



OpenService, Inc. White Paper

NerveCenter™: Integration with Micromuse Netcool/OMNIbus



PublicationDate

Contents

- Introduction 1
- Overview 2
 - What is Netcool/OMNIbus? 2
 - What is NerveCenter? 4
 - How NerveCenter Complements Netcool/OMNIbus 6
- Components Required for Integration 7
 - Micromuse Netcool/OMNIbus Components 8
 - Open NerveCenter Components 8
- How the Integration Components Interact 9
- NerveCenter Configuration Settings 12
 - NerveCenter Server Inform Port Settings 12
 - Universal Platform Adapter Settings 13
 - Inform Action Settings 15
- Netcool/OMNIbus Configuration Settings 16
 - Object Server Data Management Settings 16
 - NerveCenter Probe Settings 17
 - Desktop settings 21
- Appendix A 29
 - Inform Messages 29
 - Inform Data Sent from NerveCenter 29
 - Debug Probe Output 31
- Appendix B 33
 - Sample Rules File 33
 - Sample Nervecenter.rules File. 34

About Open Service, Inc.

Open (OpenService, Inc.) is the premier provider of network security management solutions that enable enterprises and service providers to continuously protect and manage mission-critical business information. More than 450 customers are using Open's SystemWatch and/or NerveCenter™ for network security management. Open's products are available globally through a network of VARs and direct sales. A privately held company based in Westborough, MA, Open is backed by venture capital and an equity stake taken by Veritas. For more information, please call 800-892-3646, or visit the Open web site at <http://www.open.com>.



Introduction

Open NerveCenter™ can forward significant network events to Micromuse Netcool/OMNIBus. Netcool/OMNIBus receives these events and distributes the information to the operators, administrators, help desk systems, or other applications responsible for monitoring the related devices or services.

This document contains the following sections.

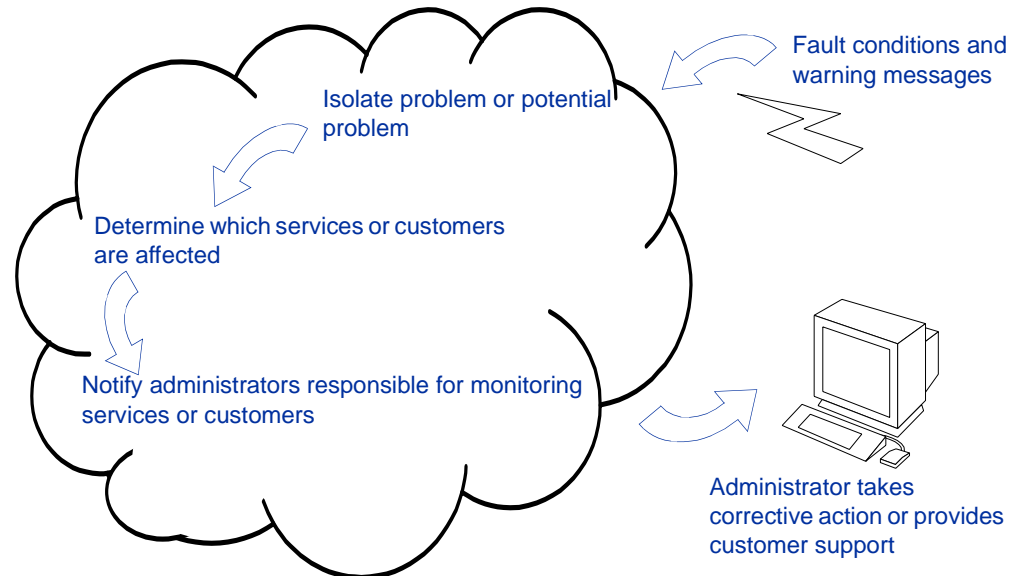
Title	Description
<i>Overview on page 2</i>	Introduces the NerveCenter and Netcool/OMNIBus applications and discusses how NerveCenter complements Netcool/OMNIBus.
<i>Components Required for Integration on page 7</i>	Lists and describes each application's components required for integration.
<i>How the Integration Components Interact on page 9</i>	Describes briefly how NerveCenter and Netcool/OMNIBus communicate with each other.
<i>NerveCenter Configuration Settings on page 12</i>	Explains what can or must be configured in NerveCenter.
<i>Netcool/OMNIBus Configuration Settings on page 16</i>	Explains what can or must be configured in Netcool/OMNIBus.
<i>Inform Messages on page 29</i>	Summarizes the data that NerveCenter sends to Netcool/OMNIBus. Describes how you can confirm that the probe is correctly receiving NerveCenter informs.
<i>Sample Rules File on page 33</i>	Includes a sample nervecenter.rules file provided by Southernview Technologies, Inc.

Overview

Recent trends within the communication industry have given rise to software tools designed to provide immediate support for real-time business and technology services. This type of service-level management is illustrated in Figure 1.

Figure 1. Service-level Management

In the event of an outage, administrators can quickly determine which device caused the problem, which customers and services are impacted, and how the condition affects their service level agreements.



