells the trapd daemon to reread the trapd.conf file for changes. In Figure 6-51, the /usr/OV/custom/scripts/threshold.ksh script is executed upon receipt of a NetView threshold trap.

**Modify Event**

Event Name  
IBM\_NVIDCOL\_EV

Generic Trap  
Enterprise Specific

Specific Trap Number  
58720263

Event Description  
This event is generated by IBM Tivoli NetView when it detects a data collector threshold event  
The data passed with the event are:  
1) ID of application sending the event

Event Sources (nodes) (all sources (nodes) if list is empty)

Source

T/EC Event Class  
TEC\_ITS\_SNMPCOLLECT\_THRESHOLD

T/EC Slot Map...

Event Category  
Threshold Events

Status  
Default Status

Severity  
Warning

Source Character  
D

Do Not Forward Trap

Event Log Message  
\$3 \$4

Popup Notification (Optional)

Command for Automatic Action (Optional)  
/usr/OV/custom/scripts/threshold.ksh

OK Reset Cancel Help

Figure 6-51 Modifying a trap for automated action in NetView

When trapd receives a trap, it passes it along to several other daemons. If the trap was configured to run a command, trapd passes it to the ovactiond daemon. This daemon is responsible for formatting the command and passing it to the shell for interpretation and execution.

The variable bindings in the trap are available for use by the script or command that is started by ovactiond. They are referenced by \$1, \$2, etc. within the automated action command. In our example, the threshold.ksh script can be passed to the first two variables from the trap by typing the following command in the Command for Automatic Action field:

```
/usr/OV/custom/scripts/threshold.ksh $1 $2
```

The variables are sanitized for security compliance. See “Security fix and its impact on NetView automated actions” on page 350 for more information. For a list of the variables that can be passed with each NetView internal trap, refer to Appendix A in the *IBM Tivoli NetView for UNIX Administrators Guide, Version 3.9*, SC32-1246.

### ***Customizing ovactiond***

The ovactiond daemon logs its output by default to the /usr/OV/log/ovactiond.log file. You can change this by modifying the options for the daemon using the **serversetup** command or by editing the /usr/OV/lrf/ovactiond.lrf file and restarting the daemon.

Another configuration parameter that you can change is maxWait time. This parameter sets the number of seconds ovactiond will wait for completion of the executed command. If maxWait is set to 0, ovactiond does not wait for the command to complete. If maxWait is set to a non-zero value, the return status and message can be logged to the log file. If the command does not complete within the requested time, it is ended. Figure 6-52 shows the options window where this is changed.

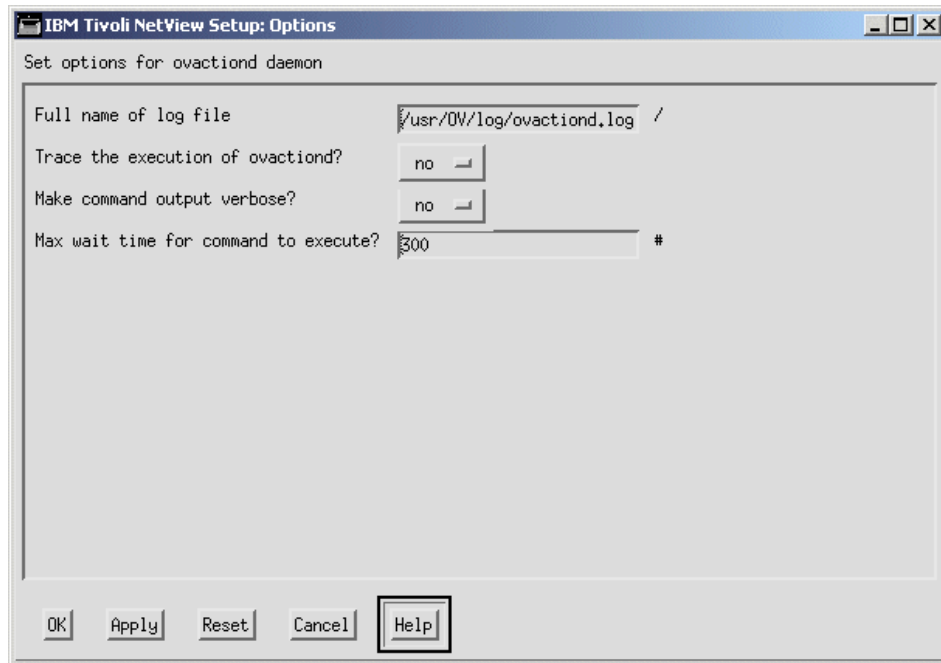


Figure 6-52 Setting ovactiond configuration parameters using serversetup

### **Performance considerations of using ovactiond**

If `maxWait > 0`, the `ovactiond` daemon waits until either the automatic action ends or until `maxWait` amount of time expires before executing the next command. If the actions are long running, or can time out, the queue of commands awaiting execution by `ovactiond` can grow.

An example of this is attempting to validate node down or interface down traps by automatically executing a demand poll of the resource referenced in the trap. If the down trap is legitimate (not due to a missed ping or SNMP status request) and `maxWait > 0`, `ovactiond` waits either `maxWait` seconds or until the demand poll times out before executing the next command. If there are several outages at once, the queue of commands awaiting execution by `ovactiond` can grow quickly.

Coding `maxWait = 0` or running the command as a background process can eliminate this problem. However, prevents the audit trail of execution errors from being recorded in `/usr/OV/log/ovactiond.log`, even when tracing is enabled.

### **Executing commands from the rule sets**

In 6.3.1, “Correlation with NetView and IBM Tivoli Switch Analyzer” on page 218, we discuss the concepts of NetView rule sets and ruleset nodes. This section explains how to use them to implement automated actions.

### ***Configuring automation in a NetView rule set***

To initiate automation through a NetView rule set, use the Action or Inline Action nodes in conjunction with the appropriate decision nodes. The decision nodes test for the criteria that the traps must meet before automation is initiated for them. These are discussed in “Correlation using NetView rules” on page 226.

The main difference between Action and Inline Action is how they are executed. Inline Action is performed by the `nvcorrd` daemon. Rule processing waits for completion of the action before continuing to the next node in the rule set. Action is performed asynchronously by the `actionsvr` daemon. Rule processing continues after spawning a process to run the action.

When defining Action or Inline Action nodes, specify the operating system, NetView command, or fully-qualified script name to execute when an event is forwarded to this node. The Inline Action node also allows setting fields that govern how long the script will run.

Unlike a command specified in an Action node, a command specified in an Inline Action node is not sent to the `actionsvr` daemon. Instead, the command is executed immediately. Processing continues to the next node if the return code of the action matches the return code that you specify within the specified time period.

The Inline Action window (Figure 6-53) contains the following relevant fields:

- ▶ **Command:** Specifies any operating system command, the full path name of any shell script or executable, or any NetView command.
- ▶ **Wait Interval:** Specifies the time period, in seconds, that the ruleset processor should wait for the specified action to return. Values can be in the range of 0 through 999 seconds. If the wait interval is 0, the return code from the action is ignored and processing immediately proceeds to the next node. If a wait interval is specified, and the return code from the action is not received in the wait interval, it is considered to be a failure and processing does not proceed to the next node. If the action is not completed within the specified time period, processing does not proceed to the next node.
- ▶ **Command exit code comparison:** Specifies the type of comparison that you want to make.
- ▶ **Exit Code:** Specifies the return code value from the specified action that you want to use in the comparison.



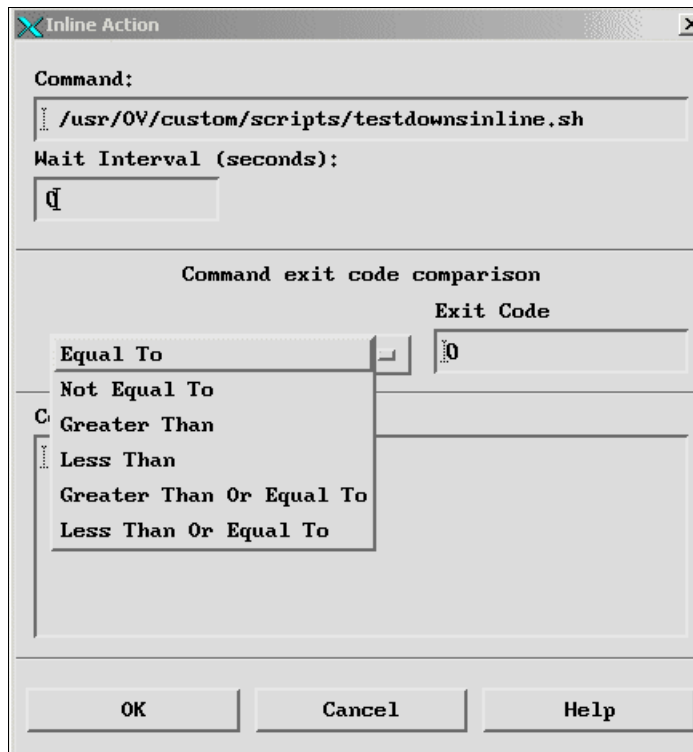


Figure 6-53 Inline Action window

### Activating rule sets

There are three ways to activate a rule set:

- **ESE.automation file:** You can activate one or more rule sets for automatic action when you start NetView by editing the `/usr/OV/conf/ESE.automation` file, adding the names of the rule sets on separate lines. To change the file when NetView is already active, make changes and recycle the `actionsvr` daemon for them to take effect. Use this method for rule sets that need to perform a specific action regardless of whether there is an event display open.

**Important:** Do not set the default processing action as Pass in the Event Stream icon or include a Forward node in rule sets started from the `ESE.automation` file. Forwarded events are passed to the `actionsvr` daemon, which has no events display and no mechanism to dequeue the events. Therefore, the socket connection to the `nvcorrdd` daemon fills up and causes the `nvcorrdd` daemon and all `nvevents` windows to hang.

Since ESE.automation runs regardless of whether there is an open console, it is an effective method to execute rule sets that perform notifications such as paging or trouble ticketing, recovery actions, or problem verification. It is also a useful way to execute scripts that add information to a trap by generating a second trap or an event that contains the additional data.

- **Filtered event work space:** You can activate a new or modified rule set by creating a new dynamic work space and pointing to the rule set that you want to use. For each dynamic workspace, you can activate only one rule set and one or more filters.

In the main work space, select **Create** → **Dynamic Workspace**. On the window that opens, enter the ruleset name and any other appropriate information. The new work space uses the rule set and filters, if any, to determine which events are displayed. If you edit a rule set while it is active, close and reopen the dynamic workspace window to put the changes into effect. In the Dynamic Workspace window, click the Help button for information about the window's fields.

Filtering traps from display on the Event Display is effective for reducing CPU consumption. While filters add cost to the processing of traps (check the trap to see if the filter applies), the cost of checking the filter is far less than the cost of displaying the trap on the Event Display.

Because a rule set activated in this fashion executes only when the event work space is open, use it only in production to perform actions on behalf of a particular NetView user. An operator can filter events from display by using a filtered event work space. Or the operator can execute a script to gather diagnostic data when a certain event is received. Automated actions may need to run regardless of whether an operator is logged on and using the event display. Execute these actions using another method, such as ESE.automation. We do not recommend using filtered event work spaces for triggering these types of automation.

An exception to this guideline is to employ this method of activating a rule set to test automation before using the rule set in ESE.automation or with nvserverd. You can open the work space using the rule set, generate test traps, and observe the results. If the rule set does not perform as expected, modify the rule, and re-open the dynamic workspace to quickly test again.

- **IBM Tivoli Enterprise Console forwarding:** In “Using nvserverd” on page 189, we explain how to activate a NetView rule set by inserting the name into the configuration of the nvserverd daemon. This rule set can contain automation. Like ESE.automation, you can use this method for actions that need to be executed regardless of an open console.

In general, if the automation affects whether the event is forwarded to IBM Tivoli Enterprise Console (such as problem verification automation), add it to the rule

set used by nvserverd. Otherwise, use ESE.automation. Use filtered event work spaces to test rule sets containing automation.

### ***Problem verification before forwarding an event to IBM Tivoli Enterprise Console***

A common problem with NetView monitoring is the generation of false down traps due to missed status poll responses. If NetView does not receive a response to its SNMP query or ICMP ping for an object, it marks the object down and issues the appropriate trap.

We recommend that you determine whether the trap is a false warning before you forward it to the IBM Tivoli Enterprise Console event server. We describe a sample method of performing this verification which we implemented in our lab.

The solution is to automate a status check of the object for which the down is received. Should the test indicate the object is really available, NetView automatically changes its status to *up* and generates the appropriate up trap.

It is possible to modify the rule set used to forward events to IBM Tivoli Enterprise Console to hold the interface, node, and router down traps a few minutes to see if their clearing traps are received as a result of the automation. However, this IBM Tivoli Enterprise Console forwarding rule is typically large. Editing it can be cumbersome. Therefore, we decided to use the state correlation gateway rule to drop both the up and down events if they are received in close proximity to each other. See 6.3.2, “IBM Tivoli Enterprise Console correlation” on page 232, for more information about the state correction engine.

To automate the status check, we wrote a NetView rule called *testdowns.rs* that checks for interface down, node down, and router down traps. It also runs a script, `/usr/OV/custom/scripts/testdowns.sh`, to perform a test of the object referenced in the trap, as shown in Figure 6-54.

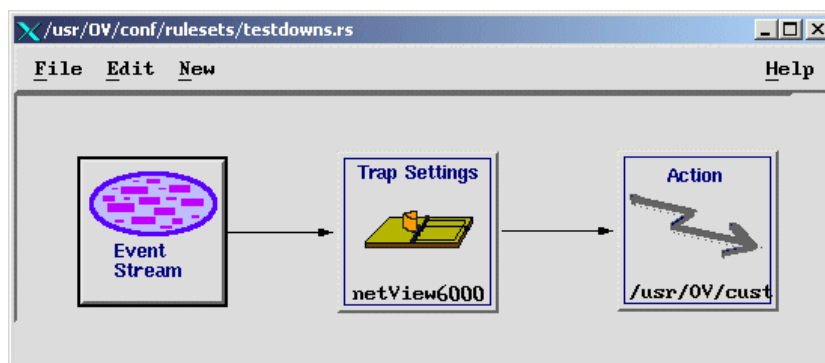
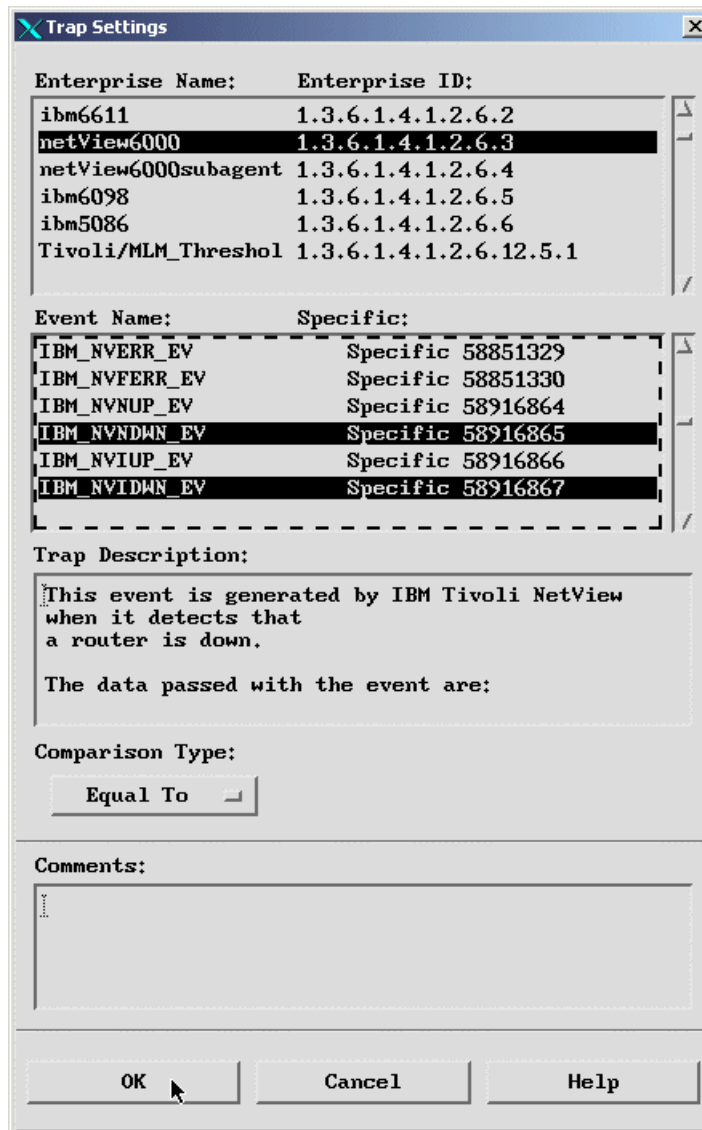


Figure 6-54 The rule set (*testdowns.rs*) to validate down traps

The testdowns.rs rule set uses the event stream default of block to prevent forwarding traps that do not meet the trap criteria to nvcorrdr. Node down, interface down, and router down traps are defined in the Trap Settings node, shown in Figure 6-55. This means that these types of traps are forwarded to the next node, the Action node that executes the appropriate script.



The image shows a 'Trap Settings' dialog box with a title bar containing a close button. It contains several sections: 'Enterprise Name' and 'Enterprise ID' with a list of entries; 'Event Name' and 'Specific' with a list of events where several rows are highlighted; 'Trap Description' with a text area; 'Comparison Type' with a dropdown menu; and 'Comments' with a text area. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Enterprise Name:	Enterprise ID:
ibm6611	1.3.6.1.4.1.2.6.2
netView6000	1.3.6.1.4.1.2.6.3
netView6000subagent	1.3.6.1.4.1.2.6.4
ibm6098	1.3.6.1.4.1.2.6.5
ibm5086	1.3.6.1.4.1.2.6.6
Tivoli/MLM_Threshol	1.3.6.1.4.1.2.6.12.5.1

Event Name:	Specific:
IBM_NVERR_EV	Specific 58851329
IBM_NVFERR_EV	Specific 58851330
IBM_NVNUP_EV	Specific 58916864
IBM_NVNDWN_EV	Specific 58916865
IBM_NVIUP_EV	Specific 58916866
IBM_NVIDWN_EV	Specific 58916867

**Trap Description:**

This event is generated by IBM Tivoli NetView when it detects that a router is down.

The data passed with the event are:

**Comparison Type:**

Equal To

**Comments:**

OK Cancel Help

Figure 6-55 Selecting multiple traps in the Trap Settings node

We entered this ruleset name into ESE.automation (Example 6-54) to ensure that it is executed for each trap generated. We recycled the actionsvr daemon.

*Example 6-54 ESE.automation file containing testdowns.rs rule set*

---

```
#This file should contain a list of filenames
#that will be automatically started by actionsvr.
#Each ruleset name is on a separate line; the pound sign
#indicates a comment line.
#An example line, with the name commented out) is below:
#your_ruleset_here.rs
testdowns.rs
```

---

## **Best practices for automation with NetView**

To help you determine when to use ovactiond or actionsvr to perform automation, follow these performance considerations:

- ▶ Do not use ovactiond for long running processes unless necessary. Use it for automation that generates second traps that contain more information.
  - The ovactiond daemon processes serially. If one command is long running, the others wait.
  - Processes can be ended if they time out.
  - Setting maxWait=0 or running the actions in the background fixes the first problem, but prevents errors resulting from the commands to be logged to ovactiond.log.
  - You have to specifically pass data from the trap to it.
- ▶ Use actionsvr for executing long running automated actions.
  - The actionsvr daemon spawns a child process and allows event processing to continue.
  - It still reports the action's return code.
  - It makes all data in the trap available to the action as environment variables.
- ▶ Monitor the daemon performance to determine how to best split the automation tasks between them.

Rule sets are processed by the nvcorrdd daemon, which invokes the actionsvr daemon to perform automation. If many rule sets are in use, either from ESE.automation or through NetView event displays, the nvcorrdd daemon can become busy. In these cases, it makes sense to split the automated tasks between the daemons and assign each the automated actions that are most suited to it.

## Security fix and its impact on NetView automated actions

A fix has been made to ovactiond, nvcorr, and actionsvr to close a potential security hole. This hole may allow any non-authorized user, with some knowledge of NetView trap customization, to gain root access to the NetView system by sending a trap to the NetView system from anywhere in the network.

This did not happen in the product as it is shipped, but can occur after trap customization is done by the NetView administrator or anyone with root authority on the NetView system. The security hole opened when a trap was customized to include a variable in the Command for Automatic Action field. A trap can then be sent from any system using command substitution, rather than the intended variable, to execute unauthorized operating system commands on the NetView system.

The UNIX daemons impacted by this fix are ovactiond, nvcorr, and actionsvr. The Windows daemons impacted by this fix are nvcorr and trapd. These daemons now filter out all non-alphanumeric characters except for the minus sign (-) and the decimal point (.). All characters that do not fall into this set are replaced with an underscore (\_). If a minus sign or decimal point is encountered, it is escaped (preceded by a back slash (\)) as a precaution.

If any non-alphanumeric character is encountered (and filtering is not disabled), a message is logged to the appropriate log file (if logging is enabled). On UNIX, the log files are /usr/OV/log/nvcorr.aalog, /usr/OV/log/ovactiond.log, and /usr/OV/log/nvaction.aalog. On Windows, the log files are \usr\ov\log\nvcorr.aalog and trapd.log.

The modified characters include: \$, ', ;, &, |, @, #, %, ^, <, >, /, \, =, {, }, -, ", and !. When these characters are encountered, a message is entered into the appropriate daemon log file.

This list of filtered characters can be configured by creating a variable (UNIX) or a registry variable (Windows) called *AdditionalLegalTrapCharacters*. If you set this variable to disable, then no filtering is done. If you set the variable to a string containing nonalphanumeric characters, then the filtering allows those characters to pass through the filter, but they are escaped.

Stop and restart the NetView daemons after setting the variable.

**Note:** On UNIX, the best way to set an environment variable for an ovspmd-controlled daemon is to place the definition of the environment variable into the /usr/OV/bin/netnsrc.pre file.

When a variable contains one of the special characters listed previously, it is sanitized to include a \ in front of the special character. The command that runs

needs to account for the \. If a custom script is executed, it can be coded to remove the appropriate \ characters. Example 6-55 shows a Korn shell script snippet that provides this function.

*Example 6-55 Korn shell code to handle sanitized variables*

---

```
#!/bin/ksh
#
# The purpose of this script is to simulate a user script which
# sends a page after some processing to test nvcorrd and actionsvr
#
# We used to send the following:
# /usr/OV/bin/nvpage 1234567@skytel hi joe `date` $NVS $NVATTR_2 $NVATTR_3
# Now we will use sed to remove the escape characters
set -x
# fix up $NVATTR_2 to remove backslash “\”
host=`echo $NVATTR_2 | sed “s:\\\\\\\\: :g”`
echo $host
/usr/OV/bin/nvpage 1234567@skytel hi joe `date` $NVS $host $NVATTR_3
exit
#
```

---

## 6.9.2 IBM Tivoli Enterprise Console

There are two main methods of executing automation from IBM Tivoli Enterprise Console. Tivoli tasks can be run from an IBM Tivoli Enterprise Console rule, or a script can be executed. The method that you choose depends upon the purpose of the automation.

### Tasks

If you want to run a task out of a rule from IBM Tivoli Enterprise Console, you can use the `exec_task` predicate. For example, you may have a task called `example_task` in the T/EC Tasks library, which runs a script called `example.sh` and echoes “hello world” into a log file that you want to run every time you receive an event of the class `EVENT`. In this case, you can use a rule such as the one shown in Example 6-56.