Internet, cellular, network, and other service providers all need a way to ensure the availability of services, devices, and applications in any IT environment. Service-oriented tools provide administrators the status of business and technology services being provided to the users. These tools do not necessarily replace network management platforms; rather, they can extend a platform's capability to provide real-time service-oriented views of the network.

The Micromuse Netcool/OMNIbus suite of tools has emerged as a viable solution for enterprises concerned with the availability, reliability, and quality of direct communication technologies. By partnering with key vendors like Open, Micromuse enables network administrators to organize, measure, predict, and improve service levels for applications, database systems, network devices, and business-critical services such as virtual private networks or Internet connections. In the event of an outage, administrators can quickly determine which device caused the problem, which customers and services are impacted, and how the condition affects their service level agreements.

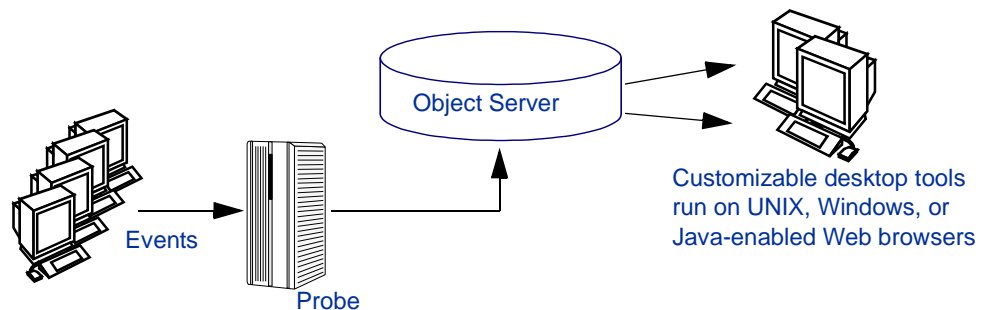
What is Netcool/OMNIbus?

Netcool/OMNIbus is a suite of management tools that collect and distribute network events to the administrators responsible for monitoring related services. Netcool/OMNIbus uses specialized software agents, called probes, to intercept data coming from network systems, devices, and applications. Micromuse has over 100 probes, each designed to identify, collect, and format data from a particular management environment or network application. The Micromuse NerveCenter probe was developed specifically to gather network and system data from NerveCenter servers.

A probe formats events into Netcool alerts and forwards these alerts to the Object Server, Netcool's active database. The Object Server manipulates alerts based on user-defined associations, and groups the alerts into logical units. The Object Server then directs status information to the operators, administrators, help desk systems, or other applications responsible for monitoring the related services. Administrators who receive the alerts can compare each alert against existing service level agreements and determine which services—and ultimately which users—are affected by particular faults. Desktop tools enable administrators to design personalized views of service availability.

Software agents called probes relay network events to the Object Server. The Object Server groups the events and forwards them to Netcool desktop applications.

Figure 2. Netcool/OMNibus



In addition to the Object Server, probes, and desktop tools, Netcool/OMNibus includes the following:

- ♦ Gateways allow event data to be shared with other software programs.
- ♦ Process Control systems enable you to configure and manage UNIX processes remotely. Processes can be started in sequential order after any defined dependencies have been met.
- ♦ Java Event List provides event lists and views of service availability from Web browsers.

Micromuse also provides the following tools:

- ♦ Netcool/Reporter uses data from the Object Server and historical databases to generate graphical reports that can be configured to match service agreement specifications.
- ♦ Netcool/CNMview provides service-level information from remote Desktops at the customer site.
- ♦ Netcool/Internet Service Monitors are active probes that provide availability information about Internet services (HTTP, FTP, DNS, SMTP, POP-3, NNTP, and RADIUS). Each Monitor periodically attempts to access a URL or perform a file transfer. It then reports the time it took to get a response or indicates that the service is unavailable.

What is NerveCenter?

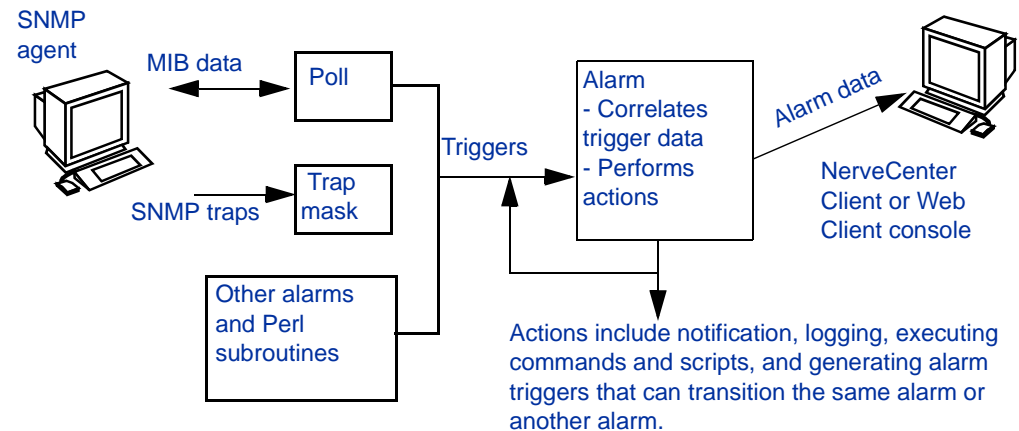
NerveCenter obtains data from SNMP agents running on managed nodes by processing incoming SNMP traps and polling the nodes for specific MIB values. NerveCenter interprets and correlates this data to detect predefined network conditions and determine which actions should be performed.

Behavior Models

To correlate network data, NerveCenter relies on configurable models of network and system behavior, or behavior models, for each type of managed resource. When a predefined network condition is detected, NerveCenter generates alarm instances that track the status of the interface, node, or enterprise being monitored. The alarm waits for subsequent events or issues polls to determine if the condition warrants further action. Each transition in an alarm can trigger actions, including notifying an administrator or a network management platform, executing a program or Perl script, modifying the node's properties, changing SNMP values, and logging the critical data.

A behavior model is a group of NerveCenter objects that detect and handle a particular network or system behavior. A typical behavior model consists of one or more alarms with all their supporting polls and masks.

Figure 3. A NerveCenter Behavior Model

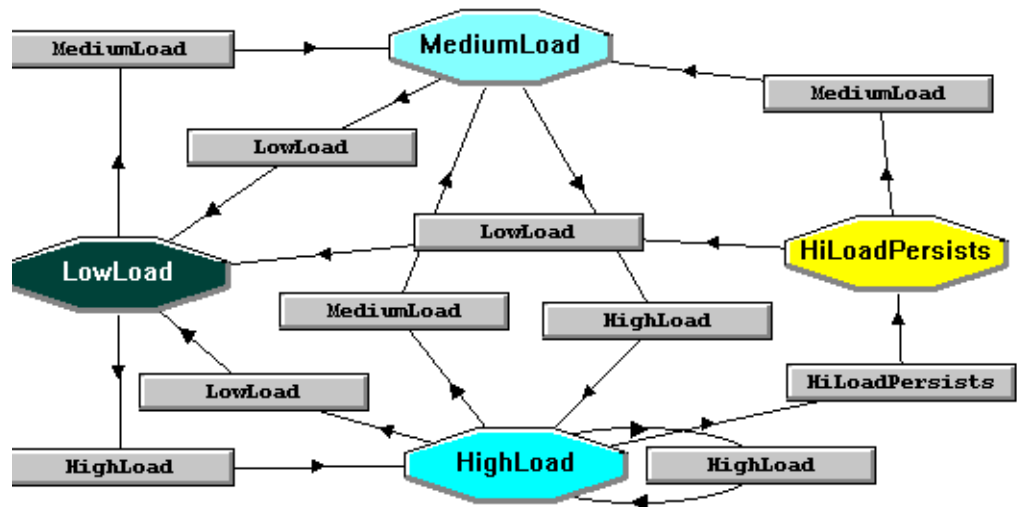


Alarms

Alarms are key to the correlation of events. Each alarm defines a set of operational states (such as Normal or Down) and transitions between the states. Transitions are caused by trigger-generating objects such as polls, trap masks, Perl scripts, or other alarms. When the alarm receives the proper trigger, one or more transitions occur. If actions are associated with a transition, the NerveCenter Server performs these actions each time the transition takes place.

The following diagram illustrates an alarm that monitors each interface on managed nodes and determines whether device load is low, medium, or high. Load is the amount of interface traffic compared to the media's capacity. The *IfLoad* alarm can give an immediate impression of network and system utilization. By measuring traffic against capacity, you can determine, for example, whether more file servers need to be added to the network.

Figure 4. IfLoad Alarm



The alarm transitions to a corresponding state when it receives a MediumLoad trigger or a HighLoad trigger. The HighLoad state fires a trigger after the alarm has received its third HighLoad trigger, transitioning the alarm to the HighLoadPersists state.

Once the alarm has entered the MediumLoad, HighLoad, or HighLoadPersists state, receiving a LowLoad trigger returns the alarm to Low and clears any alarm instances.

Inform Messages

Actions can be associated with alarm transitions. NerveCenter has several actions that notify an administrator, network management platform, or another NerveCenter of an alarm transition. One important notification action is the NerveCenter inform action. An inform message contains the variable bindings associated with the event that caused the alarm to transition.

NerveCenter can send informs to Netcool/OMNIBus when specified events trigger NerveCenter alarms.

NerveCenter sends informs to Netcool/OMNIBus when an alarm transition occurs and the transition includes an inform with Netcool designated as the recipient. The NerveCenter Server forwards the inform data to Netcool's NerveCenter probe, which formats the data and sends alerts to the Object Server.

You can configure NerveCenter to send informs to Netcool when certain network conditions are detected, thereby greatly reducing the number of alerts sent to the corresponding Netcool Event List. For example, in the previous alarm (see Figure 4), NerveCenter sends an inform only when the IfLoad alarm transitions to the HiLoadPersists state.