*Example 6-56 Running a task from within a rule using the `exec_task` predicate*

---

```
rule:
run_task_example:
(
    event: _event of_class EVENT where
```

```
[
    status: equals 'OPEN',
    hostname: _hostname
],
reception_action:
(
    exec_task(_event, 'example_task', '-l "T/EC Tasks" -h "%s"', [_hostname],
'YES'),
    commit_action
)
).
```

---

This rule runs the task specified on whichever host name the event came from. If you look at the line starting with `exec_task`, you can see, at the end, that there is a 'YES'. If this parameter is set to YES, you can see a display of the output from the task on the event console. You can see this by clicking the icon next to the event. Then you see a window showing you the output and the success or failure of the task, as shown in Figure 6-56.

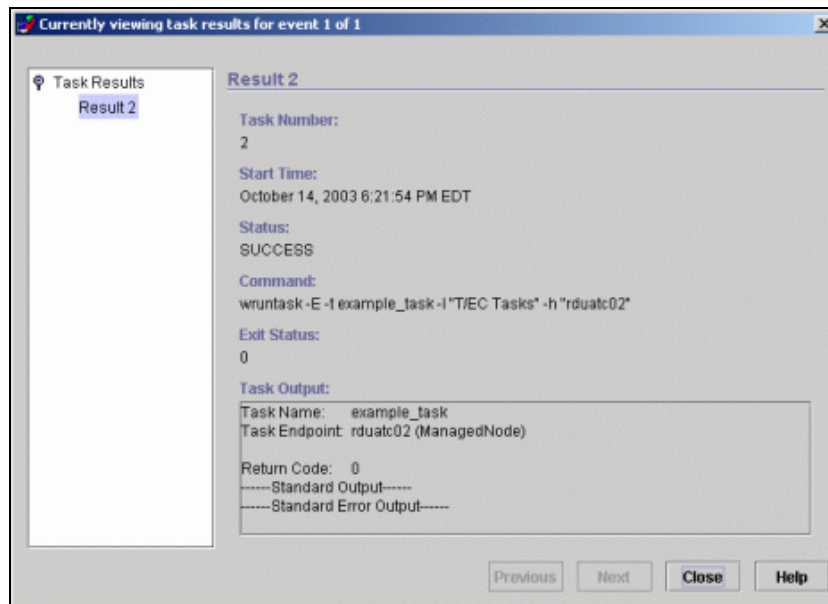


Figure 6-56 Output of a task executed from a rule

## Scripts

If you want to execute the `example.sh` script without using a task and you want to run it from the IBM Tivoli Enterprise Console event server itself, use the `exec_program` predicate.



**Note:** Any programs the run from the `exec_program` predicate must reside on the machine where the event server is located.

To use this predicate, we modify our rule as shown in Example 6-57.

*Example 6-57 Run a script from a rule using the `exec_program` predicate*

---

```
rule:
run_task_example:
(
  event: _event of_class EVENT where
  [
    status: equals 'OPEN',
    hostname: _hostname
  ],
  reception_action:
  (
    exec_program(_event, '/tmp/example.sh', '', [], 'YES'),
    commit_action
  )
).
```

---

This runs the same script that we ran from the task. Notice the 'YES'. You can also view the output from this program by clicking in the same icon as you would the task. The output should look similar to what is shown in Figure 6-57.

Use tasks to execute automation that must run on a system other than the event server.

Tivoli tasks can be executed upon any system defined to the Tivoli Management Region. Therefore, this is a good way to run automation on a problem system. Since the `exec_program` predicate runs a script on the event server itself, it is not as useful in automation that must run on the failing machine, such as gathering diagnostic data and attempting recovery.

It is possible to use `exec_program` to run a script on the event server and have the script trigger Tivoli tasks on the problem machine. This serves the same function as using the `execute_task` predicate for the target machine. However, it adds complexity, which makes it more difficult to determine what the rule set is doing and how to maintain it.

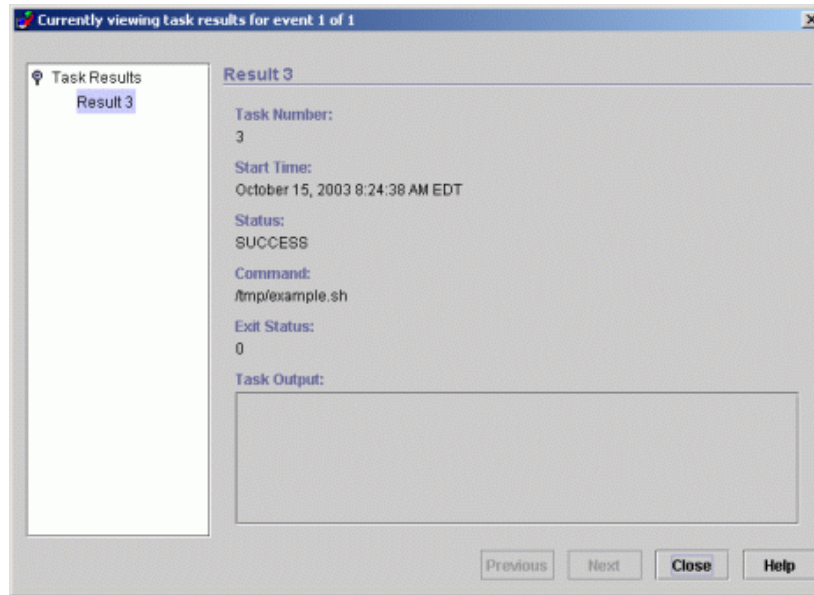


Figure 6-57 Output of program executed from within a rule

### 6.9.3 IBM Tivoli Monitoring

In IBM Tivoli Monitoring, you can specify your monitor to run any task once an indicator is met. You can do this from the actions section on the Indications and Actions window. This window is shown in Figure 6-58.

If you click the Tasks button, you can select the task that you want to run from a list of tasks. You can run any task that is defined in the TMR where your IBM Tivoli Monitoring installation resides.

Indications and Actions

Resource Model: Memory

Indications	Occurrences	Holes	Send TEC	Send TBSM	Severity
Low Storage Space	20	0	YES	NO	CRITICAL
System Thrashing	6	0	YES	NO	CRITICAL
Low Swap Space	4	0	YES	NO	CRITICAL

Low Storage Space

Number of Occurrences: 20

Number of Holes: 0

Send TEC Events: ☒

Send to TBSM: ☐

Severity: CRITICAL

Apply

Action List

- Built-in...
- Tasks...
- Recovery
- Edit...

Low Storage Space is generated when the percentage of available storage space is low compared to the total virtual available storage space. If this percentage is too low, it might affect the general performance of the machine being monitored.

Apply Changes and Close    Apply Changes    Cancel    Help...

Figure 6-58 IBM Tivoli Monitoring Profile for assigning tasks





## A case study

This chapter discusses a case study that was implemented within the International Technical Support Organization (ITSO) lab during the creation of this IBM Redbook. It describes the lab environment, installation issues that we encountered, and helpful diagnostic information regarding the products that we implemented.

## 7.1 Lab environment

This section provides a detailed description about the lab environment that we use in the case study. It includes a description about the software and versions that we use in the lab. Finally it looks at the reasons why we chose this layout for our environment.

### 7.1.1 Lab software and operating systems

This lab environment uses the following software and operating systems.

#### Operating systems

The operating systems that we install in our lab are:

- ▶ AIX 5.1 with AIX 5110–2, running on level 12
- ▶ Windows 2000 Server with Service Pack 3
- ▶ Windows 2000 Professional with Service Pack 3
- ▶ Linux Intel
  - RedHat Version 7.2 (2.4.7-10 kernel)
  - RedHat Advanced Server Version 2.1
  - SuSE Version 7.2 (2.4.4-4GB kernel)
  - UnitedLinux Version 1.0

#### Databases

We use DB2 Universal Database™ (UDB) 7.2 with Fix Pack 7.

#### Tivoli products

The Tivoli products that we use are:

- ▶ Tivoli Management Framework, Version 4.1 with:
  - 4.1–TMF–0010E
  - 4.1–TMF–0013
  - 4.1–TMF–0014
  - 4.1–TMF–0015
- ▶ IBM Tivoli Enterprise Console 3.9
- ▶ IBM Tivoli NetView with Integrated TCP Service Component 7.1.4
- ▶ IBM Tivoli Switch Analyzer 1.2.1
- ▶ IBM Tivoli Monitoring 5.1.1
  - IBM Tivoli Monitoring Component Services 5.1.0 with 5.1.0-ITMCS-FP01
  - IBM Tivoli Monitoring for Business Integration 5.1
  - IBM Tivoli Monitoring for Web Infrastructure 5.1.1
  - IBM Tivoli Monitoring for Databases 5.1

## Applications

We use the following applications:

- ▶ WebSphere Application Server Version 5.0 Base Edition
- ▶ Peregrine ServiceCenter Version 4.0.4
- ▶ WebSphere MQ 5.2 or 5.3 (IBM Tivoli Monitoring)
- ▶ Java Runtime Environment (JRE) 1.3.1

### 7.1.2 Lab setup and diagram

This section looks at the layout of our lab environment. It includes a description of the software that we install on each individual box. It discusses the network setup and naming conventions.

#### Naming conventions

We use the following naming conventions:

- ▶ The first three letters correspond to the location of the machine.
- ▶ The next letter is either an A for AIX or W for a Windows operating system.
- ▶ The next two characters correspond to that machine's purpose. For Tivoli machines, we use an abbreviation of the Tivoli software installed on the machine. For endpoints, we specify whether they are an application server or file server. The majority are application servers to host DB2, WebSphere MQ, and WebSphere Application Server.
- ▶ For the machines installed with Tivoli software, TC corresponds to IBM Tivoli Enterprise Console, NW corresponds to Integrated TCP Service Component in NetView, WS corresponds with WebSphere Application Server, and TF corresponds to Tivoli Management Region (TMR) (Tivoli Framework).
- ▶ The last two characters are a unique two-digit number for that machine.

For example, the first machine located in Raleigh, North Carolina, has Windows 2000 as its operating system and is used as a file server. It has the name RDUWFS01.

**Note:** Having a standard naming convention for your endpoints greatly increases the ease of correlating events.

#### Lab layout

Figure 7-1 shows the layout that we create in the ITSO lab for our case study and testing purposes.

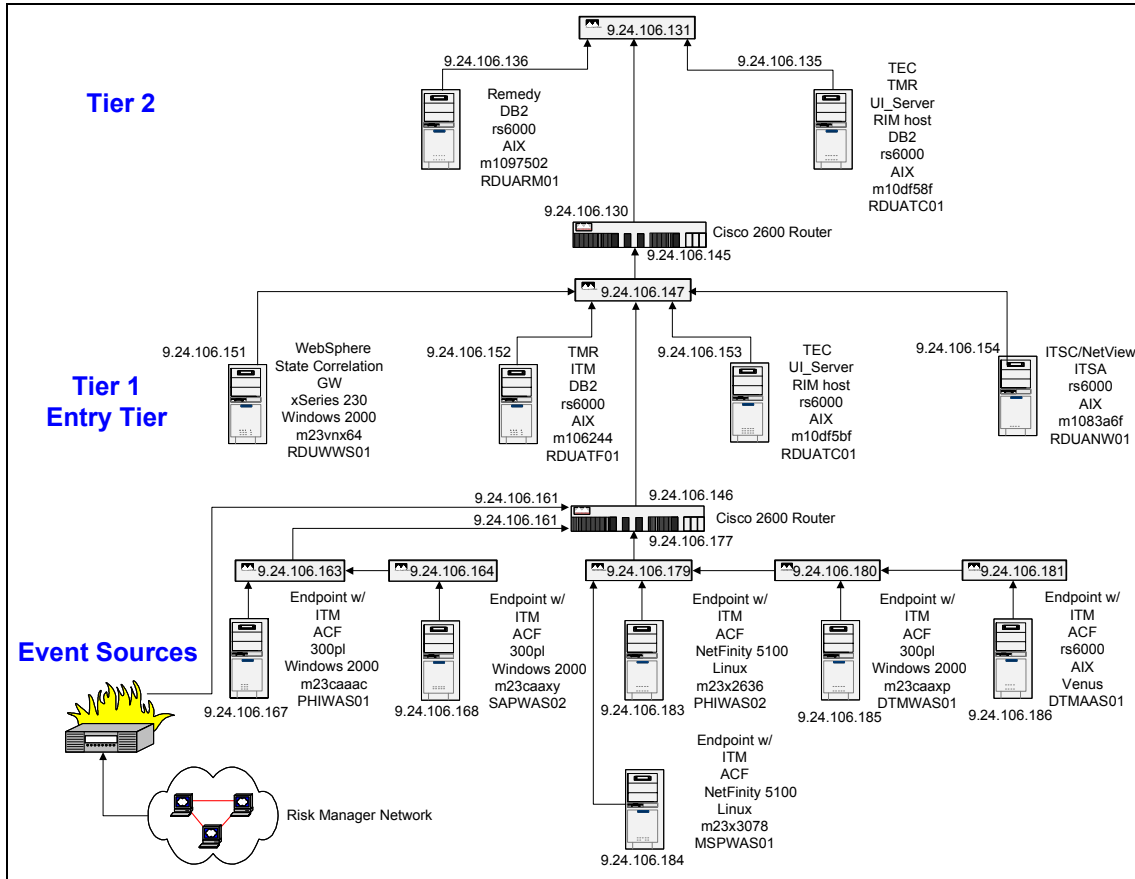


Figure 7-1 Lab layout

## Servers

We use a total of six AIX-based and six Intel-based servers. The following list includes the names of these servers and their purpose. Refer to Figure 7-1 for their location in the network.

- ▶ The servers beginning with RDU located in Raleigh are:
  - RDUARM01: This server hosts RemydyHelp Desk for trouble ticketing. A DB2 database is also installed for Remydy's use.
  - RDUATC01: This server hosts the Focal Point IBM Tivoli Enterprise Console. It has the UI server, RIM host, and a TMR installed. This server also hosts an instance of DB2 for the use of the Focal Point IBM Tivoli Enterprise Console.



- RDUWWS01: This server hosts a WebSphere server for the IBM Tivoli Enterprise Console Web Console. It is installed with a gateway to run the IBM Tivoli Enterprise Console Gateway Process and state correlation.
- RDUATF01: This server hosts another TMR for the use of the lower-level IBM Tivoli Enterprise Console. IBM Tivoli Monitoring is installed on this TMR. An instance of DB2 is installed for the lower-level IBM Tivoli Enterprise Console server's use.
- RDUATC01: This server hosts the lower-level IBM Tivoli Enterprise Console. It also has the UI server and RIM host installed.
- RDUANW01: This server hosts Integrated TCP Service Component in NetView as well as IBM Tivoli Switch Analyzer.
- The following servers are used for an endpoint with IBM Tivoli Monitoring Resource Models and ACF Logfile Adapters running at each site:
  - Server starting with MSP for Minneapolis, Minnesota
    - MSPWAS01
  - Server beginning with SAP located in San Paulo, Brazil
    - SAPWAS01
  - Servers beginning with PHI located in Philadelphia, Pennsylvania
    - PHIWAS01
    - PHIWAS02

**Note:** These PHI servers are a Linux cluster.

- Servers beginning with DTM located in Dortmund, Germany
  - DTMAAS01
  - DTMWAS01

## Network components

We use a combination of one Cisco 2600 router, one Cisco 3600 router, and seven switches. Our main router, the Cisco 3600, is named rdur02. The 2600 router is named rdur01. For our purposes, here we refer to our switches by their IP address. Refer to Figure 7-1 for the general layout.

Our lab has four networks, 9.24.106.160, 9.24.106.176, 9.24.106.144, and 9.24.106.128. The main router rdur02 is connected to the Risk Manager Network and connects the 9.24.106.160 and 176 networks.

In the 160 network, we have two switches: 9.24.106.163 and 9.24.106.164. The 164 is cascaded off of the 163, and the 163 is connected to rdur02. Both switches on this network have one server connected to them.

In the 176 network, there are three switches: 9.24.106.179, 9.24.106.180, and 9.24.106.181. The 180 and 181 are cascaded off of the 179, and the 179 is connected to rdur01. Switch 179 has two servers connected to it. The other two switches have one server connected to them.

In the 144 network, we have one switch 9.24.106.147. This switch is between rdur01 and rdur02 and has four servers connected to it.

In the 128 network, we have one switch 9.24.106.131. This switch is connected to rdur01 and has two servers connected to it.

### **7.1.3 Reasons for lab layout and best practices**

We set up our lab to try to test most situations that customers would configure in their networks. In this section, we discuss the reasoning behind our lab layout and the positioning of the IBM Tivoli software.

#### **Network layout**

We are limited to the amount of networking hardware and the type of network hardware available to the ITSO at the time of running the lab. We acquire and use two Cisco routers (one 3600 and one 2600) and seven Cisco Catalyst 1900 switches.

We position our network to try to create as many situations as possible in this limited test environment. First we divide our lab into four networks, one for each location that we specify. Our locations here represent virtual geographical locations. This assists us with using locations to separate each network event.

Another goal we accomplish is to cascade at least one network switch off of another network switch. This determines whether IBM Tivoli Switch Analyzer recognizes a failure on the cascaded switch and forwards the event to NetView. For this same reason, we locate a switch between our main and secondary router. In addition, we use our limited network hardware to include at least one switch in each of our virtual geographic network locations.

#### **Machine layout**

In conjunction with our network layout, we locate at least one machine in each of our four locations. The servers at each location represent servers that run business critical applications. Again, we use this configuration to assist with event management and event source location. This type of information is invaluable when working specifically on correlation and other event management disciplines.

## Tivoli software

The IBM Tivoli software configuration in the lab consists of two interconnected TMRs, each with a local IBM Tivoli Enterprise Console installation. This is configured to show an IBM Tivoli Enterprise Console hierarchy and to explain how to implement an event management solution in such an environment. In this IBM Tivoli Enterprise Console configuration, the lower-level IBM Tivoli Enterprise Console server handles filtering and drops unnecessary events. In turn, it sends only events to the higher-level IBM Tivoli Enterprise Console when those events should be acted upon.

We set up the IBM Tivoli software as a centralized installation. Our high-level servers, such as NetView, IBM Tivoli Switch Analyzer, TMF, and IBM Tivoli Enterprise Console are located in one location. TMF gateways are in each remote location to represent a typical centralized management solution.

## 7.2 Installation issues

This section describes the installation issues that we encounter during our case study in the ITSO labs.

### 7.2.1 IBM Tivoli Enterprise Console

For IBM Tivoli Enterprise Console, we encounter the problems presented in the following sections.

#### **tec\_gateway**

The Adapter Configuration Facility (ACF) `tec_gateway_sce` does not contain file `.tec_gateway_diag_config`.

#### **Windows**

In Windows, you change the `tec_gateway.conf` file. The parameter that points to the Extensible Markup Language (XML) file is incorrect by default. You must change all back slashes (\) to forward slashes (/), as shown in Example 7-1.

*Example 7-1 `tec_gateway.conf` file changes for Windows*

---

```
# Fri Oct 03 09:12:11 2003
#
# tec_gateway Configuration
#
ServerLocation=@EventServer
GatewaySendInterval=5
```

```
GatewayQueueSize=40000
BufEvtPath=C:\WINNT\system32\drivers\etc\Tivoli\tec\tec_gateway_sce.cache
Pre37Server=no
UseStateCorrelation=YES
StateCorrelationConfigURL=file:///C:/WINNT/system32/drivers/etc/Tivoli/tec/tecr
oot.xml
SendEventPort=5561
ReceiveAckPort=5562
ReceiveEventPort=5563
SendAckPort=5564
gwr_ActiveConnections=20
gwr_ActiveConnectionsSafety=80
gwr_Enable=yes
LogLevel=ALL
LogFileName=c:\temp\tec_gateway.log
TraceLevel=ALL
TraceFileName=c:\temp\tec_gateway.trc
#
```

---

## 7.2.2 NetView

We do not encounter installation issues during the NetView installation.

## 7.2.3 IBM Tivoli Switch Analyzer

On AIX, ensure that there is at least 50 MB in /tmp, or run the installation using another temporary directory with at least 50 MB by issuing the following command:

```
./switchanalyzer_install -is:tempdir alternate temp directory
```

Make sure you have X Window System capability, since the InstallShield wizard is used.

Use the **./switchanalyzer\_install** command to invoke the InstallShield wizard and answer the appropriate questions. Figure 7-2 shows the initial window. Then click **Next**.

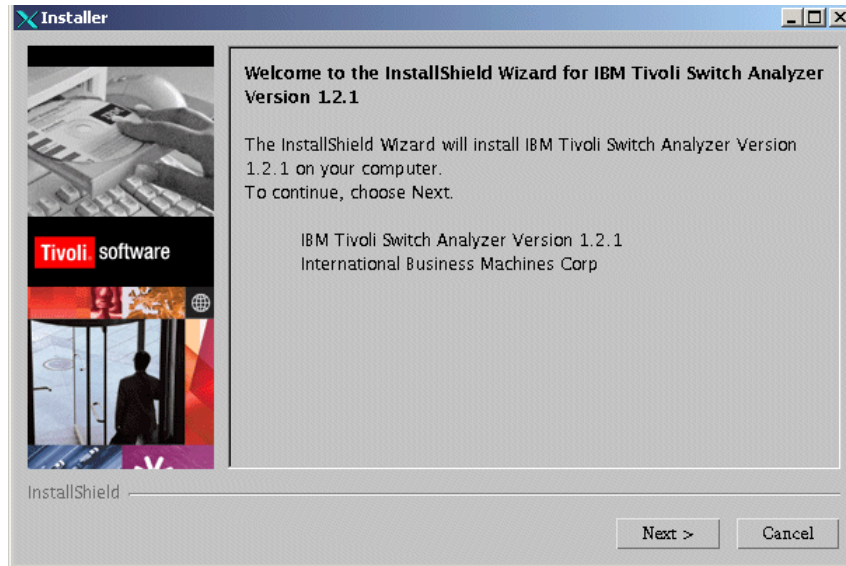


Figure 7-2 InstallShield wizard initial window

The Software License Agreement panel (Figure 7-3) opens. Agree to the licenses and click **Next**.

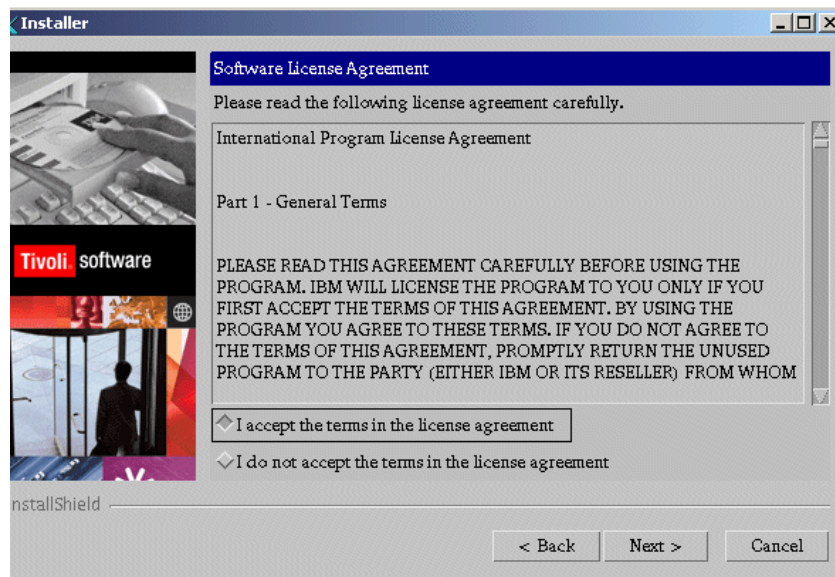


Figure 7-3 License agreement window

You see a window similar to the one in Figure 7-4. If you are ready to begin the installation, select **Yes** and click **Next**.

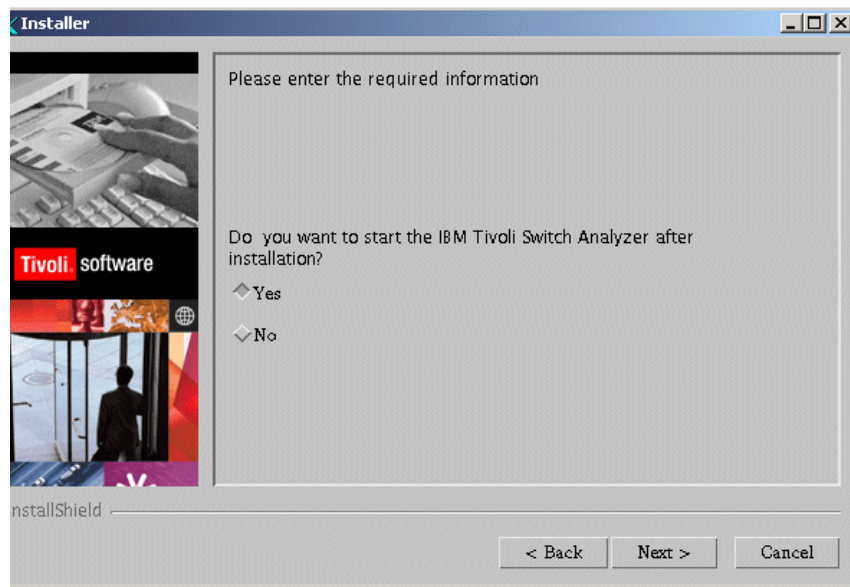


Figure 7-4 Confirmation window

You see a summary window (Figure 7-5). Confirm the information and click **Next**.

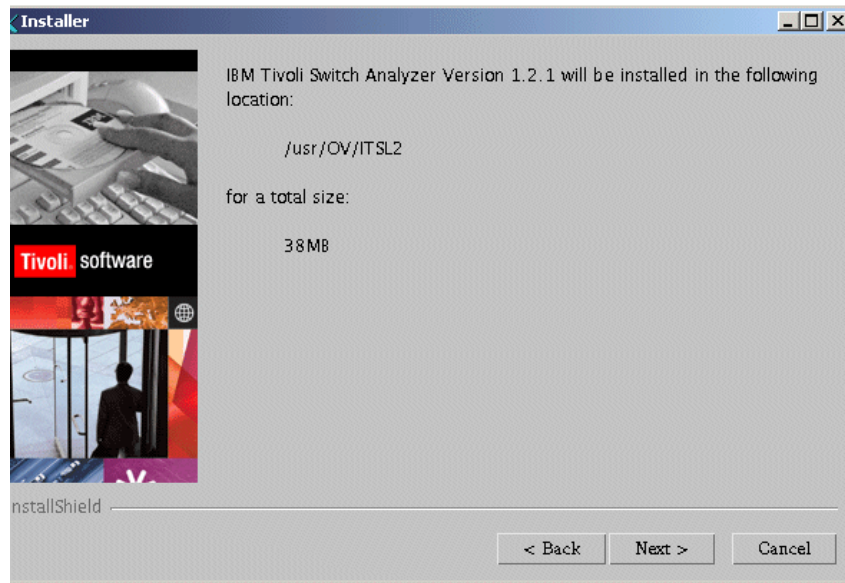


Figure 7-5 Summary window

The installation begins. When it completes, you see a window similar to the example in Figure 7-6.

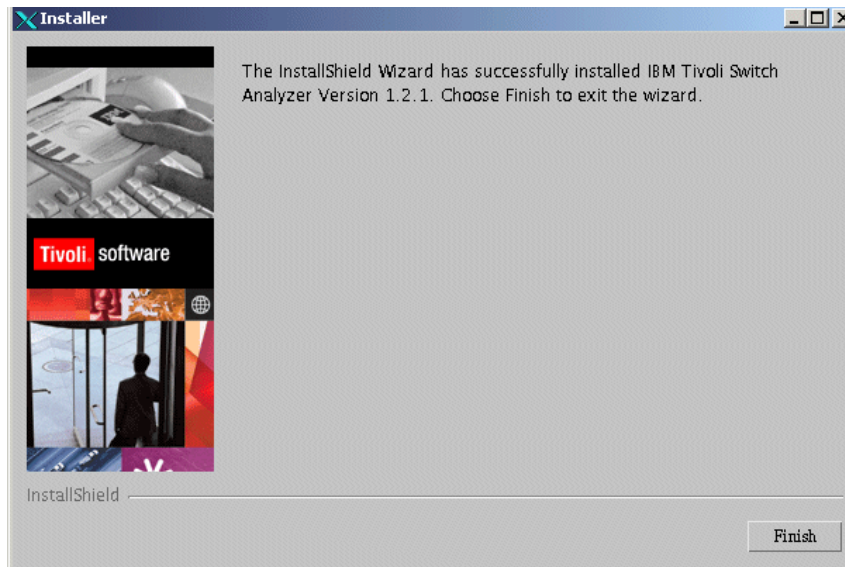


Figure 7-6 Completion window

When the installation is completed, verify that the daemon is running using the **ovstatus itsl2** command. Use the following procedure to manually start the itsl2 daemon:

- ▶ For UNIX, if the Tivoli NetView product is not started, always use the `/usr/OV/bin/serversetup` utility to start the daemons. This utility automatically starts the Tivoli Switch Analyzer product. If the Tivoli NetView product is started, use the Tivoli NetView **ovstop** and **ovstart** commands to start and stop itsl2 daemon. For example, use the following command to start the itsl2 daemon:

```
/usr/OV/bin/ovstart itsl2
```

- ▶ For Windows, use the following command to start IBM Tivoli Switch Analyzer:  

```
\usr\ov\bin\ovstart itsl2
```

Layer 2 root cause events and topology status updates are available immediately after IBM Tivoli Switch Analyzer is installed and started. IBM Tivoli Switch Analyzer reports are enabled immediately for the SuperUser role. However, you must activate them as follows:

1. From the main menu of the Tivoli NetView native console, select **Administrator** → **Security Administration** → **Web Console Security** to start the security console.
2. From the main menu, select **File** → **Save**, and then click **File** → **Restart Web Server**.

IBM Tivoli Switch Analyzer reports are not available for other user roles until you enable and activate them as follows:

1. From the main menu of the Tivoli NetView native console, select **Administrator** → **Security Administration** → **Web Console Security** to start the security console.
2. Check the following Tivoli Switch Analyzer reports for the required roles. The SuperUser role includes them all as the default value.
  - Impact Analysis
  - Impact Analysis (connectors)
  - Discovery
  - Rediscovery
3. From the main menu, select **File** → **Save**, and then click **File** → **Restart Web Server** to integrate the menu items, as shown in Figure 7-7.



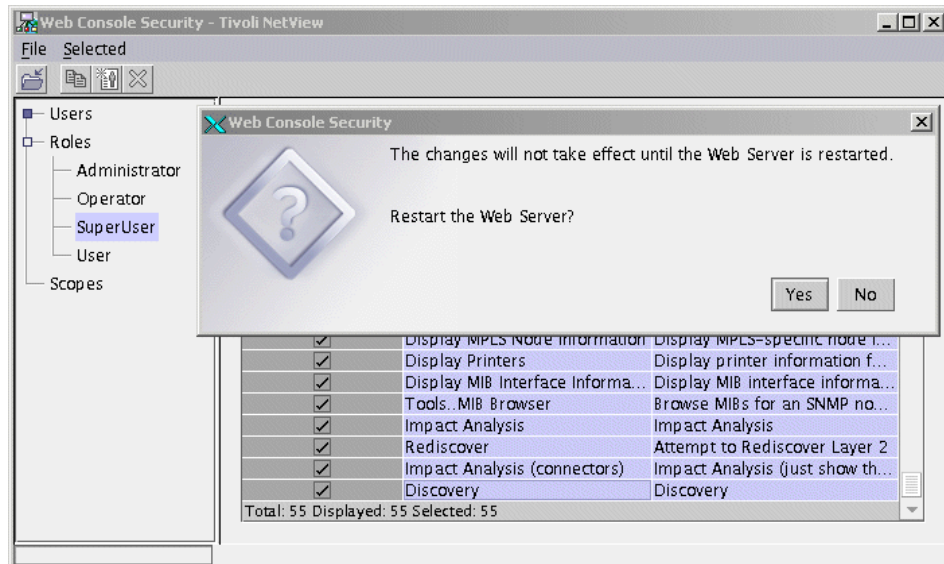


Figure 7-7 Restart Web Server window

IBM Tivoli NetView Web console users with the appropriate roles now have access to the IBM Tivoli Switch Analyzer report menu items.

The installation of IBM Tivoli Switch Analyzer performs the following tasks:

- ▶ Copies files and binaries to /usr/OV/ITSL2
- ▶ Updates IBM Tivoli Switch Analyzer specific files
- ▶ Imports layer 2 NetView object identifiers (OIDs)
- ▶ Updates system files
- ▶ Installs NetView files
- ▶ Creates itsl2 daemon
- ▶ Adds traps

Before IBM Tivoli Switch Analyzer can recognize a switch, you must define it as a switch in the Tivoli NetView database. Run the following command:

```
/usr/OV/bin/ovtopodump -X
```

This command determines the following information:

- ▶ That the machine the NetView product is installed on is discovered
  - If this machine is not discovered, the itsl2 daemon does not start.
  - The node must be managed.
  - All devices downstream from an unmanaged device are ignored by IBM Tivoli Switch Analyzer.

- ▶ That devices classified as switches by the Tivoli NetView product have Simple Network Management Protocol (SNMP) sysObjectId entries in the /usr/OV/ITSL2/conf/files/l2\_oids.cfg file

This file is automatically updated during installation of the IBM Tivoli Switch Analyzer using information from the IBM Tivoli NetView oid\_to\_type file.

- ▶ That the isConnector capability field is set to True and the isRouter field is set to False for the device object

The analysis output provided by the command includes the following information about a node:

- OVwDb object ID
- Node name
- IP status
- SNMP address
- sysObjectId
- Layer2Status field value

The command output also provides a column named Layer 2 OID?, which indicates whether the SNMP sysObjectId is missing from the /usr/OV/ITSL2/conf/files/l2\_oids.cfg file. This output is shown in Example 7-2. In Example 7-2, the switches are recognized, but still have no layer 2 status.

*Example 7-2 Recognized switches missing layer 2 status*

root:/ >ovtopodump -X						
Node ID	Object	IP Status	L2 Status	IP Address	SNMP OID	Layer 2 OID?
OID?						
541	rdusw03	Up	Unset	9.24.106.163	1.3.6.1.4.1.9.5.18	Yes
543	rdusw04	Up	Unset	9.24.106.164	1.3.6.1.4.1.9.5.31	Yes
549	rdusw05	Up	Unset	9.24.106.179	1.3.6.1.4.1.9.5.18	Yes
551	rdusw06	Up	Unset	9.24.106.180	1.3.6.1.4.1.9.5.18	Yes
555	rdusw07	Up	Unset	9.24.106.181	1.3.6.1.4.1.9.5.31	Yes
568	rdusw01	Up	Unset	9.24.106.131	1.3.6.1.4.1.9.5.18	Yes
605	rdusw02	Up	Unset	9.24.106.147	1.3.6.1.4.1.9.5.31	Yes

## 7.3 Examples and related diagnostics

This section provides a set of examples and diagnostics regarding things found in our case study.

### 7.3.1 Event flow

One of the first actions performed in debugging event correlation and automation is to trace the path of the event through various event processors. Determining

how far the event has proceeded gives an indication of where to look for the problem. This section explores the means that are available to verify the reception of events in NetView and IBM Tivoli Enterprise Console.

## Event reception in NetView

When troubleshooting NetView event flow, consider the items in the following sections.

### *Determining if an event was received by NetView*

There are several ways to determine if a trap was received by NetView:

- If you already have an active NetView console, see if the event is displayed in the event window. Make sure that the event window you are viewing is not one to which filters are applied, which may prevent display of the event of interest.

Examine the time stamp on the traps. If the most recent trap in the list occurred a while ago, it can be a sign that NetView is not processing traps or is being flooded by them. For flooding conditions, the event window still updates periodically with new entries. If trapd stops processing, no new events are displayed.

Become familiar with the rate at which you normally receive traps. This helps you to determine from the time stamp of the latest trap which condition has occurred.

Slow processing of traps may be due to long running inline actions in your rule sets. For example, if NetView executes automation that may time out, all other rule processing for that event waits until the end of the timeout interval.

- Examine the trapd.log file. If you follow the best practices described in Chapter 6, “Event management products and best practices” on page 173, all traps received by NetView are logged in this file, and several days worth of data are stored online in files.

If the trap of interest is not in the appropriate file, this may be the result of several reasons:

- The trapd daemon may not be running. Use the **ovstatus trapd** command to verify that the daemon is active. If it is not active, start it by executing the **ovstart trapd** command.
- There may not be a definition for the trap in the trapd.conf file. If the trap is received but not formatted, there may be an unknown format trap in trapd.log.
- The trap may be configured as “Don’t log or display”. Check the trapd.conf file (either by viewing the file directly or through the event configuration screen) to see if the trap is defined there. If it does not exist, add it. If it exists, ensure that it is not defined as “Don’t log or display”.

- SNMP in the networking devices may be improperly configured. This affects unsolicited traps sent by a device. It does not pertain to such traps as Node Down, which are generated by NetView.

If a device's log shows an error condition that you want to see at your consoles and no corresponding trap is received by NetView, this may result for a few reasons. The device may be misconfigured to suppress the trap, or its trap destination configuration may be set improperly. Check the SNMP definitions in the device to ensure that these are not the problem.

### ***Determining why a NetView event has not reached the IBM Tivoli Enterprise Console server***

If a NetView event is expected at IBM Tivoli Enterprise Console and does not arrive, this can result for several reasons. First, verify whether NetView itself received the trap by using the techniques outlined in the previous section. Next, ensure that IBM Tivoli Enterprise Console is actually receiving events from other sources. If it is, then the problem is on the NetView side. If not, the problem may be with the IBM Tivoli Enterprise Console server itself.

If NetView receives the trap but IBM Tivoli Enterprise Console does not, determine whether IBM Tivoli Enterprise Console is receiving other NetView traps from this server. If it is not, then perform the following steps. If it is, skip the first three steps.

1. Verify that the `nvserverd` daemon or `tecad_nv6k` adapter is running.  
Depending on your operating system and event processing requirements, use one of these methods to forward events to IBM Tivoli Enterprise Console.  
  
Use the **ovstatus** command to check the operation of the appropriate daemon. If the daemon is not running, start it by using the **ovstart** command.
2. If the appropriate daemon is running, check its configuration for the correct destination IBM Tivoli Enterprise Console server.  
  
For `nvserverd` forwarding, the `/usr/OV/conf/tecint.conf` file indicates the target IBM Tivoli Enterprise Console server as shown in Example 7-3.

#### ***Example 7-3 Two ways to specify target IBM Tivoli Enterprise Console in tecint.conf file***

---

```
ServerLocation=146.84.157.39  
TecRuleName=forwardall.rs
```

```
ServerLocation=tecserver.raleigh.tivoli.com  
TecRuleName=my_tec_rule.rs
```

---

The first method specifies the IP address of the destination IBM Tivoli Enterprise Console server. The second method specifies its name. Verify that

the entry in your `tecint.conf` file is correct for the destination IBM Tivoli Enterprise Console server and that NetView can route traffic to that subnet. If specifying by name, ensure that NetView can properly resolve the host name.

When using the NetView adapter, the information is stored in `\usr\OV\conf\tecad_nv6k.conf` file, as shown in Example 7-4. The `ServerLocation` parameter indicates the destination IBM Tivoli Enterprise Console. Again, ensure that the setting is correct and that NetView can reach the subnet.

---

*Example 7-4 Specifying the target IBM Tivoli Enterprise Console in `tecad_nv6k.conf`*

---

```
ServerLocation=tecsrvr.raleigh.tivoli.com
ServerPort=0
```

```
ServerLocation=@EventServer
ServerPort=5529
```

```
ServerLocation=146.84.157.39
ServerPort=0
```

```
ServerLocation=@EventServer#tmr-central
ServerPort=0
```

---

The destination IBM Tivoli Enterprise Console can be specified by host name, IP address (non-TME), or Tivoli Event Server object (TME). In an interconnected TMR environment, specify `@EventServer#region_name` if the IBM Tivoli Enterprise Console to which events should be forwarded is in an interconnected TMR.

Next, verify the port. Specifying 0 is typical, and means to use portmapper to determine the port. If a port is coded with a different value, ensure that it matches the setting of the `tec_rcv_agent_port` parameter in the `.tec_config` file on the IBM Tivoli Enterprise Console server:

```
tec_rcv_agent_port=5529
```

In firewall environments, ensure that the proper firewall rules exist to allow the traps to flow to the IBM Tivoli Enterprise Console.

3. Examine the NetView cache. If NetView is unable to reach the IBM Tivoli Enterprise Console server, it caches events until the connection can be re-established. The location of the cache file is specified in the configuration files for `nvserverd` and `tecad_nv6k` as shown in Example 7-5.

*Example 7-5 Specifying cache location in tecad\_nv6k.conf*

---

```
BufEvtPath=%SystemRoot%/system32/drivers/etc/Tivoli/tec/nv6k.cache
BufferFlushRate=5
```

---

In this example, events are cached in the file specified by the `BufEvtPath` parameter. The `BufferFlushRate` specifies the number of events sent per minute. When the adapter recovers the lost connection, and there are events in the buffer, the events are sent in bursts at this rate per minute. The default value is 0. All events are sent in one burst.

The `nvserverd` daemon caches events by default in the `/etc/Tivoli/tec/cache` file. This location may be changed by specifying `BufEvtPath=pathname` parameter in the `tecint.conf` file.

4. Review NetView's event forwarding configuration.

NetView may be configured, for example, to prevent the forwarding of a particular trap to NetView. Review each configuration option to ensure that it is not blocking the event.

When using `nvserverd` to forward events, check the rule set used by the daemon to see if there are any Trap Settings nodes or other logic that may prevent the trap from being forwarded. Also, see what IBM Tivoli Enterprise Console class to assign to the event either by viewing the Event configuration screen or the `trapd.conf` file.

For the NetView adapter, check for Filter statements in the `tecad_nv6k.conf` file. If the trap is listed, and `FilterMode=OUT`, the trap is blocked. If `FilterMode=IN` and the trap is not listed in a Filter statement, add it.

Likewise, if a `FilterCache` statement exists for the trap and the adapter goes into caching mode, it is possible that the event is dropped. The `FilterCache` keyword prevent traps from being cached in the event that IBM Tivoli Enterprise Console is unreachable. It is used for traps that are time-dependent, which are those for which action cannot be taken after a certain interval has passed.

5. If it is still not apparent why the event is not forwarded, then trace the appropriate subsystem.

Use subsystem tracing when you suspect a problem with the Tivoli NetView program. To trace subsystems with the `nettl` operation in the Server Setup application, use the **nettl** command, or complete the following steps:

- a. Log in as root.
- b. From the command line, perform the following steps:
  - i. Enter the **serversetup** command to invoke the Server Setup application.

- ii. Select **Diagnose** → **Set subsystem tracing and logging options (nettl)** → **Start subsystem (nettl) tracing**.
- c. The Start subsystem (nettl) tracing window opens. Enter the subsystem and trace type. For event management services, the subsystem is OVE. The trace options are proc, state, error, and logging. Refer to the online help for information about the fields in this window. Click **OK**.

The **nettl** command (and nettl option of the Server Setup application) creates a single log file of entries. By default, the file is /usr/OV/log/nettl.LOGxx, where xx is 00 or 01. To change the default log file, use the Server Setup application.

The **nettl** command requires root user authority. Disaster and error logging are enabled at system startup and should not be disabled.

Each log entry contains header information and a user data field. The header information has a date and time stamp, the name of the subsystem, a Log Class field indicating the type of event being logged, and other miscellaneous fields. Under the header, a text string describes the log event and a subsystem error number. In some cases, the text string may include corrective actions.

Use the **netfmt** command to format the nettl trace output.

- 6. Consider tracing the trapd daemon for further diagnostic information.

By default, trapd trace records are written to /usr/OV/log/trapd.trace. Use the **trapd -T** command to toggle tracing on and off for both NetView for UNIX and NetView for Windows.

If you suspect a problem with the trapd daemon on NetView for Windows, try the following actions to further isolate the problem:

- a. Look at the nv.log file for trapd process errors. The default log file is \usr\ov\log\tnv.log.
- b. At the command prompt, type:  
event
- c. Check that a Node Up trap appears in the Event Browser or SQL database.
- d. If the trapd daemon appears to be hung, stop all the daemons by double-clicking the **Stop Daemons** icon in the NetView program group or by using the **ovstop** command.
- e. Restart all the daemons by double-clicking the **Start Daemons** icon or by using the **ovstart** command.
- f. Ensure that adequate paging space is available:
  - i. Double-click the **Control Panel** icon.

- ii. Double-click the **System** icon.
- iii. Click **Virtual Memory** to check the available paging space. The recommended minimum is 120 MB. If necessary, increase the paging space.

### **Generating test traps**

There are various ways to generate test traps, some easier to execute than others. Table 7-1 shows a quick summary.

*Table 7-1 Methods of generating test traps*

Method	Syntax	Example
<b>event</b> command  Sends an event to the trapd daemon	event [-a data] [-b database] [-d descr] [-E event_number] [-e event] [-h node] [-l] [-n num] [-s source] [-t hostname] [-u hostname] [-T hostname] [-x]	event -E 58916865  This example sends a node down trap.
<b>snmptrap</b> command  Issues an SNMP trap	snmptrap [-d] [-t timeout] [-r retries] [-p port] [-c community] node enterprise agent-addr generic-trap specific-trap time-stamp [variable type value ...]	/usr/0V/bin/snmptrap -c public .1.3.6.1.4.2.6.3.1 nmstation.abcd.com <agent-addr > 6 31 1 .1.3.6.1.4.1.2.6.3.1.1.3.1 octetstring \ "named" .1.3.6.1.4.1.2.6.3.1.1.3.2 octetstring "\$hostname" ""  Sends a trap (generic trap 6, specific 31 for enterprise 1.3.6.1.4.2.6.3.1) indicating that named died.
Cause error condition	Error dependent	Disconnect a link on Cisco router to cause a Link Down trap to be generated.
Generate test trap from an SNMP device	Some networking devices provide a method (command line or graphical user interface (GUI)) to generate test traps.	IBM 3494 provides this function through its console.  To send a TESTM SNMP trap, select <b>SNMP Options</b> → <b>Send TESTM Trap</b> . Selecting this menu item opens a window that allows you to enter a string to send to all the monitor stations for which the Library Manager is configured.

Choose the method that is easiest and provides the necessary information. If you have test equipment, causing the error condition to happen may be easiest and supply all the required data for the trap. You may use the **event** command with



NetView traps. However, doing so may not work for simulating unsolicited traps that come from the networking device itself.

The **snmptrap** command is most complex because it requires knowledge of the Management Information Base (MIB) variables passed in the trap, their dotted decimal format, and type (string, number, IP address, counter, etc). If you miss one of the expected variables, the trap is not generated.

### Event reception in IBM Tivoli Enterprise Console

All events received by the IBM Tivoli Enterprise Console server are stored in the IBM Tivoli Enterprise Console reception log, regardless of whether they pass parsing. To query the IBM Tivoli Enterprise Console reception log, use the **wtdump** command. This command has several parameters that you can use to modify the output, which is presented via standard output. When you have a large number of events, it is helpful to use the **-o DESC** option and pipe it to more:

```
wtdump -o DESC | more
```

This variation of the command shows the most recent events first and prevents you from dumping the entire database out to screen at one time. By default, running the **wtdump** command displays all events in the reception log in ascending order.

If you do not see the event that you expect in the output of the **wtdump** command, then the IBM Tivoli Enterprise Console server did not receive the event. From here, it is helpful for you to examine the IBM Tivoli Enterprise Console gateway log files, and the source of the event, to determine why the event was not received by the IBM Tivoli Enterprise Console server.

## 7.3.2 IBM Tivoli Enterprise Console troubleshooting

This section discusses how to solve possible problems when using IBM Tivoli Enterprise Console.