How NerveCenter Complements Netcool/OMNIBus

NerveCenter extends Netcool's ability to measure, predict, and improve service levels in several ways.

Smart Polling

Netcool/OMNIBus is designed to respond to events, such as SNMP traps, coming from managed resources. NerveCenter not only detects and filters SNMP traps, but it can poll resources at predefined intervals for specific network and system data. This information allows administrators to track network and system behavior and identify potential problems before they occur.

Polling allows administrators to track network and system behavior and identify potential problems before they occur.

NerveCenter uses a feature called smart polling to minimize unnecessary network traffic. With smart polling, NerveCenter issues polls only when the outcome of the poll can trigger an alarm. For example, if a behavior model correlates high traffic followed by high error rates, a device is not polled for error rates unless it fulfills the high traffic condition. Using this same technology, NerveCenter is able to suppress polling to nodes that are unreachable because either they or their parent devices are down.

For service-level management, NerveCenter can help track the following statistical information over local and wide area networks:

- ♦ Who is on the network, what tasks they are performing, and what tools they are using
- ♦ What applications are in use at the application and device layers and how much bandwidth they are consuming
- ♦ What is the total volume of data on different parts of the network at the busiest time of the day and whether this traffic is seasonal
- ♦ How the traffic will grow with time, taking into account increases in the number of devices or users, and changes to the applications used

This information is essential to guarantee application and network service levels to both internal and external customers.

Intelligent Correlation

Event correlation is the mechanism by which NerveCenter evaluates a number of pre-defined events and determines how the events are related, what may have caused them, and whether the condition is serious enough to notify an administrator or take other corrective action. NerveCenter's complex correlation engine filters out redundant or mundane events so that only important messages are sent to probes. Reducing the number of messages sent to probes both facilitates network management and limits network traffic.

By reducing the amount of raw data sent to administrators, event correlation makes it easier for them to identify critical conditions quickly. Event correlation also results in the delivery of important information that administrators can use to establish baselines, monitor thresholds, determine network utilization patterns, track system performance, identify potential bottlenecks and other critical conditions, and plan for future network needs.

Distributed Architecture

NerveCenter's client-server architecture supports distributed polling across large networks. NerveCenter can be configured so that all polling is accomplished on local area networks rather than across a wide area network. Using this capability, you can reduce bandwidth and increase scalability by limiting the information to be monitored for each subnet and the number of nodes to be polled. NerveCenter servers running at remote sites can notify a centrally located NerveCenter Server or management platform of the noteworthy network conditions at those sites. Because the server can run as a daemon on UNIX systems or as a service on Windows, the branch NerveCenters can be managed remotely.

Other Advantages

NerveCenter offers the following additional advantages:

- ♦ NerveCenter has the ability to parse MIB values and obtain the variable bindings related to network events. You can extend the MIB values monitored as new devices are added to your network.
- ♦ NerveCenter's alarm actions include the ability to execute commands and scripts that can remedy the problem that caused the alarm, resulting in further reduction of the number of events that need to be reported to Netcool/OMNIbus.
- ♦ NerveCenter includes a set of predefined behavior models that you can use to monitor and manage your network. These behavior models contain all the required mask, poll, alarm, and property group definitions for basic network management using MIB-II objects. NerveCenter also ships with predefined vendor-specific models for monitoring Cisco, Compaq, and Wellfleet devices.
- ♦ NerveCenter includes a Web Client that makes it easy to monitor alarms from any machine that has a Web browser.
- ♦ NerveCenter includes tools for generating reports about the network.

Components Required for Integration

This section summarizes the main components required for NerveCenter integration with Netcool/OMNIbus. The summary does not attempt to list all the components that you should install for each product. Rather, it describes those components that are involved with NerveCenter-Netcool communication or that should be configured specifically for integration.

Note Make sure you have compatible versions of both product lines before integrating NerveCenter with Netcool/OMNIbus. Contact your sales representative for information about recent versions and patches.

For a full description of all components available for NerveCenter and Netcool/OMNIbus, refer to each product's documentation.

Micromuse Netcool/OMNIBus Components

The Netcool/OMNIBus components listed in Table 1 must be configured for integration:

Table 1. Micromuse Components Required for Integration

Component	Description
Netcool/OMNIBus Object Server	The Object Server is an active database that stores and manages all Netcool events. Events are passed to the Object Server from external programs such as probes and gateways. The Object Server filters out redundant events and make decisions about the data based on user-defined parameters.
Netcool/OMNIBus NerveCenter probe	<p>The NerveCenter probe is distributed by Micromuse and is neither a Open product nor a component of NerveCenter. The probe collects network data from NerveCenter and then formats and forwards that data to the Object Server. The probe has associated rules and properties that define how the probe operates and how NerveCenter events are mapped to Netcool alerts.</p> <p>Note The NerveCenter probe is not included with a standard Netcool/OMNIBus package but must be obtained separately from Micromuse. Probes are identified by the platform and version of NerveCenter you are using.</p>
Netcool/OMNIBus desktop tools	Desktop tools provide event lists and views of service availability. By manipulating the data using Netcool utilities, you can design personalized views of network services and devices managed by NerveCenter.