### Gateway and state correlation

You can perform IBM Tivoli Enterprise Console gateway diagnosis and troubleshooting in three different ways:

- ▶ Use the **LogLevel** and **TraceLevel** parameters in the **tec\_gateway.conf** file to debug Java problems.
- ▶ Use the **.tec\_gateway\_diag\_config** file to debug event flow and state correlation.
- ▶ Use external tools to write and test XML rules before production.

To configure debug in the `tec_gateway.conf` file, follow these steps:

1. Edit the `tec_gateway.conf` file to specify the level of tracing that you want. The sample `tec_gateway.conf` file is located in the event server in `$BINDIR/./generic_unix/TME/ACF_REP`.

You can specify the following debug parameters in `tec_gateway.conf`:

```
LogLevel=value
LogFileName=filename
TraceLevel=value
TraceFileName=filename
```

Here *value* is the value of the parameter you want to use, and *filename* is the name of the output file for the debug.

Example 7-6 shows the `tec_gateway.conf` file with all debug options enabled in the four last lines.

*Example 7-6 tec\_gateway.conf file with all debug options enabled*

---

```
#
# cat tec_gateway.conf
# Fri Oct 3 18:33:25 2003
#
# tec_gateway Configuration
#
ServerLocation=@EventServer
GatewaySendInterval=5
GatewayQueueSize=40000
BufEvtPath=/etc/Tivoli/tec/tec_gateway_sce.cache
Pre37Server=no
UseStateCorrelation=YES
StateCorrelationConfigURL=file:///etc/Tivoli/tec/tecroot.xml
SendEventPort=5561
ReceiveAckPort=5562
ReceiveEventPort=5563
SendAckPort=5564
gwr_ActiveConnections=20
gwr_ActiveConnectionsSafety=80
gwr_Enable=yes
LogLevel=ALL
LogFileName=/tmp/tec_gateway.log
TraceLevel=ALL
TraceFileName=/tmp/tec_gateway.trc
#
```

---

2. Distribute a gateway configuration profile with the updated `tec_gateway.conf` through the Adapter Configuration Facility.

To troubleshoot problems with the `tec_gateway` program, you can configure tracing for the `tec_gateway` process, which is controlled by one of these files:

- ▶ UNIX: `/etc/Tivoli/tec/.tec_gateway_diag_config`
- ▶ Windows:  
`%SystemRoot%\system32\drivers\etc\Tivoli\tec_gateway_diag_config`

To configure tracing for the `tec_gateway` process, follow these steps:

1. Edit the `.tec_gateway_diag_config` file to specify the level of tracing that you want. The sample `.tec_gateway_diag_config` file is located on the event server in the `$BINDIR/./generic_unix/TME/ACF_REP` directory. By default, the `.tec_gateway_diag_config` file looks similar to this example:

```
Highest_level error
Truncate_on_restart true
#tec_gateway
#####
tec_gateway Highest_level error
tec_gateway GW_Send_error /tmp/tec_gateway
tec_gateway State_Correlator error /tmp/tec_gateway
```

Both `Highest_level` keywords set the highest trace level possible within the following sections in the file. The tracing levels from least verbose to most verbose are:

```
error
warning
trace0
trace1
trace2
```

The `Truncate_on_restart` keyword specifies whether trace files are truncated to zero bytes when the `tec_gateway` process starts.

2. Distribute the gateway configuration profile with the Adapter Configuration Facility.

**Note:** If you upgrade from a previous release of the Tivoli Enterprise Console product, the Distribution tab for existing gateway configuration profiles is not updated with the `.tec_gateway_diag_config` file. If you want to enable tracing, you must delete the existing gateway configuration profile and create and distribute a new gateway configuration profile.

You should only set tracing on to determine the cause of a problem. Otherwise, disable tracing or set tracing at the error level. If you distribute a gateway configuration profile and you want to disable tracing, delete the `.tec_gateway_diag_config` file from the gateway configuration profile. If you already distributed the `.tec_gateway_diag_config` file and you want to disable

tracing, delete the `.tec_gateway_diag_config` file manually from the Tivoli Enterprise Console gateway.

IBM Tivoli Enterprise Console V3.9 does not come with any tools to edit or test state correlation XML rules. However, many free tools are available for XML editing. We recommended that you use one of these tools to make your work easier. We also recommend that, as always, you test the rules before you go to the production environment.

## Reception buffer

The reception buffer is a first in, first out (FIFO) queue. If you routinely see QUEUED events in the output from the `wtdump` command, the rule engine is too busy. If you see only PROCESSED events in the output from the `wtdump` command, the reception buffer is adequately sized, and rule engine processing is efficient.

When the reception buffer accepts the event, the reception engine process changes the event to the QUEUED state. If you routinely see WAITING events in the output of the `wtdump` command, the reception buffer is not large enough, the rule engine is too busy, or both.

When the event server is restarted, the reception engine is reloaded with events from the reception log that are in the WAITING or QUEUED state.

The reception engine does not process internally generated events (for example, those generated by rules). Internally generated events never appear in the reception log or the reception buffer.

The reception buffer is located in system memory (RAM). You can configure the size of the reception buffer using the `wsetesvrcfg` command or from the Event Server Parameters window.

## Rules cache

The event cache is basically a list of received events in RAM that have been through rule processing. The default size is 1000 events. It is configured with the `wsetesvrcfg` command or from the Event Server Parameters window.

Events are uniquely identified by a number that is a combination of the event attributes `event_handle`, `server_handle`, and `date_reception`, sometimes referred to as an *event ID*.

After rule processing, the event is placed into the event cache of the rule engine. The rule engine evaluates and correlates events with other events in the event cache. The rule engine uses the event cache for its processing, and the event

cache is kept in memory. The rule engine interacts with the dispatch engine to synchronize the updates of its event cache with the event database.

When the event server is started, the dispatch engine retrieves events from the event database to reload the event cache for the rule engine.

When the internal TEC\_Notice event *Rule Cache full: forced cleaning* occurs, five percent of the events are removed from the cache. Events are removed in order by age, with the oldest events removed first.

**Note:** The event cache may have different contents than the event repository or an event console. This is because it is primed with events from the event repository when the IBM Tivoli Enterprise Console server is started.

If this TEC\_Notice event is received, you must either increase the size of the event cache or to reduce the time that events are kept in the event cache. For more information about setting these parameters, see the description of the **wsetesvrcfg** command in the *IBM Tivoli Enterprise Console Command and Task Reference, Version 3.9*, SC32-1232. You can also configure these parameters through the Tivoli Desktop using the Event Server Parameters window.

### ***Event cache searching***

Searching the event cache for related events begins with the most recent event in the cache and progresses backwards to the oldest.

### **Rule base**

When the event server is started, it activates a rule base into memory for rule engine use. The rule base contains all rules and event class definitions that are to be evaluated against events.

### **Measuring event processing performance**

You can measure event processing performance on the event server by generating a report of event arrival and processing during a user-defined sample period. This report includes the overall count of events received and processed. You can also specify a time interval to be used to periodically calculate throughput during the sample period.

Events received by the event server are inserted into the reception log with a state of QUEUED. When processing by the event server is complete, the state of the event in the reception log is updated to PROCESSED.

To create a report of this processing activity, add the following two parameters to the `.tec_config` configuration file:

```
tec_benchmark_report_period=report_period
tec_benchmark_sample_period=sample_period
```

Here *report\_period* is an integer that specifies the rate at which processing rates are printed. *sample\_period* is an integer that specifies the time window for which event arrival and processing rates are computed. After you add these parameters, stop and restart the event server.

If either of these parameters is specified in the configuration file, the output is printed to the `tec_reception` trace file. Example 7-7 shows the output produced with time stamps removed.

---

*Example 7-7 Output with time stamps removed*

---

```
=====
Event Throughput Statistics
=====
Reporting Interval is 2 seconds
Sample Interval is 60 seconds
Actual Period 8 seconds
Events Received:592
Event Arrival Rate:74.000000 events/second
Events Processed:700
Event Processing Rate:87.500000 events/second
-----
Total Events Waiting:0
Total Events Received:6604
Total Events Processed:5666
Processing Backlog:938
=====
```

---

The output is displayed in two sections. The first section displays the following information:

- ▶ The reporting and sample intervals specified in the configuration file
- ▶ The current cumulative time within the sample period
- ▶ The count of events received and the arrival rate
- ▶ The count of events processed and the processing rate (event server processing throughput in events per second)

The second section shows event-related statistics computed since the server was started, including the number of received, processed, and waiting events, as well as the current server back log. The *back log* is the difference between the received and processed events.

# Monitoring and maintaining the IBM Tivoli Enterprise Console logs

IBM Tivoli Enterprise Console uses several database tables to store events, logs, and information about console users and configuration. Some of the tables are created as *database managed*. These tables have a fixed size. When they become full, IBM Tivoli Enterprise Console stops processing events.

The IBM Tivoli Enterprise Console reception log is stored in the `tec_t_evt_rec_log` table. The events in the reception log are deleted from the table when they expire.

The IBM Tivoli Enterprise Console event repository, `tec_t_evt_rep`, stores events that can be viewed on the Event Console. This table is not cleaned up by any default process.

When events are deleted from the IBM DB2 Universal Database tables, the event data is removed. However, the space occupied by the deleted events is *not* released. A DB2 **reorg** command is necessary to free these pages.

DB2 tables are stored in containers called *tablespaces*. One or more tables may be stored in a single tablespace. When setting the size of a DB2 table, you actually set the size of the tablespace that is holding the table.

Perform the following steps to view the state of the tablespaces:

1. Launch a DB2 command line processor.
2. Connect to the `tec` database using the following command using the correct password for the `db2` account:  

```
connect to tec user db2 using password
```
3. Execute the following command:  

```
list tablespaces show detail
```
4. You see a list of all tablespaces in the `tec` database along with vital information about free space. The lab test system returned the data shown in Example 7-8.

Example 7-8 Example tablespace data

Tablespace ID	= 3
Name	= TS_REC_LOG
Type	= Database managed space
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	

Total pages	= 23360
Useable pages	= 23296
Used pages	= <b>160</b>
Free pages	= <b>23136</b>
High water mark (pages)	= 160
Page size (bytes)	= 16384
Extent size (pages)	= 32
Prefetch size (pages)	= 32
Number of containers	= 2
Tablespace ID	= 4
Name	= TS_EVT_REP
Type	= <b>Database managed space</b>
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 30720
Useable pages	= 30688
Used pages	= <b>2112</b>
Free pages	= <b>28576</b>
High water mark (pages)	= 4128
Page size (bytes)	= 16384
Extent size (pages)	= 32
Prefetch size (pages)	= 32
Number of containers	= 1

---

The data highlighted in bold reveals that:

- Both tablespaces are database managed. They will *not* be increased in size when full.
- The Reception Log tablespace is using the default, minimum number of pages (160). The reception log was cleared and the table was recently reorganized.
- The Event Repository tablespace has grown to about 7% of its maximum size. This is not a problem at this point. We allocated 2 GB rather than the default 75 MB for the database. The table will grow and needs regular, scheduled maintenance.

To clear the reception log and the event repository, use the following steps:

1. Open a Tivoli command prompt and run the following command:

```
wtdbc1clear -l -t 0
```

This deletes all events from the reception log database table.

2. Enter the following command:

```
wrmdbc1clear -t 24
```



Assuming that you have events in the database that are more than 24-hours old, the command displays the number of events scheduled for deletion and asks for confirmation. Select y to confirm the deletion.

At this point, you may see an error message. If you selected more than 1,000 events for deletion and you are using a default DB2 transaction log size, the command may fail. If this happens, increase the DB2 transaction log size using the DB2 control center and reboot the event server. We used a transaction log size of about 500 MB and could delete more than 100,000 events using this command.

3. The events are deleted. If you list the tablespaces again, you will find that the pages are not released. Using the DB2 command line processor, enter the following commands:

```
reorg table db2.tec_t_evt_rec_log
reorg table db2.tec_t_evt_rep
```

4. Enter the following DB2 command again:

```
list tablespaces show detail
```

5. The pages for the tablespaces are released as shown in Example 7-9.

*Example 7-9 Tablespace data after database cleanup*

---

Tablespace ID	= 4
Name	= TS_EVT_REP
Type	= Database managed space
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 30720
Useable pages	= 30688
Used pages	= <b>160</b>
Free pages	= 30528
High water mark (pages)	= 4128
Page size (bytes)	= 16384
Extent size (pages)	= 32
Prefetch size (pages)	= 32
Number of containers	= 1

---

The event repository tablespace now displays only 160 pages used.

**Important:** Make sure that you run a database cleanup at regular intervals. If you don't, IBM Tivoli Enterprise Console stops processing events when one or more tablespaces reaches the maximum allocated size. There is no warning in IBM Tivoli Enterprise Console when this occurs.

Creating a task that cleans the reception log and event repository is not entirely straightforward. You must be able to run both DB2 and Tivoli commands sequentially. The batch file shown in Example 7-10 does this. You can improve it if necessary. The DB2 password is supplied in clear. Substitute the word password with the DB2 password.

*Example 7-10 db\_cleanup.cmd*

---

```
REM *****
REM * This batch file will clear the TEC Reception Log, Remove      *
REM * events older than 24 hours from the TEC Event Repository and *
REM * reorganize the database tables to free the pages occupied by *
REM * the deleted events.
REM *****
```

chcp 1252

```
REM *****
REM * Set Tivoli environment - required for Tivoli commands.      *
REM *****
```

@echo off

```
rem Licensed Materials- Property of IBM
rem (C) Copyright IBM Corp. 1996, 2002 All Rights Reserved
rem
rem US Government Users Restricted Rights- Use, duplication,
rem or disclosure restricted by GSA ADP Schedule Contract with
rem IBM Corp.
rem
rem Tivoli Environment Configuration Script
rem
```

```
rem Mon Sep 15 10:44:37 2003
rem generated at TMP install time
```

```
set BINDIR=C:\PROGRA~1\Tivoli\bin\w32-ix86
set DBDIR=C:\PROGRA~1\Tivoli\db\m23x2900.db
set TMRDIR=C:\WINNT\SYSTEM32\DRIVERS\ETC\Tivoli
set o_dispatch=94
set WLOCALHOST=
set INTERP=w32-ix86
```

```

if not "%PERLLIB%" == "" set PERLLIB=%BINDIR%\tools\lib\perl;%PERLLIB%
if "%PERLLIB%" == "" set PERLLIB=%BINDIR%\tools\lib\perl
set TivPath=%BINDIR%\bin;%BINDIR%\tools;%BINDIR%\ADE;%BINDIR%\AEF
set Path=%TivPath%;%Path%
set TMP=%DBDIR%\tmp
set TEMP=%DBDIR%\tmp
set REGI=%DBDIR%\region.out
if exist %REGI% if exist %TMRDIR%\tmrset.txt copy %TMRDIR%\tmrset.txt+%REGI%
%TMRDIR%\tmrset.bat
if exist %TMRDIR%\tmrset.bat call %TMRDIR%\tmrset.bat
set TISDIR=%BINDIR%\..\generic
set NLSPATH=C:\PROGRA~1\Tivoli\msg_cat\%%L\%%N.cat
echo Tivoli environment variables configured.
echo on
cmd /C wtdbclear -l -t 0
cmd /C wrmdbclear -t 24 -f
db2 connect to tec user db2 using password
db2 reorg table db2.tec_t_evt_rec_log
db2 reorg table db2.tec_t_evt_rep
db2 reorg table db2.tec_t_slots_evt

```

---

Save this file as db\_cleanup.cmd.

We pasted the contents of the Tivoli environment batch file, C:\winnt\system32\drivers\etc\Tivoli\setup\_env.cmd, into the batch file. Use the contents of the file on *your* systems to ensure that the path statements are correct.

Finally, you must have a path to the DB2 bin directory, in our case, C:\SQLLIB\bin. Enter the following command:

```
db2cmd db_cleanup.cmd
```

It starts the DB2 command line processor, loads the Tivoli environment, and executes the cleanup commands sequentially.

Use the Windows or Tivoli scheduler to run this batch file at regular intervals. You may want to run it every night to keep IBM Tivoli Enterprise Console running for extended periods with no manual maintenance.

With the base tuning done, the system is prepared to handle a large number of events, stay responsive, and display only the required information. The next section looks at fine tuning performance, setting individual thresholds for event categories, and the possibility of suppressing certain incident group categories.

## Debugging rules with trace directive

In our lab environment, we initially test the `maintenance_mode.rls` rule and do not achieve the expected results. We use IBM Tivoli Enterprise Console rule tracing to determine the cause of the problem.

To enable IBM Tivoli Enterprise Console tracing, first set tracing to *on* for the event server. On the Tivoli desktop, simply right-click the **Event Server** icon and select parameters. On the resulting window (see Figure 7-8), click **Trace Rules**, and then click **Save & Close**. Note the entry for the Rule Trace file. This is where the trace output is written. The entry defaults to `/tmp/rules.trace`.

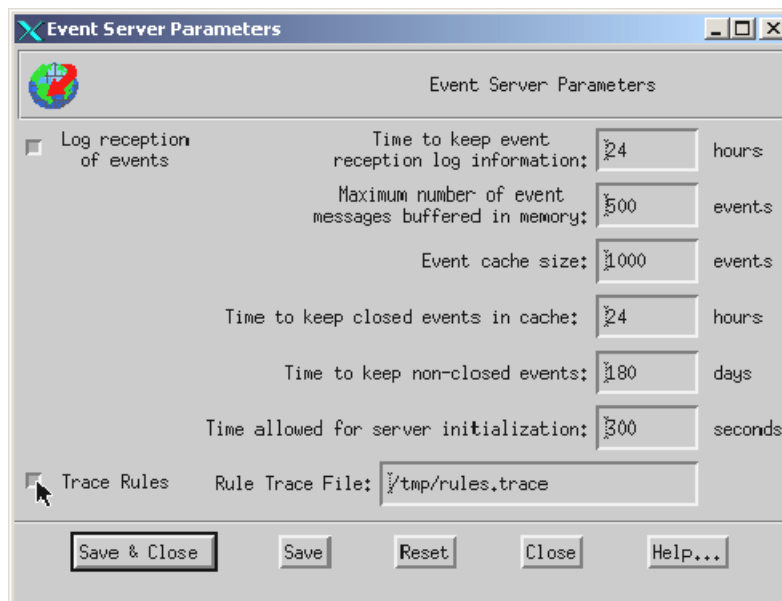


Figure 7-8 Enabling IBM Tivoli Enterprise Console tracing on the Tivoli desktop

You can also do this by entering the following command on a command line:

```
wsetesvrcfg -t tracefile
```

Here the *tracefile* name is optional. If you do not specify it, it defaults to `/tmp/rules.trace`.

Next, add the trace directive to the rule set or rule for which tracing is desired. It is possible to trace all rules. However, the trace output file quickly becomes unreadable due to the large volume of events being processed concurrently. Example 7-11 shows the trace directive that we added to the `maintenance_mode.rls` file. Notice that the entry is placed at the beginning of the rule set before any rules to trace the entire rule set. To trace only one rule within

the rule set, specify the directive as the first entry in the rule, before the event filter definitions.

*Example 7-11 Excerpt from maintenance\_mode.rls showing the trace directive*

---

```
%-----  
%  
% RULESET : maintenance_mode  
%  
%         Tivoli Enterprise Console  
%         Correlation & Automation  
%         IBM Corporation  
%  
% DESCRIPTION  
%  
% This ruleset implements rules supporting the Maintenance function.  
%  
%-----  
directive: trace  
  
%-----  
% RULE: maintenance_mode_configure  
%  
% DESCRIPTION  
%  
% This rule is used to configure the maintenance_mode behavior.  
%-----  
rule: maintenance_mode_configure:  
(  

```

---

Next, recompile, reload, and recycle the IBM Tivoli Enterprise Console rule base using the following commands:

```
wrb -comprules rulebasename  
wrb -loadrb rulebasename  
wstopesvr  
wstartesvr
```

**Note:** Specifying **-trace** on the **wrb -comprules** command enables tracing for the entire rule base, if desired.

The trace entries for the rules traced with the trace directive (or the entire rule base, if the -trace parameter is used on the **wrb -comprules** command) are written to the trace file specified or /tmp/rules.trace by default.

In our example, we place a machine, dtmwas01, in maintenance mode using the script wstartmaint.sh. Next, we generate a node down event for the device. The event does not automatically close as expected.

The check\_maintenance\_mode rule in the rule set detects when a device is in maintenance mode. We hypothesized that it was not working and searched for it in the trace output. Example 7-12 lists an excerpt of the output.

*Example 7-12 Excerpt of IBM Tivoli Enterprise Console rule trace output*

---

```

43] 18:44:30-> rule    check_maintenance_mode
                        event : 0x20a48218 of_class TEC_ITS_NODE_STATUS
[944]          call    condition
[945]          call    fqhostname : _1142
[946]          exit    fqhostname : ''
[947]          call    hostname : _1820
[948]          exit    hostname : dtmwas01
[949]          call    date_reception : _2498
[950]          exit    date_reception : 0x3f8b2ace
[951]          exit    condition
[952] 18:44:30 call    action    check_for_event_server
[953]          call    recorded(mt_event_server, _3810 )
[954]          exit    recorded(mt_event_server,rduatc01)
[955]          call    '' == rduatc01 , recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
; dtmwas01 == rduatc01 , recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
[956]          fail    '' == rduatc01 , recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
; dtmwas01 == rduatc01 , recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
[957] 18:44:30 fail    action    check_for_event_server
[958] 18:44:30 call    action    drop_events_in_maintenance
[959]          call    maintenance('', _3413 , _3414 , _3415 )
[960]          fail    maintenance('', _3413 , _3414 , _3415 )
[961] 18:44:30 fail    action    drop_events_in_maintenance

```

---

The first action in the rule is check\_for\_event\_server. It checks to see if the event is specifying the IBM Tivoli Enterprise Console server as the machine to place in maintenance mode. In our example, it is not. Line [955] shows that the dtmwas01 = rduatc01 comparison fails, so the dtmwas01 maintenance mode host is not the event server rduatc01.

Next, we examine the drop\_events\_in\_maintenance action. This compares the entries in the maintenance fact file (host name, mode, start time, and maximum duration) with the blank, and the values assigned to variables \_3413, \_3414, and \_3415. Since there are no entries in the fact file with a blank full-qualified host name, we determined the problem. The fqhostname slot was not getting

populated for the event. Therefore, it never matched any entries in the maintenance mode fact file.

For more information about using IBM Tivoli Enterprise Console rule traces, see Chapter 6, “Testing, tracing, and profiling rules” in *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9*, SC32-1234.

## Generating sample events

Sometimes it is not possible to create a condition in which an event source sends the event that you want to test. You can use the **wpostemsg** and **postemsg** commands to send test events to the IBM Tivoli Enterprise Console.

In the previous example, we wanted to see if specifying an fqhostname slot variable in the event allows the event to be properly processed by the maintenance\_mode.rls. We issued the following statement:

```
wpostemsg -r CRITICAL -m "Node down" fqhostname=dtmwas01.itso.ral.ibm.com
hostname=dtmwas01 TEC_ITS_SA_STATUS dtmwas01
```

The event arrived in IBM Tivoli Enterprise Console with the fully-qualified host name. It was successfully dropped by the rule, as shown by the last line in Example 7-13.