Open NerveCenter Components

Open NerveCenter is a distributed client/server application that includes a server, files used for storing NerveCenter data, user-interface software, and several additional tools. The NerveCenter components listed in Table 2, all part of the standard package, are required for integration:

Table 2. NerveCenter Components Required for Integration

Component	Description
NerveCenter Server	The Server carries out all the major tasks that NerveCenter performs and manages NerveCenter communications and processes.
Universal Platform Adapter	NerveCenter's Universal Platform Adapter establishes a connection with Netcool/OMNIBus and relays events from the NerveCenter Server to Netcool's NerveCenter probe.
NerveCenter Administrator	After installing NerveCenter, you use the NerveCenter Administrator to configure various settings for the connected Server. These settings include the host name and port number of the machine on which NerveCenter's Universal Platform Adapter resides. This information is required for sending informs to Netcool/OMNIBus.

Table 2. NerveCenter Components Required for Integration (continued)

Component	Description
NerveCenter Client	The NerveCenter Client lets you monitor the network as well as create and modify the behavior models managed by the NerveCenter Server. In the Client, you set up alarm actions that send NerveCenter informs to Netcool when defined network conditions are detected.

How the Integration Components Interact

For integration to occur, NerveCenter must be configured to send informs to Netcool/OMNIBus, and Netcool/OMNIBus must be configured to receive and process those informs. See *NerveCenter Configuration Settings* on page 12 and *Netcool/OMNIBus Configuration Settings* on page 16 for details.

The following must be running:

- ♦ Netcool/OMNIBus Object Server
- ♦ Netcool/OMNIBus NerveCenter probe
- ♦ NerveCenter Server
- ♦ NerveCenter Universal Platform Adapter

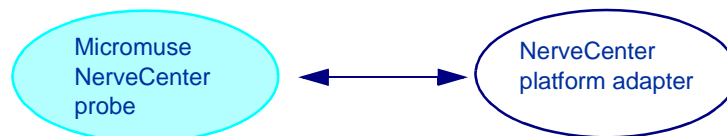
You may also want to start the Netcool/OMNIBus desktop applications, especially the Event List, to view messages and alerts.

Once the applications are running, the probe immediately creates a TCP socket and listens for connections from the NerveCenter platform adapter process. When the adapter sends a connection request, the probe confirms the connection.

The following illustrations show the components and their paths of communication for a simple integration configuration.

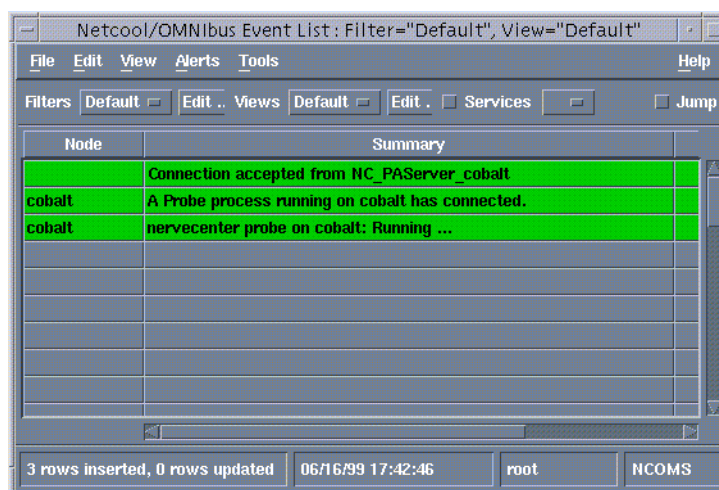
Note The illustrations do not suggest the physical location of the components, which can all reside on the same system or on different systems. However, it's worth noting that installing the NerveCenter probe on the same machine as the NerveCenter Server and platform adapter can reduce network traffic, since all messaging among those components is contained within a single system.

Figure 5. The Probe Connects with NerveCenter's Platform Adapter



When the two applications have established a connection, the probe relays the connection status to the Netcool Object Server, and the Netcool Event List viewer displays the status messages received from the Object Server. The following sample Event List window shows the messages received when the probe is started and connects with the Universal Platform Adapter (paserver.exe).