*Example 7-13 IBM Tivoli Enterprise Console rule trace output successful*

---

```
[1366] 18:55:56-> rule    check_maintenance_mode
          event : 0x20a4ba18 of_class TEC_ITS_SA_STATUS
[1367]      call    condition
[1368]      call    fqhostname : _1142
[1369]      exit    fqhostname : 'dtmwas01.itso.ral.ibm.com'
[1370]      call    hostname : _1820
[1371]      exit    hostname : dtmwas01
[1372]      call    date_reception : _2498
[1373]      exit    date_reception : 0x3f8b2d7c
[1374]      exit    condition
[1375] 18:55:56call    action    check_for_event_server
[1376]      call    recorded(mt_event_server, _3810 )
[1377]      exit    recorded(mt_event_server,rduatc01)
[1378]      call    'dtmwas01.itso.ral.ibm.com' == rduatc01 ,
recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 ) ; dtmwas01 == rduatc01 ,
recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
[1379]      fail    'dtmwas01.itso.ral.ibm.com' == rduatc01 ,
recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 ) ; dtmwas01 == rduatc01 ,
recorded(commit_rule,exec_rule, _4449 ) , cut( _4449 )
[1380] 18:55:56fail    action    check_for_event_server
[1381] 18:55:56call    action    drop_events_in_maintenance
```

```

[1382]      call      maintenance('dtmwas01.itso.ral.ibm.com', _3413 , _3414 , _3415 )
[1383]      exit      maintenance('dtmwas01.itso.ral.ibm.com',ON,0x3f8b2d65,1800)
[1384]      call      0x3f8b2d7c > 0x3f8b2d65 , _3457 is 0x3f8b2d7c - 0x3f8b2d65 , ( _3457
< 1800 , recorded(maintenance_mode,maint_admin, _3477 ) , recorded(maintenance_action, _3483 )
, ( _3483 == CLOSE , set_event_administrator(0x20a4ba18, _3477 ) ,
set_event_status(0x20a4ba18,CLOSED) ; _3483 == DROP , drop_received_event) ,
recorded(commit_set,exec_rule, _3517 ) , cut( _3517 ) ; true) ; true
[1385]      exit      0x3f8b2d7c > 0x3f8b2d65 , 23 is 0x3f8b2d7c - 0x3f8b2d65 , (23 < 1800 ,
recorded(maintenance_mode,maint_admin,'maintenance_mode.rls') ,
recorded(maintenance_action,CLOSE) , (CLOSE == CLOSE ,
set_event_administrator(0x20a4ba18,'maintenance_mode.rls') ,
set_event_status(0x20a4ba18,CLOSED) ; CLOSE == DROP , drop_received_event) ,
recorded(commit_set,exec_rule,39) , cut(39) ; true) ; true
[1386] 18:55:56exit      action      drop_events_in_maintenance

```

---

For more information about using the **wpostmsg** and **postmsg** commands to test IBM Tivoli Enterprise Console rules, see the command syntax in *IBM Tivoli Enterprise Console Command and Task Reference, Version 3.9, SC32-1232*.

## Verifying action taken on events

You may use IBM Tivoli Enterprise Console to determine the action taken upon an event. Through the console, you can see whether an event is closed, its severity is escalated, or its slot variables are set appropriately.

Another way to verify action is to query the event repository table of the IBM Tivoli Enterprise Console database for the event. The **wtdumper** command, supplied by IBM Tivoli Enterprise Console, easily lists the contents of the table. The output can be limited to events occurring within a specific time frame and may be ordered in ascending or descending sequence. The default action is for events to be listed in the order in which they occurred. Example 7-14 shows the syntax of the command.

### Example 7-14 wtdumper syntax

---

```

wtdumper [-f file] [-t start_time] [-e end_time] [-o ASC | DESC] [-m number]
[-d] [-w where_clause]

```

---

Note the following explanation for this command:

- d** Lists detailed formatted information in the event report.
- e end\_time** Lists events that occurred prior to the specified date and time. *end\_time* is a date in the format Mon dd hh:mm:ss yyyy. If this flag is omitted, the command uses the current time for the end time.



- f file** Writes output to the specified file.
- m number** Specifies the maximum number of events to record in the report. If the number of events in the database exceeds the specified value, the command omits entries from the end of the report. For example, if the report is displayed in ascending order, the most recent database entries are not included in the report.
- o ASC | DESC** Sets the order in which events are listed to ascending or descending respectively.
- t start\_time** Lists events that occurred after the specified date and time. The start\_time parameter must be a date in the format Mon dd hh:mm:ss yyyy.
- w where\_clause** Specifies a partial SQL WHERE clause for the event database query. This clause is appended to the internally generated WHERE clause with the AND operator. This option is useful if you are experienced with SQL statements.

**Tip:** If **wtddumper** is run from a node other than the Tivoli Enterprise Console server, it uses the time from the local system to determine which events to display. This may cause unexpected behavior. For example, if the time on the node is 9:00 and the Tivoli Enterprise Console server is 9:30, a **wtddumper** command run from the node displays every event in the database, except for those occurring during the 30 minutes specified. The same command run on the Tivoli Enterprise Console server displays the entire database.

In our example, it seems as though the maintenance\_mode rule closed the Node down event for dtmwas01 as expected. We can verify this by querying the IBM Tivoli Enterprise Console event repository for the event and checking its status. We issue the following command:

```
wtddumper -d
```

The -d flag was added to make the output more readable and to display more of the event's slot variable. The applicable event from the output is included in Example 7-15. Notice that the status is CLOSED and the administrator is maintenance\_mode.rls. This indicates that the maintenance mode rule closed this event.

*Example 7-15 Excerpt from wtdumper output*

---

```
TEC_ITS_SA_STATUS;  
    server_handle=1;  
    date_reception=1066085756;  
    event_handle=1;  
    source=dtmwas01;  
    sub_source=N/A;  
    origin=9.24.106.185;  
    sub_origin='';  
    hostname=dtmwas01;  
    adapter_host='';  
    status=CLOSED;  
    administrator=maintenance_mode.rls;  
    acl=[ admin];  
    severity=CRITICAL;  
    date='Oct 13 18:55:56 2003';  
    duration=0;  
    msg='Node down';  
    msg_catalog='';  
    msg_index=0;  
    num_actions=0;  
    credibility=1;  
    repeat_count=0;  
    cause_date_reception=0;  
    cause_event_handle=0;  
    category=undefined;  
    fqhostname=dtmwas01.itso.ral.ibm.com;  
    nv_generic=0;  
    nv_specific=0;  
    sastatus=ifDown;  
  
END
```

---

For more information about **wtdumper**, see *IBM Tivoli Enterprise Console Command and Task Reference, Version 3.9, SC32-1232*.

### 7.3.3 NetView

This section discusses debugging state correlation and rule sets.

#### Debugging state correlation

NetView's use of state correlation is defined in the `/usr/OV/conf/tecint.conf` file. Example 7-16 shows the relevant entries.

First, state correlation is activated by specifying `UseStateCorrelation=YES` in the `tecint.conf` file. The `StateCorrelationConfigURL` specifies a file on the NetView

machine that contains the XML state correlation rules. Refer to Example 7-16 for a sample of the `tecint.conf` file.

*Example 7-16 Entries in `tecint.conf` for state correlation*

---

```
ServerLocation=nswin11
TecRuleName=TEC_ITS.rs
ServerPort=5529
DefaultEventClass=TEC_ITS_BASE
BufferEvents=YES
UseStateCorrelation=YES
StateCorrelationConfigURL=file:///usr/OV/conf/nvsbcrule.xml
# The following four lines are for debugging the state correlation engine
# LogLevel=ALL
# TraceLevel=ALL
# LogFileName=/usr/OV/log/adptlog.out
# TraceFileName=/usr/OV/log/adpttrc.out
```

---

NetView traps in Example 7-16 are subject to the XML rules specified in the file `/usr/OV/conf/nvsbcrule.xml`, which resides on the NetView server. If these rules do not work as expected, enable tracing by uncommenting (or adding, if they are not in your file) the lines specifying `LogLevel`, `TraceLevel`, and their corresponding file names. Check the listed files for relevant debugging information.

## Debugging NetView rules

Events and traps provide information about changes in the status of network elements and alert the NetView program to occurrences in the network. When events and traps are received, they are acted upon in the manner defined in the NetView rule sets. These rule sets perform the major functions of interest, including correlation, notification (including paging and sometimes trouble ticketing), and automated actions.

The event and trap processing daemons described in this section—`nvcorr`, `actionsvr`, and `ovactiond`—execute those functions on behalf of NetView. If you achieve unexpected results, trace these daemons to find the cause.

The logs and trace files for these and other NetView daemons are stored in `/usr/OV/log` (UNIX) or `\usr\OV\log` (Windows). Check this directory for logs that are applicable to problems that you are debugging.

### ***nvcorr daemon***

The `nvcorr` daemon executes rule sets in the foreground, in dynamic work spaces running in the Events window, or in the background, having been loaded

in ESE.automation. The nvcorrd daemon executes all nodes in a rule set, except for Action and Paging nodes.

By default, nvcorrd logs its activities to /usr/OV/log/nvcorrd.alog and nvcorrd.blog. First, nvcorrd.alog is written. When this file becomes full, it is moved to nvcorrd.blog, and a new nvcorrd.alog is started. Consult these files to determine the event processing activities performed by nvcorrd.

**Note:** You can change the log file name by using the -l parameter of the **nvcorrd** command.

The Windows version of nvcorrd is called *nvcord*. Nvcord registers for trap callback using the standard OVW application programming interface (API) and is given a copy of every trap received by the system. It processes these traps according to the specific rule sets that are active at the time and determines if the rule passes or fails.

Every time a rule is activated, nvcord adds the ruleset name and a rulesetID to a table in the event database (unless the rule is already present). This provides a database of all ruleset names.

**Note:** It is assumed that most traps do not pass correlation. If a trap passes a rule, then nvcord creates a new trap based on the last trap processed by the rule. This trap is given a source value that is associated with the active rule. For example, if there is a rule set where the correlation passes if a node-down event is received for a node that is a member of smartset routers, and such a node-down trap arrives, nvcord creates a new node-down trap. The two traps are identical, except that the first trap has Source==Netmon and the second trap has Source==RuleSetID.

### **actionsvr daemon**

The actionsvr daemon executes the Action and Paging nodes in rule sets (nvcorrd does inline actions). Upon startup, actionsvr also loads the rule sets listed in /usr/OV/conf/ESE.automation for nvcorrd to execute in the background.

When an action is to be processed, the actionsvr daemon starts a child process to execute the action. The event is passed to the next node in the rule set. When Tivoli NetView is started, the actionsvr daemon checks the rule sets for automatic action in the /usr/OV/conf/ESE.automation file.

The actionsvr daemon makes all data in the trap available as environment variables. If the action returns a nonzero return code, the actionsvr daemon generates a failed action trap.

By default, actionsvr logs its activities to /usr/OV/log/nvactiond.alog and nvactiond.blog. First, nvactiond.alog is written. When it the file becomes full, it is moved to nvactiond.blog, and a new nvactiond.alog is started. Consult these files to determine the event processing activities performed by nvactiond. They list the ESE.automation rule sets loaded at started, trap variable sanitation, and identify the automated actions that actionsvr performs.

**Note:** You can change the log file name by using the -l parameter of the **actionsvr** command.

### ***ovactiond daemon***

This daemon executes a shell command upon receipt of an event. The ovactiond daemon is configured by selecting **Options** → **Event Configuration: SNMP**. The ovactiond daemon listens to trapd for predefined events. Then it formats and passes a string to the shell for interpretation and execution.

You start the ovactiond daemon using the **ovstart** command without any options. You can change the starting options by editing the ovactiond.lrf file in /usr/OV/lrf.

The ovactiond daemon logs its activities to the /usr/OV/log/ovactiond.log file. Again, you can use the -l parameter on the ovactiond statement to change the log file to which ovactiond logs its activities. Similarly, you can use the -v flag to make the output to the log file more verbose for enhanced debugging information. To trace the execution of ovactiond, use the -t flag.

### ***Rule sets***

Rule sets are criteria applied to the event flow in NetView by the correlation daemon, nvcorr, to perform automatic action when the selected events occur. Rule sets are similar to filters, in that you can use them to alter the operator display. They are similar to automatic actions (defined in trapd.conf and executed by ovactiond) in that they can trigger processes to run in the background.

### ***Ruleset editor***

The ruleset editor is a tool for creating rule sets using a graphical display rather than a line-oriented text editor. This shields the user from the underlying complexities of ruleset syntax.

You activate rule sets either by creating a dynamic work space in the events window or by adding them into the ESE.automation file. The ESE.automation file is loaded when the actionsvr daemon starts. This daemon must be recycled if additional rule sets are added to ESE.automation after the NetView GUI starts.

All rule sets are kept in /usr/OV/conf/rulesets by default and have the suffix .rs appended to them. You must have root authority to edit a rule set. When an

existing rule set is opened for edit, a backup copy of the original is made in the rulesets directory. For information about rule sets and using the ruleset editor, see the *Tivoli NetView for UNIX Administrator's Guide, Version 7.1*, SC31-8892.

### ***Ruleset creation problems***

If you have a problem creating a rule set (for example, a problem with the ruleset editor), perform these steps before you contact Tivoli Customer Support:

1. Look for error messages in the ruleset editor logs `/usr/OV/log/nvrsEdit.alog` and `nvrsEdit.blog`.
2. Get a copy of the rule set from the `/usr/OV/conf/rulesets` file and any files with the same name but with different suffixes such as:
  - BAK, which is the backup copy
  - Meaningless letters on the end, which are temporary copies used to hold changes while the editor is open
3. Look for core files, especially the `/usr/OV/conf/rulesets` file and the root directory. Save the core file in another directory for customer support when you call to report the problem. Customer support may have you run debugging commands on the file.

### ***Ruleset directory problems***

You may find that the ruleset directory, `/usr/OV/conf/rulesets`, fills up with strange files, which look like your rule sets, but have different suffixes. The files with the suffix `.bak` are backup copies of existing rule sets opened for edit. You can delete these `.bak` files after the original is filed safely away. The files with suffixes, such as `IQ8Ekr` and `P%wKiw` or other such combinations, are temporary copies that are made while updates are performed.

You delete these files if you exit the editor by selecting **File** → **Exit**. However, they may remain if you only close the window when you are finished. You can delete these files (unless you need them for problem determination) or the editor deletes them the next time you edit that rule again and close the editor properly.

### ***Tracing***

After `nvccorrd` is running, enter the following command:

```
nvccdebug -d all
```

Generate a test event or wait for the real one to occur, whichever is better. Test events can be generated with the event command (for selected netmon events) or the `snmptrap` command (for any events). To enter these commands from the Tivoli desktop, select **Diagnose** → **Send SNMP trap to a node** or **Diagnose** → **Send event to trapd daemon**.

### ***Hanging or halted events***

Events can be suspended or stopped if rule sets that are designed to send events to a display window are run in the background. This is because actionsvr has no way to delete events that are sent to it, unless there is an action to run and it has no display capability. Therefore, the events build up on the receive socket until it is full. Then, they back up on nvcorrd sending socket until it becomes full. 32767 is the limit of messages on a socket. This backup causes nvcorrd to stop suspending events.

Therefore, rule sets that run in the background, out of ESE.automation, must not contain a forward node or have the default action (on the beginning node) change from block to pass. In the short run, if this problem occurs, to clear the problem until the rule sets can be corrected, enter:

```
ovstop actionsvr
```

### ***Using filtered dynamic work space***

As mentioned in Chapter 6, “Event management products and best practices” on page 173, one method to debug rule sets is to use filtered dynamic work spaces. Select **Create** → **Dynamic Workspace** in the main work space. Then enter the ruleset name of the rule set that you are testing, along with any other appropriate information. The new work space uses the rule set and filters, if any, to determine which events are displayed and correlated.

Generate a test trap using one of the methods described in “Generating test traps” on page 376, or wait for the events to occur naturally. View the main work space to verify that the events are received and view the filtered work space to observe the results. If the rule set does not perform as expected, modify the rule and reopen the dynamic work space to quickly test again.

### **netview.rls rule set**

You can load and activate the netview.rls rule set supplied with NetView in the IBM Tivoli Enterprise Console server. Use the debug capability within IBM Tivoli Enterprise Console to debug this rule set. See “Debugging rules with trace directive” on page 388 for more information.

## **7.3.4 IBM Tivoli Switch Analyzer**

When troubleshooting IBM Tivoli Switch Analyzer, download a copy of the *IBM Tivoli Switch Analyzer Troubleshooting Guide, Version 1.0*, from the Web at:

<http://www.ibm.com/software/support>

This is an official Tivoli Field Guide and was written by Michael L. Webb and Terri Peterson. This troubleshooting guide for IBM Tivoli Switch Analyzer is an

indispensable document to have when trying to troubleshoot IBM Tivoli Switch Analyzer. It covers a wide range of issues that you may experience and explains how to resolve those issues.

An issue in our labs is in regard to moving a device from one port to another. In this case, IBM Tivoli Switch Analyzer reports the related device as marginal until the next layer 2 rediscovery. By default, IBM Tivoli Switch Analyzer rediscovers the layer 2 environment once a day.

In networks with a reasonable amount of movements between ports, adjust the default rediscovery rate by modifying the `discovery_interval` in `/usr/OV/ITLS2/conf/topo_server.ini`. The default is once a day (1440 minutes). Refer to Example 7-17 for details.

*Example 7-17 /usr/OV/ITSL2/conf/topo\_server.ini*

---

```
[Server]
log_file=../log/topo_server.log
log_size=1000
msg_lvl=5
timeout=15

[Layer2]
base_topo_oid=50000000
req_cnt=10
retry_interval=900
retry_cnt=3
discovery_interval=1440
mac_timeout=15
debug=0
log_file=../log/L2_data.log
log_size=20000

[SNMP]
req_cnt=20
timeout=5000
retry_cnt=5

[Correlator]
host=localhost
port=31100
timeout=30
```

---





## Suggested NetView configuration

This appendix contains information, which you may find helpful when working on NetView configurations. These are not necessarily best practices. Instead, they are configurations used by the authors of this IBM Redbook throughout their workings with the NetView product.

## Suggested NetView EUI configuration

In most cases, you access the NetView EUI via a normal workstation. In this case, most administrators prefer to have the main NetView window, the topology, displayed. You should minimize the tools and navigation window. You should also disconnect the event console to have it independent from the main window. To achieve this, you need to modify the appropriate configuration files after you make backup copies of the files:

1. Open `/usr/OV/app-defaults/OVW` with your editor. In this file, search for the string **CopyRightWindow** and change its attribute to *False*.
2. In the same file, search for **strings toolShellIconify** and **navTreeShellIconify**. Set both attributes to *True*.
3. In the file, locate the strings **shellwidth** and **shellHeight**. The assigned number are the dimensions of the NetView main window. Adjust them to your needs.

Example A-1 shows the changes that we made for the NetView layout in our lab environment. This finishes the EUI modification.

*Example: A-1 Changes in /usr/OV/app-defaults/OVw*

---

```
!*****
!  
! Tired of clicking in OK button for Copyright information window ?  
! Then change this to False.  
!  
!*****  
  
*displayCopyRightWindow:                False  
.  
.  
!  
! Defines the EUI shell x and y coordinates used in the creation. Not used  
! when the creation is related with a drag/drop operation. If this resource  
! is not set ( omitted ) the shells are open in cascade mode.  
! The unit is number of pixels.  
!  
OVw*shellX:                             0  
OVw*shellY:                             0  
!  
!  
! Defines the EUI shell width and height to be used in the creation.  
! The unit is number of pixels.  
!  
OVw*shellWidth:                         800  
OVw*shellHeight:                       650
```

```

.
.
!
! Determines whether the EUI ToolPalette shell is created iconified or not
!
OVw*toolShellIconify:          True
.
!
! Determines whether the EUI NavTree shell is created iconified or not
!
OVw*navTreeShellIconify:      True

```

---

After you make these changes, the EUI presents a navigation bar and a toolbar that is minimized or appears as an icon. The event console still appears attached to the main EUI window. The following modification causes the event console to be detached from the main EUI window. In addition, the modification resizes the event console and changes the initial display of events from card to list mode. It also makes sure that any dynamic work spaces that you create are displayed as disconnected from the main EUI window.

## Event console configuration

Open the `/usr/OV/app-defaults/Nvevents` file and modify the following entries:

1. Change the `nvevents.initialPresCard` field to *False*.
2. Adjust the width and height of the event window by modifying the `nvevents.widthMain` and `nvevents.heightMain` to a dimension, which fits into your screen layout. The assignments for the two fields are measured in pixels.
3. Set the field `nvevents.outside` to *True*. This ensures that the event console starts outside the main NetView window.
4. Change the field `nvevents.wsOutside` to *True*. Any dynamic work spaces that you open are created outside the main window.

Optionally, you can set the entries `nvevents.loadEnvOnInit` and `nvevents.saveEnvOnExit` to *True* in case you want to save and reload the layout and possible dynamic work spaces during NetView initialization.

Example A-2 summarizes the changes.

*Example: A-2 Changes in /usr/OV/app-defaults/*

---

```
! defines initial presentation style (card or list)
!
nvevents.initialPresCard      : False
.
.
! size of nvevents windows
nvevents.widthMain           : 500
nvevents.heightMain          : 300
.
.
! defines if application starts up outside of the control desk
! valid when running integrated to OVw
!
nvevents.outside              : True
! defines if new workspaces are opened outside the control desk
!
nvevents.wsOutside            : True
!
.
.
! defines if nvevents loads an existing environment file
!
nvevents.loadEnvOnInit : True
! defines if nvevents saves all workspaces during exit process
!
nvevents.saveEnvOnExit : True
```

---

## Web console installation

You can access the NetView Web console via a stand-alone Java application or via a Web browser using a Java applet. The following sections outline both methods for accessing the Web console.

### Web console stand-alone installation

We show the installation of the Web console for a Windows desktop. To bring the console to other platforms, follow the same steps, but select the correct distribution. Remember to specify at least one user account to the Web console using the NetView native EUI as discussed in “Web console security” on page 407.

1. From a Web browser, access the download page of your NetView server by typing:

`http://your_netview_address:8080/download`

Here *your\_netview\_address* is the name or IP address of the NetView server. A Web page opens similar to the one shown in Figure A-1. For a Windows platform, you need to download the `nvwinstall.exe` package, which contains all necessary code to run the Web console.

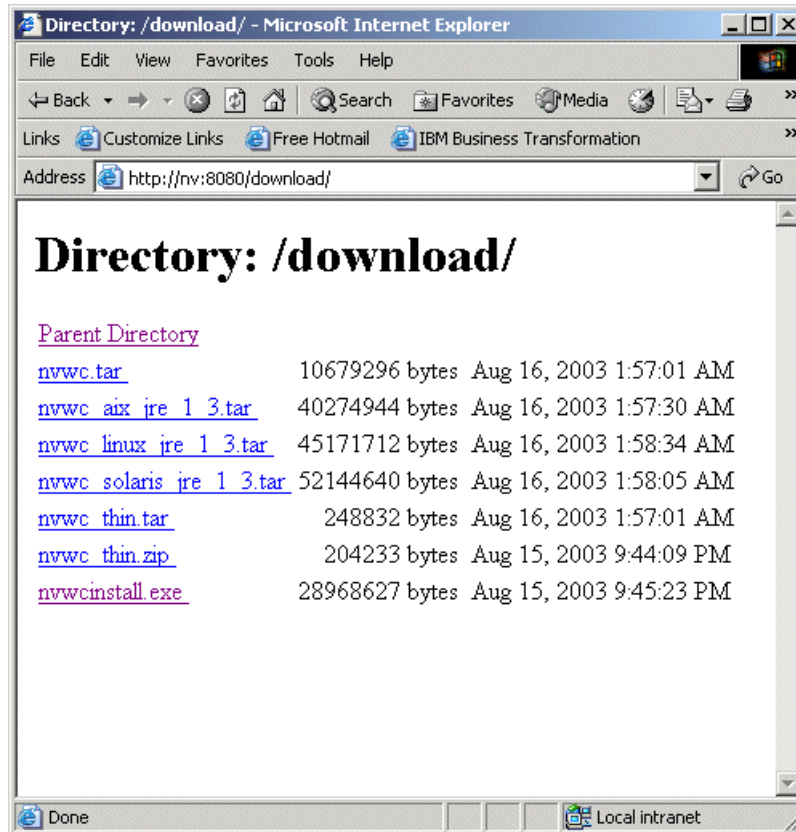


Figure A-1 Web console download page

2. Locate the package that you downloaded. Double-click it to start the installation. A normal Installshield operation is started.
3. When you reach the dialog asking for the install location, we suggest that you replace the default suggestion and change it to a short path, which contains no spaces and follows the ancient 8.3 convention like `c:\nwc` as shown in Figure A-2. If you plan to integrate the Web console into other applications, an 8.3 path name is often required. Our experience shows some problems to pass parameters containing long pathnames or spaces to an application.  
After you change the path name of the installation path, the Web console installs without requiring further intervention.

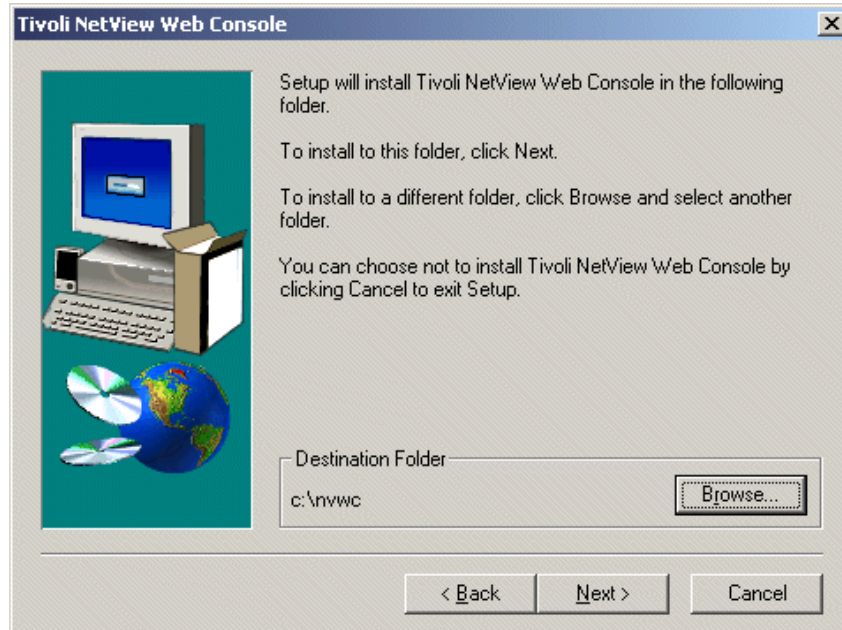


Figure A-2 Changing the default path

After a successful installation, you see an icon that represents the Web console on your Windows desktop.

## Web console applet

As an alternative, you can access the Web console via a standard Web browser. A Java plug-in for your browser is required, which you download from the Internet on demand. Therefore, you may need a working Internet connection when you first launch the Web console via the browser. To start the Web console, follow these steps:

1. From a Web browser, access the NetView applet provided by the NetView Web server by typing:  
`http://your_netview_address:8080/netview/NetViewApplet`  
Here *your\_netview\_address* is the name or IP address of the NetView server.
2. If the required Java plug-in is not present, you see the Security Warning panel (Figure A-3). In this case, you need to download the plug-in from the Internet.



Figure A-3 Downloading the Java plug-in

3. Follow the installation instructions for the plug-in. After a successful installation, the applet starts downloading the required classes and resources from the NetView server. You can observe the loading of these components in the browser window. Finally, a Web console identical to the stand-alone version appears in a Java applet window.

## Web console security

Before you launch the Web console, you must create at least one user using the native NetView EUI using the following steps:

1. Select **Administer** → **Security Administration** → **Web Console Security**.
2. The Web Console Security window (Figure A-4) opens. In the left pane, select **Users**. Then, from the menu, select **Selected** → **Add**. In the right pane, enter a user name and password. Select a role for the user. For the first available user, you should choose **SuperUser** and **No scoping restrictions** to gain access to all Web console features.
3. When you are finished, select **File** → **Save** to save the new created user.

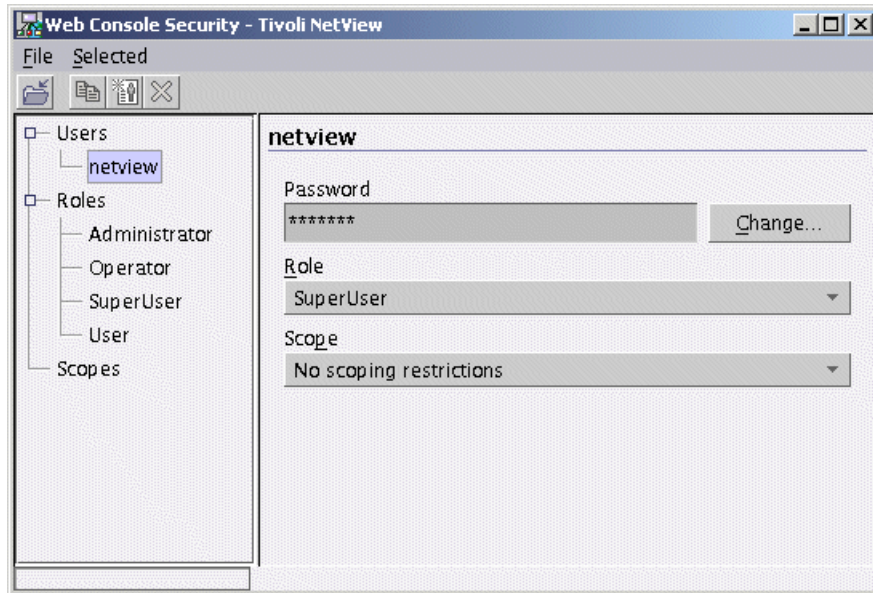


Figure A-4 Web Console Security window

4. A message box is displayed as shown in Figure A-5. Select **Yes** to restart the Web server.

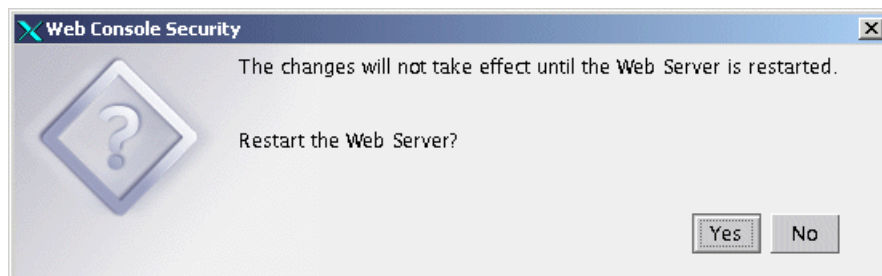


Figure A-5 The restart/save window

This completes the initial security configuration of the Web console.

## Web console menu extension

Starting with NetView Version 7.1, the NetView Web console is meant to be the main interface to NetView. Use the native console only for administrative purposes such as modifying maps and NetView working options.



Tivoli Enterprise Console also launches the NetView Web console on demand to show NetView-related topology information, diagnostic tools, and object properties.

The NetView Web console offers limited extension capabilities. You can extend the menus and execute commands as long as you can display the output in a Web browser window. The supplied Web console functions and menus are defined in two files:

- ▶ `/usr/OV/www/webapps/netview/warf/Actions.xml`  
This file contains all the actions and functions provided by the Web console in a compressed format.
- ▶ `/usr/OV/www/webapps/netview/warf/Templates/WebConsole/MenuBar.xml`  
This file contains the menu definitions for the actions and functions provided by the Web console.

**Note:** While working with NetView 7.1.4, we found more custom definition files in the warf directory. All of these files contain compressed definition calling internal Java and JavaScript functions. All these functions are not documented, and there is no intention to release any documentation.

You can modify these two files, but in the event of a NetView update or patch apply, the update may overwrite them. To prevent this, you can supply your own action definition file and your own menu file, which are not overwritten. We provide the extensions in separate definition files.

**warf subdirectory in the file path:** The role and behavior of the NetView Web console definition files are similar to NetView Application Registration Files (ARF). They are called *Web Application Registration Files* (WARF) and are located in the path's warf subdirectory. The main difference between WARFs and ARF-type registration files is the format. Unlike standard NetView registration files and their descriptive C-style format, the WARFs use Extensible Markup Language (XML) as their description language.

Each distinctive function of the NetView Web console consists of two definitions:

- ▶ An action definition, which you must store under  
`/usr/OV/www/webapps/netview/warf`: The action definition defines what you want executed when selected.
- ▶ A menu definition that you must store under  
`/usr/OV/www/webapps/netview/warf/Templates/WebConsole`: This definition specifies the position of your new menu under the menu tree.

With this information, we show all the steps required to extend the Web console with a new menu. This enables us to view the contents of the netmon seed file from the Web console. You can use this example as a template for your own extensions of the Web console. Example A-3 lists the action definition, and Example A-5 contains the Menu definition.

You also may refer to the “Web Console Enhancements” section in the *IBM Tivoli NetView for Windows Release Notes, Version 7.1.4, SC32-1239*, for additional information about action and menu definitions.

---

*Example: A-3 MyActions.xml*

---

```
<!--<!DOCTYPE WARF SYSTEM 'WARF.dtd'>-->
<WARF xmlns:xlink="http://www.w3.org/1999/xlink">
  <Meta name="Version" value="2.0" />
  <Meta name="Written" value="2002 ITS0 Austin/Glasi GA24-6610,
    modified Glasi sep-2003 Raleigh" />
  <Application name="my-webconsole">
    <!-- <Action id="viewseed" securityConstraint="RelaxedAccess" roles="SuperUser,User"> -->
    <Action id="viewseed" roles="SuperUser,User">
      <Name>View_Seed</Name>
      <Mnemonic>U</Mnemonic>
      <ShortDescription>View seedfile</ShortDescription>
      <LongDescription>Display the seedfile in a Browser window</LongDescription>
      <ActionHandler name="LaunchServerAppHandler">
        <Method>
          <MethodName>com.tivoli.netview.client.NetViewApplet.launchServerApp</MethodName>
          <ArgList>
            <Val>
              <Array>
                <Val>Glaskey1</Val>
                <Val>/usr/OV/bin/viewseed.ksh</Val>
              </Array>
            </Val>
          </ArgList>
        </Method>
      </ActionHandler>
    </Action>
  </Application>
</WARF>
```

---

Note the following explanation:

**<?xml ...>**

You must not change the first three entries in a WARF, especially the first line that specifies the encoding. Be sure it specifies UTF-8 as the character set.

<b>&lt;Application&gt;</b>	Defines the name of the application. You can assign any unique name to your application. Note that <i>webconsole</i> is a reserved name used in the main action definition.
<b>&lt;Action id=&gt;</b>	This keyword specifies a unique ID, which you reference in your menu definitions. You may apply all the roles you want to allow to perform the action, but leave the <i>securityConstraint=</i> entry unchanged.
<b>&lt;Name&gt;</b>	Defines a name for the action.
<b>&lt;Mnemonic&gt;</b>	Defines a menu shortcut for your action. Be sure the mnemonic is specified from the characters used in the name definition.
<b>&lt;ShortDescription&gt;</b>	Specify a short description of your new menu. It appears in the role window of the NetView Security Console, as shown in Figure A-6 on page 415.
<b>&lt;LongDescription&gt;</b>	Defines a more specific explanation of your new menu. Also appears in the roles window.
<b>&lt;ActionHandler&gt;</b>	Specifies the handler meant to execute the action. Don't change the name <i>LaunchServerAppHandler</i> . This handler executes your action on the NetView server.
<b>&lt;Method&gt;</b>	Starts a method subsection in which you define the program to be executed, along with all arguments it may require.
<b>&lt;MethodName&gt;</b>	Specifies the method assigned to the application handler. The full path of the method is required. In our case, it is <i>com.tivoli.netview.client.NetViewApplet.launchServerApp</i> .
<b>&lt;ArgList&gt;</b>	Starts an argument list subsection. Each argument is defined inside a <i>&lt;Val&gt;&lt;/Val&gt;</i> definition.
<b>&lt;Array&gt;</b>	Our action, issuing a bulk upload to IBM Tivoli Business Systems Manager, is actually implemented as a Korn shell script. Parameters to a shell script are passed in the form of an array. Although our bulk upload script does not require any parameters, you must supply the parameter array. The first value in the array acts as a key, which must be unique for all actions defined to execute under a given action handler. In our case, the key must be unique for all actions executed under <i>LaunchServerAppHandler</i> .

The second value in the array must be the program name. A full path is required. Therefore, our array is:

```
<Array>
<Val>Glaskey1</Val>
<Val>/usr/OV/bin/viewseed.ksh</Val>
</Array>.
```

You can place multiple action definitions into one single file. Each definition requires the full set of keywords as discussed.

The key definitions for the new menu entry are the Actionhandler, Method, and ArgList tags in the definition:

- ▶ **Actionhandler tag:** This tag defines the handler to be started on the NetView Web server. Do not change this reference.
- ▶ **Method tag:** This tag defines the Java method being launched on the Web server and the program that the method must execute along with all the required arguments. You should not change the method reference.
- ▶ **ArgList tag:** This tag specifies the arguments to the method. In our case, we pass the name of a small wrapper script placed in /usr/OV/bin as in Example A-4. The script executes a UNIX cat of the netmon seed file. You can specify the command directly in the ArgList definition. However, the indirect call makes it easier to change the behavior of the command without reloading the Action and Menu definitions.

Note the way that the script is defined in the ArgList. The Actionhandler requires arguments passed as key/value pairs. The key can be any unique name, while the value is the full path reference to the program being executed.

---

*Example: A-4 The wrapper script*

```
!/bin/ksh
cat /usr/OV/conf/netmon.seed
exit
```

---

The second set of definitions required is the menu entries. As mentioned previously in this section, menu definitions are stored under /usr/OV/www/webapps/netview/warf/Templates/WebConsole. Example A-5 shows the menu definition.

---

*Example: A-5 MyMenu.xml*

```
<?xml version='1.0' encoding='UTF-8'?>
<!--<!DOCTYPE WARF SYSTEM '../WARF.dtd'>-->
<WARF xmlns:xlink='http://www.w3.org/1999/xlink'>
```

```

<Meta name="Written" value="2002 ITSO Austin/Glasi GA24-6610,modified Raleigh Sep-2003
24-6094" />
<Application name='webconsole'>
<!--      MenuBar name cannot be 'MenuBar', which is reserved by the default MenuBar.xml
-->
<MenuBar name="MyMenuBar">
  <Menu name="tools">
    <Separator/>
    <Menu> <Name>MyTools</Name>
      <MenuItem name="View seedfile">
        <ActionRef xlink:href='MyActions.xml#xpointer(id("viewseed"))' />
      </MenuItem>
    </Menu>
  </Menu>
</MenuBar>
</Application>
</WARF>

```

---

Note the following explanation:

<b>&lt;?xml ...&gt;</b>	This is the same as for the action definition. Don't change the first three lines, and be sure the character set encoding is set to UTF-8.
<b>&lt;Application&gt;</b>	Use the same name as in your action definition.
<b>&lt;MenuBar&gt;</b>	Starts a menu bar definition. Apply the same rules as for the application name. The name must be unique. You can't give it the name <i>MenuBar</i> because this name is already used for the default menu bar of the Web console.
<b>&lt;Menu&gt;</b>	Specifies an existing menu where our submenu should be placed. The NetView Web console already features a tools menu such as the native GUI. Therefore, we specify name="tools" as the menu entry where our new menu resides.
<b>&lt;Separator/&gt;</b>	Causes a separator line to be inserted into the menu. You can use it anywhere inside a menu definition.
<b>&lt;Menu&gt;</b>	The second menu keyword specifies the name of the menu being placed under tools. We gave it the name "MyTools".
<b>&lt;Mnemonic&gt;</b>	Again, this specifies a shortcut to the menu name.
<b>&lt;Name&gt;</b>	As mentioned, our new menu should appear as TBSM in the tools submenu.
<b>&lt;MenuItem&gt;</b>	The MenuItem keyword specifies the entry in the menu tree that triggers the action. We gave it the name "View

seedfile". You can specify additional extensions as new menu items under the Mytools submenu.

#### <ActionRef>

This entry is the most important keyword in our menu definition because it defines the connection to the action definition. Remember that we named our action "launchbu" and it was stored in the MyActions.xml file. This information forms the Actionref attribute `xlink:href='MyActions.xml#xpointer(id("launchbu"))'`.

To activate the new menu, launch Web Console Security. At the time this redbook was written, there was no other way to activate Web console extensions.

Launch the security console using one of these methods:

- ▶ Using the NetView GUI: Select **Administer** → **Security Administration** → **Web Console Security**
- ▶ Using nvsetup: Select **Configure** → **Configure Web Server** → **Configure Web Console Security**
- ▶ Directly issue `/usr/0V/bin/launch_securityconsole`

In the console, select **Roles** and then the role where you want to activate the new entry. You should see your newly created menu in the list of menus, as in Figure A-6. Be sure to mark the new entry as active for the wanted role.

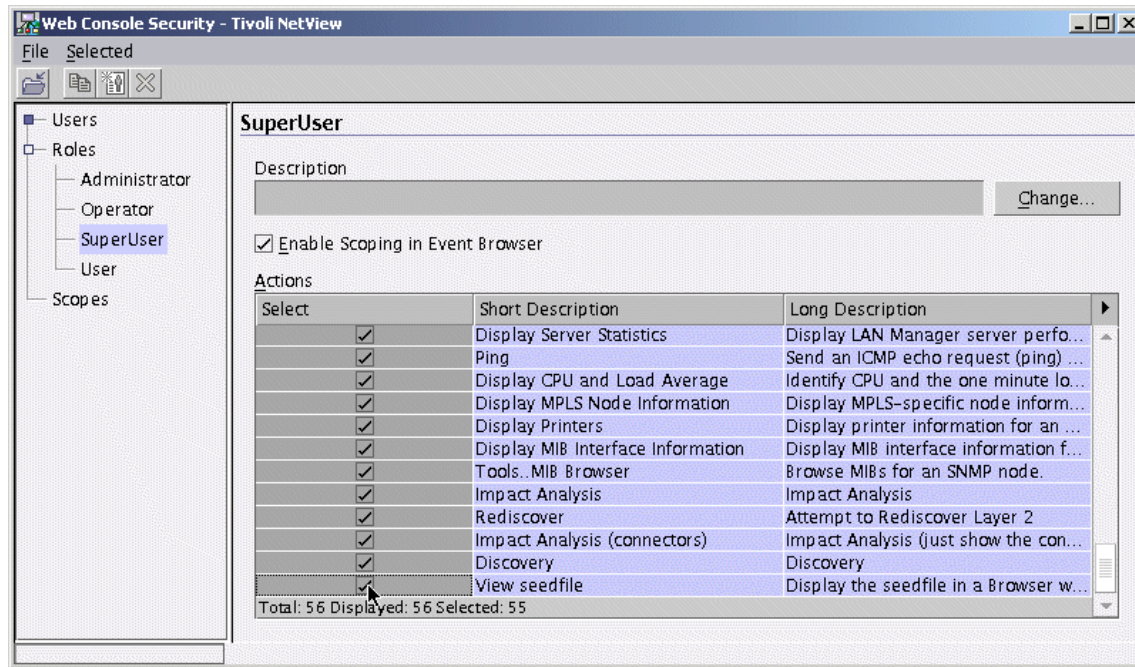


Figure A-6 The Roles window with the new menu entry

In case either the action or the menu file is malformed, you see an error message window as shown in Figure A-7. Review your definitions and correct any errors.

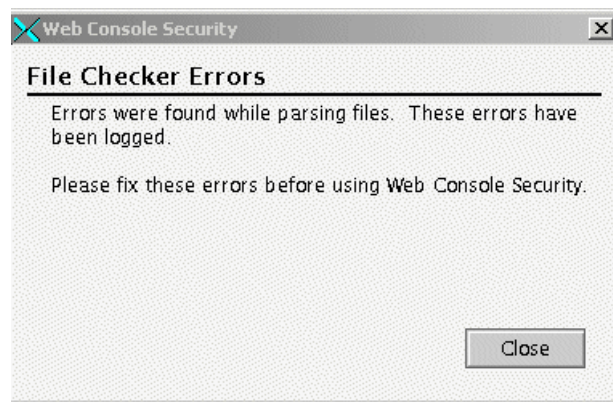


Figure A-7 Parsing error

If you launched the security console via a command line, the parsing errors are written to the console. They are also logged under `/usr/OV/log/ecurityconsole.log`. Near the end of the log file, you find such entries

as the one shown in Example A-6, which further specifies the type and location of the error.

*Example: A-6 Parsing errors in securityconsole.log*

---

```
2003-09-24 09:22:51,962 [Thread-4] ERROR com.tivoli.netview.securityconsole.server.JDOMHelper
- failed to parse byte stream -- org.jdom.JDOMException: Error on line 23: The element type
"ArgList" must be terminated by the matching end-tag "</ArgList>".
2003-09-24 09:22:52,620 [AWT-EventQueue-0] ERROR
com.tivoli.netview.securityconsole.server.SecurityConsoleUI -
1. Failed to parse file "/usr/OV/www/webapps/netview/warf/MyActions.xml"
-- Error on line 23: The element type "ArgList" must be terminated by the matching
end-tag "</ArgList>".
```

---

If the security console is displayed and the menu is activated for all necessary roles, click **File** → **Save** even if you did not change any entries. Saving initiates a Web server restart and a reread of all console definitions.

To test the function of the new entry, open the NetView Web console using your favorite method. Example 7-9 shows the new submenu in the Tools menu, which is the View seed file entry that we defined previously in this section.

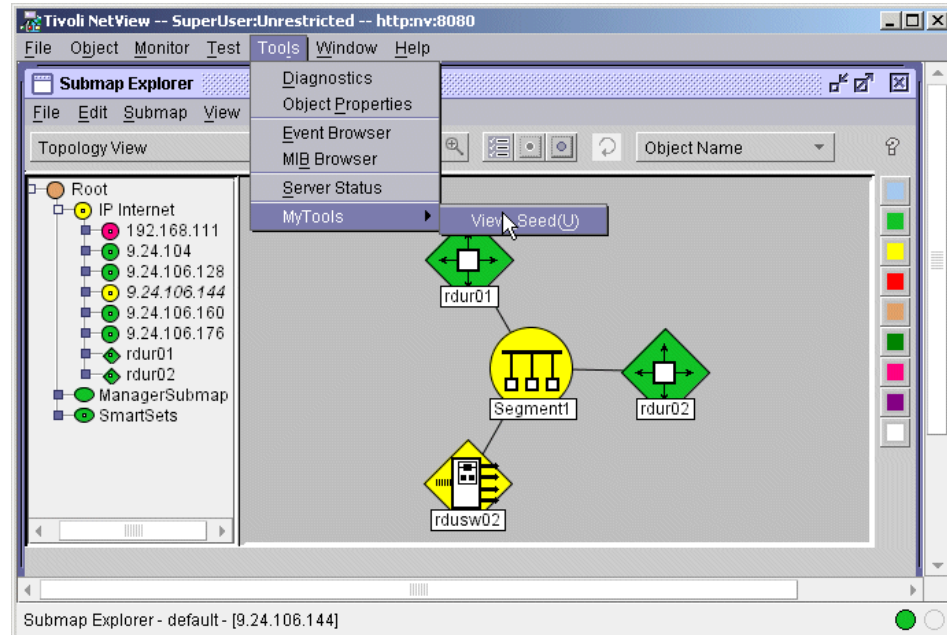


Figure 7-9 NetView Web console and the new menu



Now you can display your seed file from the Web console. Select **Tools** → **Mytools** → **View seed file** from the Web console menu. A browser window (Figure A-8) opens that shows the result of the operation.