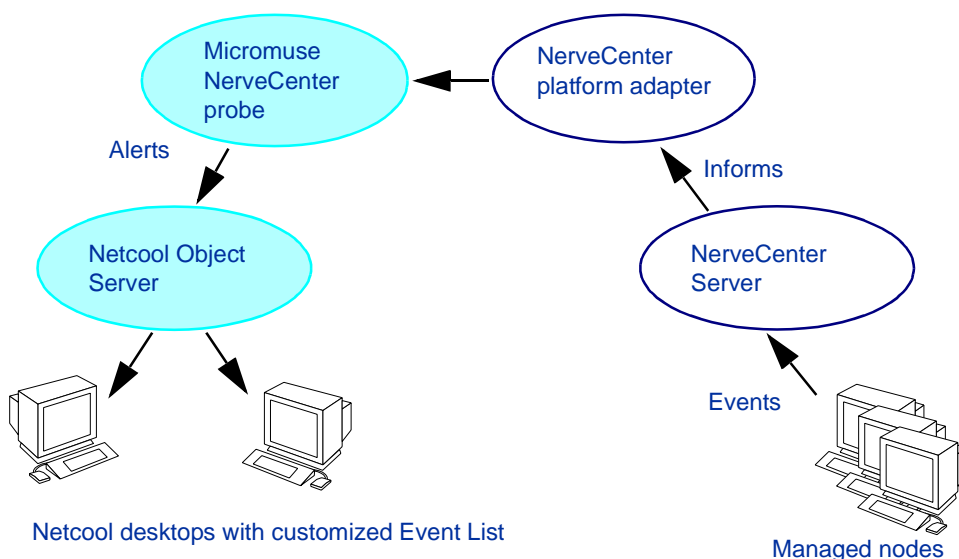
Figure 6. Event List Messages Upon Connection



Node	Summary
	Connection accepted from NC_PAServer_cobalt
cobalt	A Probe process running on cobalt has connected.
cobalt	nervecenter probe on cobalt: Running ...

After the probe and adapter establish their connection, NerveCenter can send informs to Netcool/OMNIBus. The NerveCenter Server issues an inform when a NerveCenter alarm transition occurs and the transition includes an inform action targeted for Netcool/OMNIBus. The Server sends the inform to the platform adapter, which passes the inform data to Netcool's NerveCenter probe.

Figure 7. NerveCenter Sends Informs to Netcool/OMNIBus



The probe receives the inform data and generates an identifier that uniquely identifies the event. The probe also converts the inform to an alert format that the Object Server can recognize and then forwards the alert to the Object Server.

The Object Server stores alerts in an alert table that is part of its active database. The Object Server can manipulate alerts by associating them into classes, filtering them, and assigning automated actions to them. Based on defined parameters, the Object Server determines the destination for each alert and forwards the information to the appropriate desktop applications.

The Event List displays the alerts forwarded from the Object Server, as seen in Figure 8.

Figure 8. Netcool/OMNIBus Event List

Node	Summary	Last Occurrence	Count
	Connection accepted fro	06/16/99 12:39:30	1
ATHENA	See details	06/16/99 12:32:10	1
ATHENA	See details	06/16/99 12:34:51	19
BLUERIDGE	See details	06/16/99 12:32:02	1
BLUERIDGE	See details	06/16/99 12:34:53	20
CAREY	See details	06/16/99 12:34:49	19
CAREY	See details	06/16/99 12:32:08	1
HATTERAS	See details	06/16/99 12:32:08	1
HATTERAS	See details	06/16/99 12:34:49	19
MOZART	See details	06/16/99 12:32:10	1

0 rows inserted, 10 rows updated | 06/16/99 12:40:44 | root | NCOMS

Note You can double-click an alert in the list to see all the information received for a particular inform. See Appendix A, *Inform Messages* for a description of the data sent with NerveCenter informs.

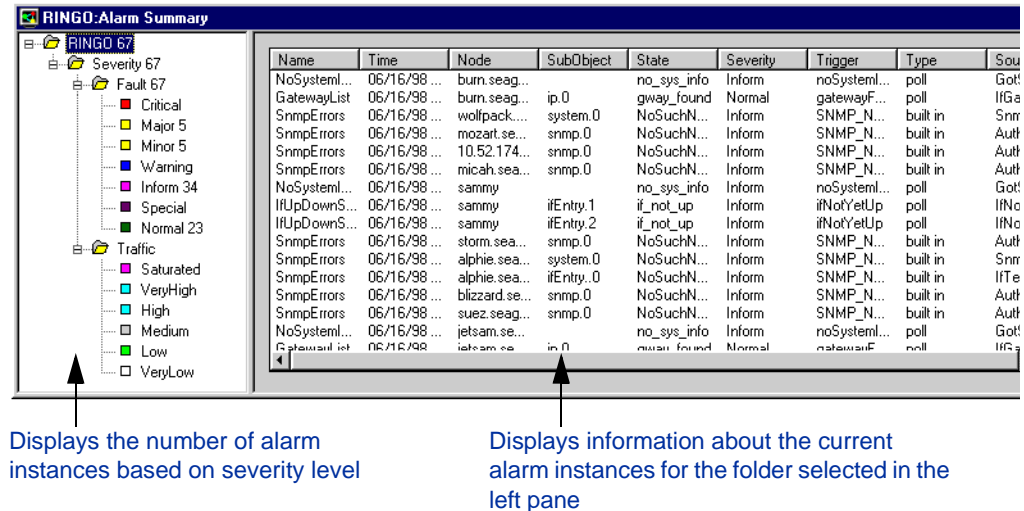
A heartbeat message monitors the connection between the probe and the platform adapter. If the probe goes down or comes back up, the probe's new operational status appears immediately in the Netcool Event List. Once the probe comes back up, the platform adapter attempts to reestablish the connection every minute.

When the platform adapter goes down or comes back up, the following occur:

- The Netcool Object Server immediately sends the status update to the Event List, which displays a message indicating the current connection status.
- After a configurable amount of time, the NerveCenter platform adapter notifies the NerveCenter Server of the status, and the Server sends a message to all connected clients.

While NerveCenter alerts are being monitored in Netcool/OMNIBus, you can also view the original alarms in the NerveCenter Client or Web Client. Figure 9 shows a sample Alarm Summary window in the NerveCenter Client.

Figure 9. NerveCenter Client Alarm Summary Window



The Client has filters that reduce the alarms to those matching specified property groups, severity levels, and IP subnets for associated servers. For more information about monitoring alarms, refer to NerveCenter's *Monitoring Your Network* online guide.

NerveCenter Configuration Settings

This section explains which NerveCenter components are configured to integrate with Netcool/OMNIbus and what should be configured for each component.

NerveCenter's configuration involves specifying the communication ports that allow the NerveCenter Server and adapter to transfer data to Netcool/OMNIbus. Secondly, NerveCenter must be set up to detect noteworthy conditions and send inform actions to Netcool/OMNIbus. Complete procedures for entering these settings are included in the online guides that are shipped with NerveCenter. The guides also describe the rights and privileges required for configuring the NerveCenter applications.

Note Make sure you have the correct version of all components before configuring NerveCenter. Contact your sales representative for information about recent versions and patches. You can download NerveCenter patches from the [Open Web](#) site as long as you have an active maintenance contract.

NerveCenter Server Inform Port Settings

After installation, you enter settings in the NerveCenter Administrator to specify which hosts are to receive NerveCenter informs. When setting up a Netcool/OMNIbus recipient host, you must provide the host name and the port number to use for sending informs to the Universal Platform Adapter. The default port is 32509.

Figure 10 shows the dialog box used for setting up informs.

Figure 10. NerveCenter Inform Configuration

The host and port number entered here must match the host and port number configured for the adapter (see *Universal Platform Adapter Settings* on page 13). If you later change the host or port number associated with the platform adapter, you must change the information entered here in the Administrator.

While setting up the inform configuration, you can specify a minimum severity level for informs or limit the informs to those with particular property groups. Refer to the *Integrating NerveCenter with a Network Management Platform* guide for more information about inform settings.

Universal Platform Adapter Settings

During installation, the platform adapter is configured with default settings that specify the adapter's host machine and the ports used to communicate with the NerveCenter Server and the NerveCenter probe. Depending on your configuration, you may need to change the default settings.

Note If you install the Universal Platform Adapter on a different machine from the one on which the probe is installed, you must change the adapter's default -nhost setting to the machine that is running the probe.

The command to change the platform adapter settings resembles the following:

```
paserver -o -p listeningport -n ON -nhost hostname -nport
sendingport
```


The command contains the switches shown in Table 3:

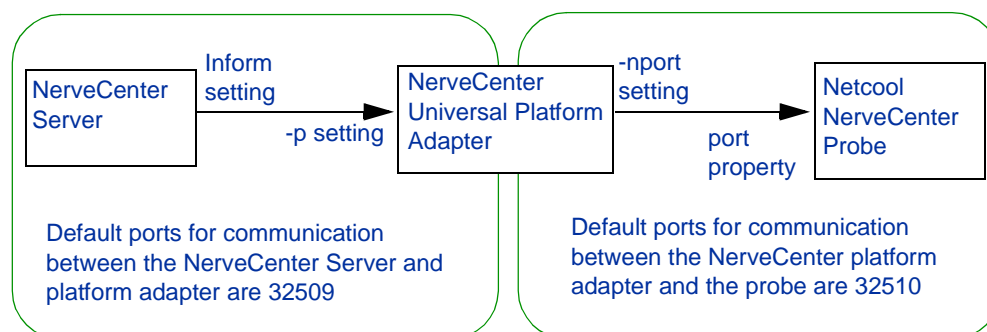
Table 3. Switches for Reconfiguring the NerveCenter Platform Adapter

Switch	Description
-o	Windows only, records values into the registry. Any options (other than -scm) become a part of the standard configuration. To use this switch, you should first stop the Universal Platform Adapter. You must then restart the Universal Platform Adapter.
-p	Defines the platform adapter's listening port. The adapter uses this port to communicate with the NerveCenter Server. Note This number must match the port number specified in NerveCenter Administrator for sending informs. The default is 32509.
-n ON OFF	Enables or disables NerveCenter integration with Micromuse Netcool/OMNIbus.
-nhost	Defines the machine on which the NerveCenter probe is located. Note The default is the local host where the adapter is installed. If the adapter and probe are on different machines, this value must be changed.
-nport	Defines the port the NerveCenter platform adaptor uses to communicate with the probe. The default is 32510. See <i>Nervecenter.props</i> file on page 17 for more information about probe settings. Note This number must match the number used by the probe to communicate with NerveCenter, as specified in the probe's property file. Some early versions of the probe used port 10001; these should be replaced with the current version.