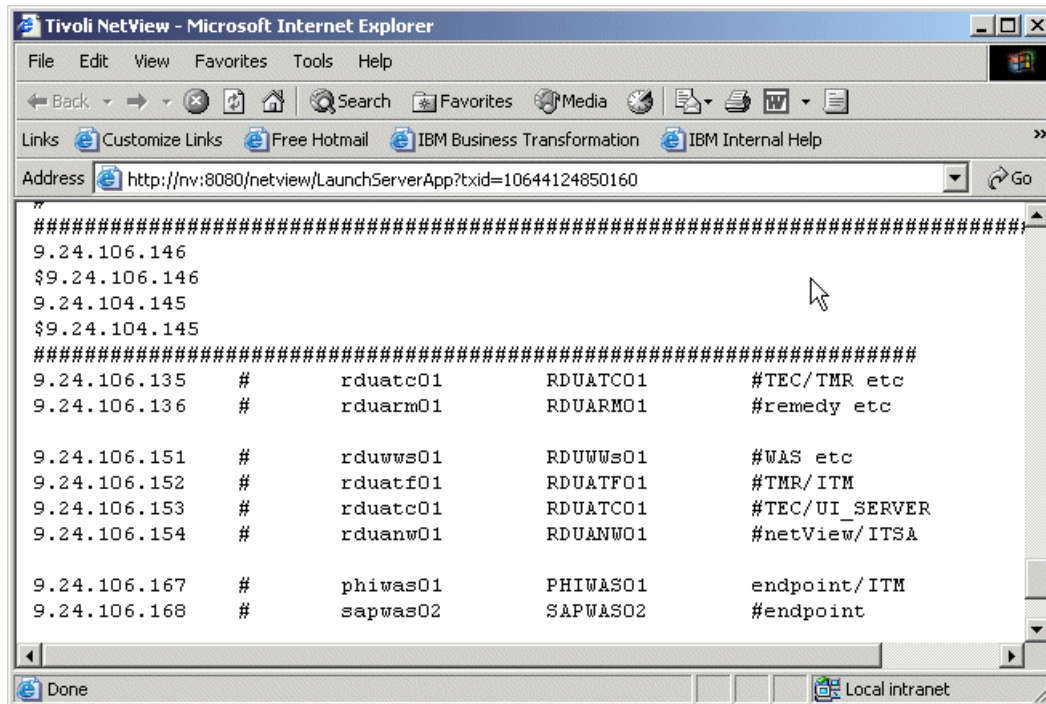


Figure A-8 The resulting output

## A smartset example

Smartsets are great when you need to group objects of the same type or in a specific location. They provide an overview of the status of the type of selected object without clicking through multiple submaps. An additional advantage is, in case you defined the correct rule, the dynamic nature of a smartset.

You can define rules to include NetView objects based on the actual contents of an attribute. For example, we define a smartset, which displays only those connection objects, switches, and routers that are marked in a status other than normal.

The status for a switch is maintained in two object attributes:

- **IP Status attribute:** Is set by the layer 3 topology management components only. In case of a switch, it never shows the status *marginal*.

- **Layer2Status attribute:** Is set by IBM Tivoli Switch Analyzer and shows a status that is *not equal normal* in case IBM Tivoli Switch Analyzer cannot status poll the ports discovered for that node.

Based on these conditions, we can develop a simple rule, which matches only those switches that are not up and running:

- The device or object being included must be of type connector.
- Its IP status or Layer2 status must be other than normal.

Now we need to bring this into a smartset rule definition:

```
((('isConnector' = True) && ('IP Status' != 'Normal') || ('Layer2Status' != 'Normal'))
```

Then define the rule with the ruleset editor:

1. From the NetView EUI, select **Tools** → **Smartset Editor**.
2. In the window that opens, select the **Text Editor** button and click **Add**.
3. The compound rule editor opens. The editor window that you normally use to define simple rules cannot be used in our case, because we compare status attributes. The simple rule editor only allows us to define an equal condition in the form <status attribute = '<enumeration of states>'. It negates the result at once, which makes an ineffective definition. In our case, we want to test for a status not equal normal.

Give the rule a name and a description. Enter the rule definitions as shown in Figure A-9. Then click **OK**.

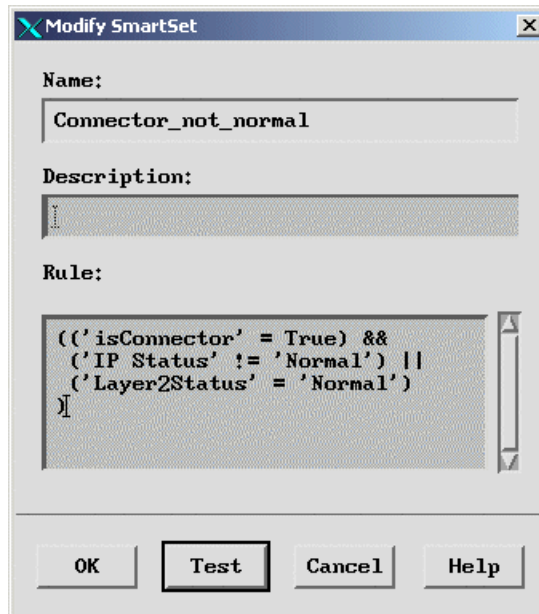


Figure A-9 The rule definition

4. If you have some failing connector devices, you can test the rule. Otherwise save it. Saving the rule creates a smartset, which remains in *Unknown* status until it is opened for the first time. Then it shows you the objects similar to what you see in Figure A-10. This example contains the failing objects from test case 2 in 5.2.7, “Real-life example” on page 161.

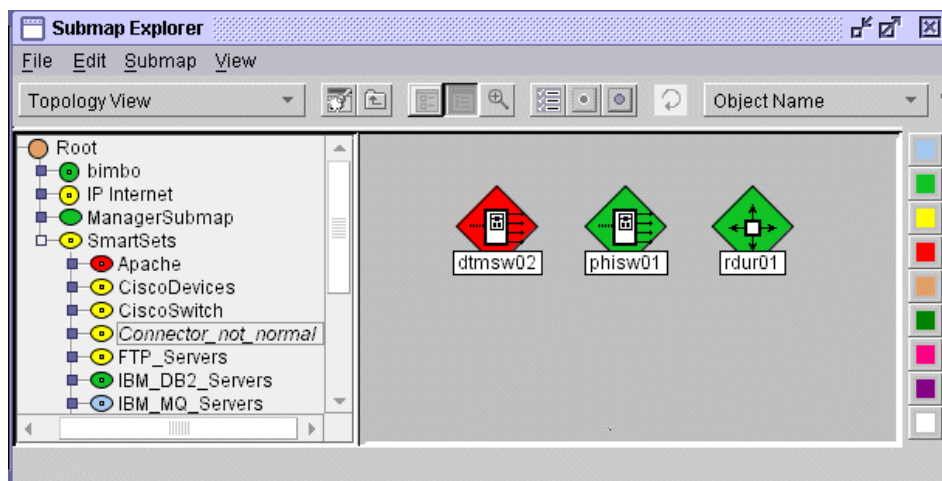


Figure A-10 A dynamic smartset

After you resolve the failure conditions, NetView removes the affected objects from the smartset submap as shown in Figure A-11.

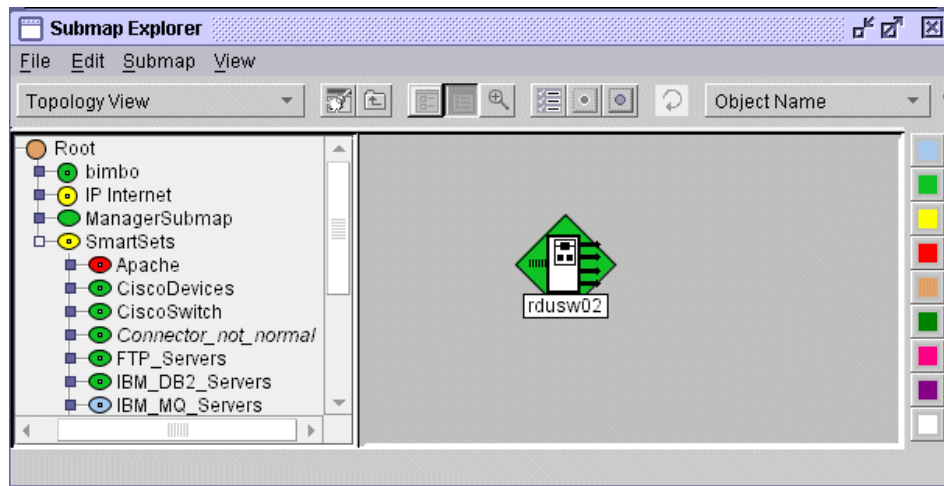


Figure A-11 Test situation resolved

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 422. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Tivoli NetView 6.01 and Friends*, SG24-6019
- ▶ *Tivoli Web Solutions: Managing Web Services and Beyond*, SG24-6049
- ▶ *Tivoli Business Systems Manager Version 2.1: End-to-end Business Impact Management*, SG24-6610

## Other publications

These publications are also relevant as further information sources:

- ▶ *Release Notes for NetView for UNIX, Version 7.1.2*  
<http://www-1.ibm.com/support/docview.wss?uid=swg21063303>
- ▶ *IBM Tivoli Switch Analyzer Troubleshooting Guide, Version 1.0*  
<http://www.ibm.com/software/support>
- ▶ *IBM Tivoli NetView for UNIX 7.1.3 Release Notes*, GI11-0927
- ▶ *Tivoli NetView for Windows NT Programmer's Reference, Version 7.1*, SC31-8890
- ▶ *Tivoli NetView for UNIX User's Guide for Beginners, Version 7.1*, SC31-8891
- ▶ *Tivoli NetView for UNIX Administrator's Guide, Version 7.1*, SC31-8892
- ▶ *Tivoli NetView for UNIX Administrator's Reference, Version 7.1*, SC31-8893
- ▶ *Tivoli NetView Web Console User's Guide, Version 7.1*, SC31-8900
- ▶ *IBM Tivoli Enterprise Console Command and Task Reference, Version 3.9*, SC32-1232
- ▶ *IBM Tivoli Enterprise Console Installation Guide, Version 3.9*, SC32-1233

- ▶ *IBM Tivoli Enterprise Console Rule Developer's Guide, Version 3.9, SC32-1234*
- ▶ *IBM Tivoli Enterprise Console User's Guide, Version 3.9, SC32-1235*
- ▶ *IBM Tivoli Enterprise Console Release Notes, Version 3.9, SC32-1238*
- ▶ *IBM Tivoli NetView for Windows Release Notes, Version 7.1.4, SC32-1239*
- ▶ *IBM Tivoli Enterprise Console Event Integration Facility Reference, Version 3.9, SC32-1241*
- ▶ *IBM Tivoli Enterprise Console Adapters Guide, Version 3.9, SC32-1242*
- ▶ *IBM Tivoli NetView for UNIX Administrators Guide, Version 3.9, SC32-1246*
- ▶ *IBM Tivoli Enterprise Console Rule Set Reference, Version 3.9, SC32-1282*

## Online resources

These Web sites are also relevant as further information sources:

- ▶ Tivoli software literature and technical resources  
<http://www.ibm.com/software/tivoli/library>
- ▶ Tivoli software support  
<http://www.ibm.com/software/sysmgmt/products/support>
- ▶ IBM software support  
<http://www.ibm.com/software/support>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)

# Index

## Symbols

\$AGENT\_ADDR 195  
\$BINDIR/./generic\_unix/TME/ACF\_REP 378  
\$BINDIR/TME/TEC directory 304  
\$BINDIR/TME/TEC/scripts directory 254  
\$BINDIR/TME/TEC/scripts/wstartmaint.sh 330  
\$COMMUNITY 195  
\$DBDIR/dependencies.pro file 240  
\$ENTERPRISE 195  
\$SOURCE\_TIME 195  
\$SPECIFIC 195  
\$TYPE 195  
\$VARBIND 195  
\*.modem 248  
.tec\_gateway\_diag\_config file 363, 377  
/tmp/notify\_bob.out log file 256  
/usr/OV/bin/ovxbeep 246  
/usr/OV/bin/ovxecho 247  
/usr/OV/ITLS2/conf/topo\_server.ini 400  
/usr/OV/log/trapd.trace 375  
/usr/OV/prg\_samples/nnm\_examples/beeper/beep\_951x 247  
@limit\_discovery 203  
\_cleanup\_admin parameter 239  
\_cleanup\_interval parameter 239  
\_cleanup\_list parameter 239  
\_default\_span parameter 239  
\_over\_time parameter 337

## A

ACF (Adapter Configuration Facility) 88, 91–92  
ack\_close\_time 288  
action node 226  
actionable event 44, 58  
Actionhandler tag 412  
actionsvr daemon 120  
activating rule set 345  
Adapter Configuration Facility (ACF) 88, 91–92, 378–379  
adapter configuration in NetView 193  
adding a field to the NetView database 320  
AdditionalLegalTrapCharacters 350  
addtrap command 181

administrator name 287, 329  
AIX installation notes 121  
all\_duplicates 236  
all\_instances 236  
allEvents 214  
ambiguous case 226  
ambiguous exception 224  
Apache smartset 133  
API (application programming interface) 17, 104  
application layer 143  
application programming interface (API) 17, 104  
Application Registration File (ARF) 409  
applications 359  
ARF (Application Registration File) 409  
ArgList tag 412  
assigning responsibilities 37  
attribute 119  
authentication trap 201  
automated actions 5, 19, 343  
    executing 339  
    security fix in NetView 350  
    triggering 339  
automated recovery 20  
automatic rediscovery 154  
automation 1–2, 4–5, 19, 77, 173, 338, 346, 354  
    best practices 78  
    cross-platform correlation 80  
    event flow 370  
    gathering diagnostic data 79  
    IBM Tivoli Enterprise Console 351  
    IBM Tivoli Monitoring 354  
    implementation considerations 80  
    in a NetView rule set 344  
    NetView 338  
    problem verification 79  
    recovery 80  
avoiding unwanted unsecure events 208

## B

back log 382  
bad practice 50  
BAROC (Basic Recorder of Objects in C) 95  
Basic Recorder of Objects in C (BAROC) 95

- best practices 25, 237
  - automation 78
  - automation with NetView 349
  - correlation 51
  - customization 338
  - duplicate detection and throttling 50
  - escalation 60
  - event management products 173
  - event synchronization 67
  - filtering 44
  - filtering by limiting monitoring scope 202
  - flowchart 82
  - for NetView filtering 200
  - IBM Tivoli Enterprise Console gateway 212
  - maintenance mode 77
  - NetView console 247
  - notification 58
  - setting severities 277
  - severity mapping between tools 263
  - trapd.conf 175
  - trapd.log data 187
  - trouble ticketing 69
  - using NetView adapter 199
- bidirectional update capability 308
- binutils 123
- blank.modem 248
- block event display ruleset node 320
- blocking traps 320
- bo\_set\_slotval 294
- boolean operator 119
- buffer filtering 205
- business impact 18, 62, 263
  - escalation rule 279
  - management 41
  - software 44
- business impact escalation 18, 64, 66, 286

## C

- cache searching 381
- called\_assoc\_flag 315
- card mode 113
- case study 357
- cause\_date\_reception attribute 243
- cause\_event\_handle attribute 243
- CDS (class definition statement) file 206
- centralized event management 21
- change request 96
- change rule 96

- change\_event\_severity 294
- Change\_Severity task 295
- check\_cache\_for\_escalation 289
- check\_maintenance\_mode rule 336
- check\_maintenance\_timeout timer rule 336
- check\_overtime\_timer timer rule 337
- choose top X problems from each support area 28
- class definition statement (CDS) file 206
- classes to escalate 288
- cleanup.rls 239
- clearing event 48, 54
- collector 100
- collector event 100
- collector rule 216
- command
  - addtrap 181
  - executing from rule sets 343
  - executing from trapd.conf 339–340
  - IBM Tivoli Monitoring Query 230
  - itmquery 230
  - mib2trap 181
  - ovstart 144
  - ovstart itsl2 154
  - ovstart trapd 371
  - ovstatus trapd 371
  - ovstop 144
  - ovstop itsl2 154
  - ovtopodump -X 146, 148, 370
  - ovxbeep 246
  - ovxecho 246
  - postemsg 191
  - reorg 383
  - repetitive sequences 20
  - snmptrap 377
  - snmpwalk 146
  - wpostemsg 191
  - wrb 97–98
    - comprules 209
    - imptgrule 209
    - loadrb 209
  - wrb -comprules 389
  - wrb -deldp 241
  - wrb -imptdp 241
  - wsetesvrcfg 380
  - wtdumprl 377
  - xecho 181
- command line utilities 157
- comparison operator 119
- compound rule 97



- connection-oriented service 92
- consistent standards 36
- console 209
  - IBM Tivoli Enterprise Console 249
  - Java-based 249–250
  - NetView 245
  - Web-based 249, 251
- console filter 210
- console filtering 209
- core router 63
- correlation 1, 8, 51, 95, 173, 218
  - avoiding upstream correlation 306
  - best practices 51
  - cross-host 14
  - cross-platform 13, 80
  - e-business 243
  - event flow 370
  - IBM Tivoli Enterprise Console 232
  - IBM Tivoli Monitoring 244
  - root cause 11
  - rule 97
  - sequence 237
  - topology-based 15, 56, 77
  - using NetView rules 226
  - with NetView and IBM Tivoli Switch Analyzer 218
- correlation rule set 238, 240, 249
- correlation.rls 238, 240
- correlation\_configure rule 240
- correlation\_parameters action 240
- corrstat1 field 321
- cprb option 98
- create\_event\_criteria 299
- critical 61
- CRM (customer relationship management) 90
- cross-host correlation 14
- cross-platform correlation 13, 80
- crtrp option 98
- customer relationship management (CRM) 90
- customization, best practices 338

## D

- daemon 144
  - trapd 119
- Data Driven Event Management Design (DDEMD) 21, 31, 51
- data flow 87
- data link layer 143

- data repository 43
- database 358
- database managed table 383
- date\_reception 243, 380
- DB2 241
- DB2 Universal Database 358
- DDEMD (Data Driven Event Management Design) 21, 31, 51
- debug parameter 378
- debugging
  - NetView rules 395
  - rules with trace directive 388
  - state correlation 394
  - state correlation and rule sets in NetView 394
- decentralized event management 21
- decision node 226
- de-duplication 45
- de-escalation 44, 65
- default XML file 99
- definitions 94
- dependency 238
  - rule set 240
- dependency.rls 240
- dependency\_type 240
  - detailed suboption 99
- device
  - discovery 203
    - priority 18
    - type 18
  - unmanaging 204
- diagnostics 20, 247, 370
- disabled mode 221
- discover
  - retry attempts for failed discovery 154
  - supported layer 2 devices 146
- discovered service 133
- discovery 146
  - network 103
  - problems 158
  - process 147
  - status 148
  - switch 155
- discovery\_interval field 154, 400
- distributed management 103
- distributed rule base 97
- distribution router 63
- dmae\_mn\_send\_email 261
- dmae\_mn\_send\_notice 261
- double monitor 45

- downstream correlation from trouble-ticketing system 306
- downstream switch 167
- dummy shell 105
- dup\_detect 216
- duplicate 100
- duplicate detection 7, 45, 50, 95, 212, 296, 314
  - IBM Tivoli Monitoring 217
  - implications 46
- duplicate event 50, 100, 212
  - detection 45
  - suppressing 45
- duplicate rule 216

## E

- e-business
  - correlation 243
  - rule set 238–240
- ebusiness.rls 238–241
- e-mail 16, 59
- e-mail notification 239
- e-mailing 57, 252
- EMMD (Event Management and Monitoring Design) 28–29, 236
- End User Interface (EUI) 105
- EndDay 319
- EndTime 319
- enterprise tier 6
- entry point into the hierarchy 6
- entry tier of the hierarchy 6
- epoch time 330
- escalate.rls 239, 286, 291
- escalate\_configure 289
- escalate\_housekeeping 290
- escalate\_old\_events 289
- escalate\_specific\_event 290
- escalating events
  - with IBM Tivoli Enterprise Console event server 285
  - with NetView 279
- escalating severity using IBM Tivoli Enterprise Console rules 293
- escalation 10, 17, 50, 60, 95, 236, 262, 274
  - best practices 60
  - business impact 18, 62, 66, 286
  - implementation considerations 65
  - in netview.Rls 285
  - intervals 62

- rule 285
- sequence 10
- severity 263
- time limits 288
- to ensure problems are addressed 17
- using escalate.rls 286
- using event\_thresholds.rls 291
- worsening condition 64, 66, 284
- escalation check frequency 287
- escalation rule set 238–239, 286
- escalation.rs rule 283
- ESE.automation file 200, 345, 399
- EUI 106, 403
  - behavior 113
  - configuration in NetView 402
- EUI (End User Interface) 105
- ev\_classes 288
- event 4, 94, 113, 217
  - actionable event 58
  - adapter 91, 94
  - avoiding unwanted unsecure events 208
  - browser 159
  - buffer filtering 205
  - cache searching 381
  - clearing 54
  - clearing event 8, 48
  - collector 100
  - colors for IBM Tivoli Enterprise Console 263
  - correlation 2, 4, 8, 236
  - correlation analysis 30
  - correlation problems and clearing 8
  - database 93, 216
  - duplicate 100
  - escalation 10, 262
  - filtering 7
  - filtering and forwarding 174
  - filtering nearer to the source 207
  - filtering rule set 238
  - flow 6, 370
  - forwarding 285
  - forwarding events of interest 210
  - group 209
  - handling 330
  - handling decisions 30
  - handling in maintenance mode 74
  - handling policies 29
  - handling related events 315
  - hanging or halted 399
  - ID 380

- informational event 58
  - internally generated 208
  - management 4
  - management hierarchy 26
  - management infrastructure 7
  - matching 100
  - name 43
  - notification 245
  - orphaned 67
  - problem event 58
  - processing 5, 119
  - processing concepts and issues 6
  - processing hierarchy 6
  - processing performance 381
  - processing tool 57
  - processors 6
  - raw 91
  - recovery 8
  - repertoires 30
  - repository 383
  - severity 5, 60, 68
  - severity escalation 129
  - source 6, 29
  - summary 245
  - symptom 52, 54
  - symptom event not requiring action 11
  - symptom event requiring action 12
  - synchronization 15
  - synchronization best practices 67
  - type 43
  - unhandled 60
  - verifying action taken 392
  - view 245
  - event attributes 327
    - node 327
  - event class 94, 252, 288
  - event console 93, 105, 112
    - configuration 403
    - Java 93
  - event management 1, 4
    - categories and best practices 25
    - policies and procedures 33
    - policies and standards 32
    - process guidelines 54
    - products 173
    - reviewing the process 33
  - Event Management and Monitoring Design (EMMD) 28–29, 236
    - approach 29
    - diagrams 31
    - methodology 29
    - tools 31
    - workbooks 31
  - Event Management Process Guidelines 54
  - event reception
    - in IBM Tivoli Enterprise Console 377
    - in NetView 371
  - event server 93, 95, 212, 216
    - CLOSED event synchronization 297, 300
  - event severity 252, 264, 288
    - in IBM Tivoli Enterprise Console 272
    - in Netview for UNIX 270
    - in NetView for Windows 271
    - in trouble-ticketing system 275
  - event synchronization 16, 66, 295–297
    - CLOSED from a high-level IBM Tivoli Enterprise Console event server 300
    - CLOSED from a low level IBM Tivoli Enterprise Console event server 297
  - event\_filtering.rls 209, 238
  - event\_handle 243, 380
  - event\_thresholds.rls rule set 213, 291
  - examples 370
  - Exceed 105
  - exec\_program predicate 254, 352
  - executing automated actions 339
- F**
- fact file name 330
  - fact files 256
  - failover process 5
  - failure reporting 48
  - false positives 45
  - FDDI (Fiber Distributed Data Interface) 110
  - FFDC (first failure data capture) 130, 134
  - Fiber Distributed Data Interface (FDDI) 110
  - field to the NetView database 320
  - filter by SMEs 42
  - filtered dynamic work space 399
  - filtered event work space 346
  - filtering 7, 39–40, 174
    - at the source 7
    - bandwidth considerations 39
    - best practices 44
    - by limiting monitoring scope 202
    - by smartset 194
    - console 209

- event buffer 205
- event server limitations 39
- events nearer to the source 207
- how to filter 40
- IBM Tivoli Enterprise Console 205
- IBM Tivoli Monitoring 210
- in tecad\_nv6k.conf 198
- manageability 40
- NetView 174
- NetView best practices 200
- network considerations 39
- redundant information 39
- tecad\_nv6k.cds 195
- trapd.conf 182
- using event\_filtering.rls 209
- what to filter 41
- where to filter 41
- why 39
- filtering and forwarding 205
- first failure data capture (FFDC) 130, 134
- firstEvent 214
- fixedWindow 215
- flagging traps 320
- flapping error condition 82
- fluttering error condition 82
- format file 205
- forwardEvents 215
- forwarding 7, 66, 174
  - events of interest and suppressing others 210
  - IBM Tivoli Enterprise Console 205, 346
  - IBM Tivoli Monitoring 210
  - NetView 174
  - traps to IBM Tivoli Event Console 159
- forwarding.rls 297–298
- forwarding\_configure rule 298
- fqhostname 241–242, 391
- fqhostname attribute 333

## G

- gateway 92, 206, 212, 296, 377
- generate\_event predicates 236
- generating sample events 391
- group paging 72
- GUI Rule Builder 99

## H

- HARMLESS severity 239
- heartbeat rule set 239

- heartbeat.rls 239
- help desk 38
- hiding traps from view 320
- high risk server 63
- Highest\_level keyword 379
- hole 217
- host 133
- host name 257
- host\_a 240
- host\_b 240
- host-based maintenance 74
- housekeeping frequency 287

## I

- IBM Event Management and Monitoring Design (EMMD) 236
- IBM Tivoli Business Systems Manager 64
- IBM Tivoli Enterprise Console 205
  - automation 351
  - automation scripts 352
  - automation tasks 351
  - components 91
  - correlation 232
  - data flow 87
  - duplicate detection 212
  - escalating events 279
  - event colors 263
  - event reception 377
  - event server to escalate events 285
  - event severity escalation 129
  - event synchronization 295, 297
  - forwarding 346
  - gateway 92, 206, 212, 296
  - highlights 86
  - IBM Tivoli NetView 92
  - input 88
  - maintenance mode 328
  - monitoring and maintaining logs 383
  - multiple servers 297
  - notification 249
  - output 90
  - overview 85
  - problem diagnosis 89
  - problem verification 347
  - processing 89
  - resolution attempts 89
  - revised integration for Version 7.1.3 125
  - revised integration for Version 7.1.4 128

- rule tracing 388
- rules 207, 251
- rules to escalate severity 293
- scripts 254
- setting severity 272
- severity 263
- state correlation engine 232
- state correlation gateway 284
- terms and definitions 94
- throttling 212
- trouble ticketing 302, 307
- troubleshooting 377
- Version 3.9 out-of-the-box rule 297
- versus NetView for trouble ticketing 307
- IBM Tivoli Event Console forwarding traps 159
- IBM Tivoli Monitoring 260, 354
  - correlation 244
  - duplicate detection 217
  - filtering and forwarding 210
  - notification 260
  - throttling 217
  - Web Health Console 260
- IBM Tivoli Monitoring for Business Integration 241
- IBM Tivoli Monitoring for Databases 241
- IBM Tivoli Monitoring for Web Infrastructure 241
- IBM Tivoli Monitoring Query command 230
- IBM Tivoli NetView 101
- IBM Tivoli NetView distributed 103
- IBM Tivoli Switch Analyzer 116, 399
  - correlation 218
  - daemons and processes 144
  - discovery 146
  - duplicate detection 212
  - features of V1.2.1 144
  - installation 364
  - ITSL2 Enterprise trap 159
  - layer 2 network management 142
  - layer 2 root cause 160
  - layer 3 network management insufficiencies 143
  - NetView layer 2 topology report 149
  - overview 141
  - root cause analysis 160
  - throttling 212
  - trap 159
- IBM\_DB2\_Servers smartset 133
- ibm5853.modem 248
- ibm7855.modem 248
- IHS smartset 133
- implementation approach 26
  - choose top X problems from each support area 28
  - perform Event Management and Monitoring Design 28
  - report only known problems and add them to the list as they are identified 27
  - send all possible events 26
  - start with out-of-box notifications and analyze re-iteratively 27
- implementation plan 31
  - import suboption 98
- imprbrule option 99
- imprprule option 99
- incident report 81
- indication 217
- inetd 123
- informational event 58
- installation issues 363
- installation notes 120
- Integrated TCP/IP Services 102
- integration platform 104
- Interface Down trap 156
- internally generated events 208
- IP Internet map 107
- IP segment symbol 110
- IP status 149
- IP Status attribute 417
- IPaddress 319
- IT environment assessment 21
- itmquery command 230
  - examples 231
- ITMQUERY function 129
- ITSATool 147
- ITSL2 Enterprise trap 159

**J**

- Java event console 93
- Java Runtime Environment (JRE) 121
- Java-based console 249–250
- JRE (Java Runtime Environment) 121

**K**

- known problems 27

**L**

- lab

- approach 237
- databases 358
- environment 358
- layout 359
- network components 361
- operating systems 358
- servers 360
- setup and diagram 359
- software and operating systems 358
- Tivoli products 358
- lastEvent 214
- latency 240, 287, 329
- layer 142
- layer 2
  - connectivity devices 224
  - devices 146
  - discovery report 150
  - network management 142
  - OID? 149
  - report 151
  - status 156
  - topology report 149
- layer 3
  - network management insufficiencies 143
  - router fault isolation 160
- Layer2Status field 156
- Layer2Status field value 149
- line mode 113
- link\_effect\_to\_cause predicate 243
- Linux installation notes 123
- load-balancing cluster 55
- location symbol 109
- logfile adapter 205, 212
- long-term maintenance 328
- IQ8Ekr suffix 398
- lsrbpack option 99
- lsrbrule option 99

## M

- machine layout 362
- maintenance event severity 329
- maintenance mode 19, 72, 318
  - automate 73
  - automation 81
  - handling through NetView rules 320
  - host-based maintenance 74
  - IBM Tivoli Enterprise Console 328
  - initiating 330

- NetView 315
  - network topology considerations 76
  - prolonged 75
  - rule set 238, 329
  - status notification 73
  - system events 74
  - terminating 334
  - unmanaging SNMP devices 316
- maintenance\_mode.rls 238, 329, 334
  - rule 335
  - test cases 337
  - testing 337
- maintenance\_mode\_configure
  - configuration rule 337
- maintenance\_mode\_configure rule 335
- maintenance\_received rule 335
- manager 38
- manager submap 107
- map 105
- map view 105
- marginal status 417
- match rule 206
- matching event 100
- mean-time to repair 20, 33, 45, 50, 55, 79
- menu integration 124
- Method tag 412
- MIB browser 106
- mib2trap command 181
- monitoring 126
- monitoring scope 202
- multiple servers 297

## N

- naming convention 36, 359
- native NetView console 105
- nested location 109
- netfmt command 375
- netmon configuration 225
- netmon seed file 203, 230, 410
- nettl command 374
- nettl operation 374
- NetView 92, 245
  - adapter 191
  - adapter configuration 193
  - Application Registration File (ARF) 409
  - automation 338
  - automation in a rule set 344
  - best practices for using the adapter 199

- changes in 7.1.3 and 7.1.4 124
- command line utilities 157
- console 245
- console best practice 247
- correlation with IBM Tivoli Switch Analyzer 218
- debugging rules 395
- debugging state correlation and rule sets 394
- duplicate detection 212
- escalating events 279
- EUI 105, 112
- EUI configuration 402
- event console 105, 112
- event console configuration 403
- event synchronization 295
- features and enhancements for Version 7.1.3 124
- filtering and forwarding 174
- filtering best practices 200
- GUI 414
- IBM Tivoli Enterprise Console event severity escalation 129
- IBM Tivoli NetView distributed 103
- installation issue 364
- installing the adapter 192
- Integrated TCP/IP Services 102
- integration into the topology map 157
- layer 2 topology report 149
- maintenance mode 315
- map view 105
- maps, submaps 106
- native console 105
- new features, enhancements for Version 7.1.4 126
- new functions in Version 7.1.4 131
- overview 101
- paging rules 247
- revised Tivoli Enterprise Console integration for Version 7.1.3 125
- revised Tivoli Enterprise Console integration for Version 7.1.4 128
- root cause analysis 160
- Router Fault Isolation (RFI) 221
- rules 199
- rules for correlation 226
- security fix impact on automated actions 350
- severity 263
- smartset example 417
- Submap Explorer 157
- suggested configuration 401
- TEC Integrated TCP/IP Services 103
- throttling 212
- tool window 105
- trap color 263
- trapd.conf 175
- using rules to handle maintenance mode 320
- versus IBM Tivoli Enterprise Console for trouble ticketing 307
- visualization components 104
- Web Application Registration File (WARF) 409
- Web console 114, 156–157
  - applet 406
  - installation 404
  - menu extension 408
  - menu integration 124
  - security 407
  - stand-alone installation 404
- NetView database, adding a field 320
- NetView layer 3 router fault isolation 160
- NetView rule set 238–239
- netview.rls 125, 238–239, 286, 295
  - escalation 285
  - rule set 399
- netviewd daemon 123
- network
  - component 361
  - discovery 103
  - layer 143
  - layout 362
  - topology considerations 76
- newhayes.modem 248
- node 327
  - block event display ruleset 320
  - name 149
  - override 327
  - override ruleset 279, 320
  - Query Database Field 325–326
  - removing 162
- Node added trap 155
- non-TME adapter 192
- not equal norma 418
- notification 16, 56, 217, 236, 244, 252, 260
  - best practices 58
  - chain 278
  - how to 56
  - IBM Tivoli Enterprise Console 249
  - maintenance mode status 73
  - NetView 245
  - out-of-the-box 27

- rule set 238
- notify.rls 238, 252, 254
  - customizing for which events to send notifications 252
- Notify\_Bob 256
- notify\_bob script 256
- Notify\_Bob.Sh 256
- notify\_configure rule 252, 254
- notify\_for\_fatal\_events rule 254
- nv.carriers 248
- nvcold daemon 127
- nvcord 396
- nvcorrd daemon 119
- nvdbimport command 322
- nvmaputil utility 317
- nvpage command 248–249
- nvpager.config 248
- nvpagerd daemon 120
- nvpaging.protocols 249
- nvserverd daemon 120, 189
- nvsetup 414
- nvsniffer daemon 126

## O

- o DESC option 377
- object list 119
- occurrence 217
- OLA (operations-level agreement) 61
- oldhayes.modem 248
- on-call list 72
- Open Maintenance event group 337
- Open Systems Interconnection (OSI)
  - application layer 143
  - data link layer 143
  - layers 142
  - model 141–142
  - network layer 143
  - physical layer 142
  - presentation layer 143
  - session layer 143
  - transport layer 143
- open\_ack\_time 288
- operating system 358
  - supported 121
- operations-level agreement (OLA) 61
- operator
  - boolean 119
  - comparison 119

- operator-initiated rediscovery 154
- organizational considerations 21
- orphaned event 67
- OSI (Open Systems Interconnection) 141–142
- out-of-the-box rule 239, 252
- ovactiond daemon 397
  - customizing 342
  - performance considerations 343
- ovelmnd 120
- override node 327
- override ruleset node 279, 320
- ovesmd daemon 120
- ovstart command 144
- ovstart itsl2 command 154
- ovstart trapd command 371
- ovstatus trapd command 371
- ovstop command 130, 144
- ovstop itsl2 command 154
- ovtopodump -X command 146, 148, 370
- OVW Maps Exists field 316
- OVW Maps Managed field 316
- OVwDb object ID 149
- ovxbeep command 246
- ovxecho command 246

## P

- P%wKiw suffix 398
- paging 16, 57, 59, 247, 252
  - group 72
  - rules 247
  - utility 247
- partition
  - appearance 223
  - identification 223
  - suppressed polling 224
- pass-through 100
  - rule 232
- pdksh 123
- Peregrine priority code 275
- Peregrine ServiceCenter 302, 308
- perform Event Management and Monitoring Design 28
- physical layer 142
- place\_change\_request 294
- plain rule 96
- planning considerations 20
- pmd daemon 119
- policies 23



- and procedures 33
  - enforcing 38
- polling 224
  - interval 133
  - to routers 225
- PollTypes 319
- pop-up window 16, 245
- post mortem 34, 44, 50
- postmsg command 191
- presentation layer 143
- primary event 11
- probabilistic mode 221
- problem diagnosis 89
- problem event 8, 58
- problem post mortem 44
- problem verification 19, 79, 81, 346–347
  - automation 346
- process owner 37
- PROCESSED event 380
- processes 144
- processing events 119
- processing logic 333
- profiles to limit 211
- project deliverables 30
- prolog rule 235
- prolog-based rule 99
- prolonged maintenance mode 75

## Q

- qblazer.modem 248
- Query Database Field node 325–326
- QUEUED event 380

## R

- raw event 91
- RDBMS Interface Module (RIM) 93
- real-life example 161
- reception buffer 380
- reception log 383
- recovery 20, 80, 223
  - actions 346
  - commands 5
  - event 8
- rediscovery 154
  - operator initiated 154
- redo request 96
- referenceNo slot 309
- related events 315

- reorg command 383
- repetitive command sequences 20
- report only known problems, add to list as identified 27
- report\_period 382
- reset on match event 100
- reset on match rule 233
- resetOnMatch 100
- resolution attempts 89
- Resource Model Builder (RMB) utility 211
- resource models 210
- responsibilities 37
- retry attempts for failed discovery 154
- RFI (Router Fault Isolation) 160, 218
- RIM (RDBMS Interface Module) 93
- RMB (Resource Model Builder) utility 211
- root cause 11
  - analysis 103
  - analysis using IBM Tivoli Switch Analyzer and NetView 160
  - correlation 11
  - of layer 2 in IBM Tivoli Switch Analyzer analysis 160
  - router 224
- root map 106
- root.baroc 95
- root.baroc EVENT class 310
- router analysis 223
- Router Fault Isolation (RFI) 160, 218
  - configuration 221
  - enabling and disabling 225
  - mode 221
  - NetView 221
  - overview 219
  - stopping 221
- router link, defective 168
- Router Status trap 160
- rule 95, 117, 143, 161, 247, 417
  - business impact escalation 279
  - cache 380
  - change rule 96
  - collector 216
  - configuring maintenance\_mode.rls 329
  - correlation 95
  - correlation rule 97
  - debugging in NetView 395
  - debugging with trace directive 388
  - duplicate 216
  - duplicate detection 95

- e-mailing 252
- escalation 95, 285
- escalation.rs 283
- event synchronization 295
- for long-term maintenance 328
- IBM Tivoli Enterprise Console 207, 251
- maintenance 328
- maintenance\_mode.rls 335
- matching 206
- out-of-the-box 239, 252
- paging 247, 252
- pass-through 232
- plain rule 96
- prolog 235
- prolog-based 99
- reset on match 233
- scripts 254
- simple rule 97
- smartset 118–119
- state correlation 233
- state correlation gateway 347
- state-based 100
- thresholding 95, 100
- timer rule 96
- trouble ticketing 236
- rule base 97, 235, 381
  - design approaches 236
  - distributed 97
  - rule set sequencing and dependencies 238
  - target 97
  - writing best practices 235
- Rule Cache full: forced cleaning 381
- rule pack 98
- rule set 98, 120, 394, 397
  - activating 345
  - correlation 238, 240
  - dependency 240
  - directory problems 398
  - e-business 238–240
  - escalation 239, 286
    - escalate.rls 238
  - event filtering 238
  - executing commands 343
  - heartbeat 239
  - maintenance mode 238, 329
  - NetView 238–239
  - netview.rls 399
  - notification 238
  - sequencing and dependencies 238

- trouble ticketing 238
  - using 199
- rule\_sets file 238
- rulepacks suboption 98
- ruleset editor 199, 397
- rulesets suboption 98

## S

- sample events 391
- sample\_period 382
- scenter.rls 310
- SCpmClosed class 314
- SCpmOpen class 308
- Scpmopened 313
- scripts 254
- secondary event 11
- Secure Sockets Layer security 125
- security fix 350
- seed file 203
  - netmon 410
- segment symbol 110–111
- send all possible events 26
- sequencing 238
- server\_handle 380
- servers 360
- serversetup command 342
- service decomposition 29
- service discovery 126
- Service Down Delete Interval 133
- service monitor 126
- service-level agreement (SLA) 18, 61, 263
- servmon 126, 131, 164
  - Apache smartset 133
  - application 117
  - daemon 131
  - IBM\_DB2\_Servers 133
  - IHS 133
  - polling interval 133
  - Service Down Delete Interval 133
  - WebSphereServers 133
- servmon.log 132
- session layer 143
- set\_event\_severity 293
- set\_fqhostname rule 241
- severity 5, 60, 68, 257, 263, 288
  - critical 61
  - defining 34
  - escalation 129, 262

- event 252
- HARMLESS 239
- IBM Tivoli Enterprise Console 263
- maintenance event 329
- mapping between tools 263
- NetView 263
- setting 264, 277
  - event severity in NetView for Windows 271
  - trap severity in NetView for UNIX 264
  - trap severity in NetView for Windows 267
- trouble ticket 263
- UNKNOWN 239
- warning 61
- simple rule 97
- skill level 22
- SLA (service-level agreement) 18, 61
- slideDown 216
- smartset 117, 133, 321
  - editor 118
  - example 417
  - filtering 194
  - membership 126
  - node 126
  - rule 118–119
  - submap 107
- SNMP address 149
- SNMP device in maintenance mode 316
- SNMP management 103
- SNMP trap 119
  - forwarding using trapd.conf 183
- snmptrap command 377
- snmpwalk command 146
- source 91
- standards 23
- start with out-of-box notifications and analyze reiteratively 27
- Start\_Maintenance task 331
- start\_maintenance\_timer timer rule 336
- StartDay 319
- StartTime 319
- state change monitoring 296
- state correlation 99, 213–214, 284, 361, 377, 394
  - engine 99–100, 232
  - gateway 92, 284
  - gateway rule 347
  - rule 99, 206–207, 233
- state machine 99
- state-based rule 100
- status 67
  - determining 156
  - displaying 156
  - marginal 417
  - not equal normal 418
- Status Events 182
- subject matter expert 38
- submap 107
- Submap Explorer 157
- subnet 119
- summary report 151
- supplied rule set 307
- support staff 38
- supported operating system 121
- supported platforms 120
- suppression 45
  - events 210
  - implications 46
  - polling in a partition 224
- switch discovery 155
- switch exception 224
- symptom 11
- symptom event 52, 54
  - not requiring action 11
  - requiring action 12
- Symptom Events 12
- sysObjectID 149
- system configuration 37
- systems management architect 37
- systems management tool implementations 33

**T**

- tablespace 383
- tasks 261
- TEC Integrated TCP/IP Services 103
- TEC\_Class 257
- TEC\_Error class 253
- tec\_forward.conf file 299, 301
- tec\_gateway 363
- tec\_gateway.conf file 363, 377
- tec\_gateway\_sce 363
- TEC\_Notice event 381
- tec\_reception trace file 382
- tec\_t\_evt\_rec\_log table 383
- tec\_t\_evt\_rep table 383
- TEC\_Test\_Event 299
- tecad\_nv6k.cds file 195, 271
- tecad\_nv6k.cfg file 192
- tecad\_nv6k.conf 198

- terminology 4
- terms 94
- testdowns.rs 347
- third-party processes 137
- threshold 100
- thresholding 95
- thresholding rule 100
- throttling 7–8, 45, 50, 212, 291
  - IBM Tivoli Monitoring 217
- time stamp 43
- time window 100, 214–215, 234, 382
  - considerations 46
- time-based rule 100
- timer rule 96
- timing considerations 15
- Tivoli Availability Intermediate Manager 92
- Tivoli Data Warehouse 104
- Tivoli Management Region (TMR) server 232
- Tivoli Monitoring for Business Integration 241
- Tivoli Monitoring for Databases 241
- Tivoli Monitoring for Web Infrastructure 241
- Tivoli NetView 101
- Tivoli products 358
- Tivoli software 363
- TME 10 Framework 192
- TME adapter 192
- TMR server 232
- tool implementer 38
- tool usage 22
- tool window 105
- topo\_server log 149
- topology display 103
- topology map 157
- topology view 245
- topology-based correlation 15, 56, 77
- trace 379
- trace directive 388
- transport layer 143
- trap 159
- trap color 263
- trap escalation 279
- trap severity
  - in NetView for UNIX 264
  - in NetView for Windows 267
- trapd daemon 119
- trapd -T command 375
- trapd.conf 175, 339
  - adding entries 176
  - configuring command execution 340

- filtering 182
- forwarding SNMP traps 183
- trapd.log file 119, 175, 182, 201
  - maintaining 186
- trapfrwd daemon 185
- triggering automated actions 339
- trouble ticket 17
  - severity 263
- trouble ticket ID (ttid) 304, 306
- trouble ticketing 17, 68, 360
  - best practices 69
  - escalating events 279
  - group paging 72
  - IBM Tivoli Enterprise Console 302, 307
  - NetView versus IBM Tivoli Enterprise Console 307
  - on-call list 72
  - prioritization 71
  - rule 236
  - rule set 238
  - system 244
  - system severity 275
- troubleshooting IBM Tivoli Enterprise Console 377
- troubleshoot.rls rule 302, 307, 315
- TroubleTicket.sh 304
- trouble-ticketing integration 302
- trouble-ticketing software 307
- trouble-ticketing system 17, 57, 65–68
  - downstream correlation 306
  - integration with event processor 69
- Truncate\_on\_restart keyword 379
- ttid (trouble ticket ID) 304, 306

## U

- ucd-snmpd 123
- unhandled events 60
- UNKNOWN severity 239
- unmanaged subnet 224
- unsecure events 208
- unwanted unsecure events 208
- update capability, bidirectional 308
- ups\_fatal\_forwarding 299
- upstream correlation 306
- upward synchronization 296
- user interface server 93

## V

- visualization component 104

## **W**

- WAITING event 380
- WARF (Web Application Registration File) 124, 409
- warf directory 409
- warning 61
- Web Application Registration File (WARF) 124, 409
- Web console 114, 156
  - applet 406
  - installation 404
  - menu extension 408
  - menu integration 124
  - NetView 157
  - security 407
  - stand-alone installation 404
- Web event console 94
- Web Health Console 260
- Web-based console 249, 251
- WebSphere Application Server 241
- WebSphere MQ 241
- WebSphere\_MQ\_ChannelNotTransmitting event 243
- Websphere\_MQ\_QueueManagerUnavailable event 243
- WebSphereServers smartset 133
- worsening condition 52, 64, 66
  - escalation 284, 291
- wpostmsg command 191
- wrb command 97–98
- wrb -comprules command 389
- wrb -deldp command 241
- wrb -imptdp command 241
- wsetesvrcfg command 380
- wtdumper 291
- wtdumpri command 377

## **X**

- xecho command 181
- xextra 123
- XFree86-Xvfb 123
- XML file 99
- xvfb 123





# Event Management and Best Practices

(0.5" spine)  
0.475" <-> 0.875"  
250 <-> 459 pages









# Event Management and Best Practices

**Implement and use best practices for event processing**

**Customize IBM Tivoli products for event processing**

**Diagnose IBM Tivoli Enterprise Console, NetView, Switch Analyzer**

This IBM Redbook presents a deep and broad understanding about event management with a focus on best practices. It examines event filtering, duplicate detection, correlation, notification, escalation, and synchronization. Plus it discusses trouble-ticket integration, maintenance modes, and automation in regard to event management.

Throughout this book, you learn to apply and use these concepts with IBM Tivoli® Enterprise™ Console 3.9, NetView® 7.1.4, and IBM Tivoli Switch Analyzer 1.2.1. Plus you learn about the latest features of these tools and how they fit into an event management system.

This redbook is intended for system and network administrators who are responsible for delivering and managing IT-related events through the use of systems and network management tools. Prior to reading this redbook, you should have a thorough understanding of the event management system in which you plan to implement these concepts.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)