You can also enter settings when you start the platform adapter. For complete procedures and settings, refer to the online book *Integrating NerveCenter with a Network Management Platform* that is shipped with NerveCenter.

Figure 11 summarizes the port settings for NerveCenter-Netcool communication.

Figure 11. Ports for Communication Between NerveCenter and Netcool/OMNIbus



Inform Action Settings

After installation, you can customize or create new behavior models in the NerveCenter Client. The alarms used in behavior models define the types of actions performed when specific network conditions are detected. For each alarm you want to forward to Netcool/OMNIbus, you must define an inform action in the corresponding alarm.

Figure 12 shows a sample alarm along with the dialog box used to define transitions and configure actions for the transitions. Also shown is the pop-up menu containing the available actions.

Figure 12. IfLoad Alarm and Transition Definition Dialog Box

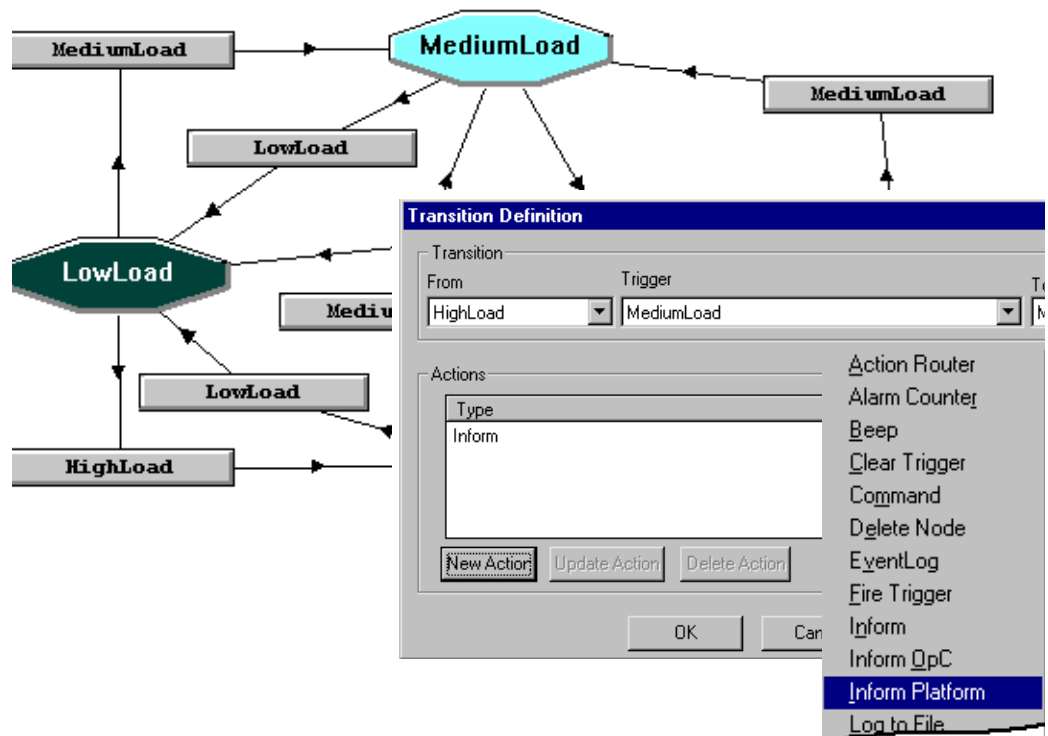
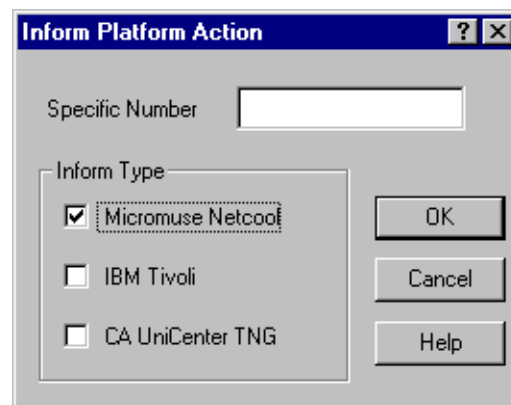


Figure 13 shows the dialog box used to define inform actions for Netcool/OMNIbus.

Figure 13. Inform Action Dialog Box



When creating an inform action, you have the option of providing a specific inform number. This number becomes the \$MesgID value sent with each inform and helps Netcool/OMNIBus identify the type of event. If you don't provide a specific number, the message ID defaults to the value 1000 for any transition whose destination state has a severity less than 9 (Warning). For severity levels of 9 or greater, the \$MesgID defaults to the value 1000 plus the destination state's severity level.

Note Although the message that the inform action sends to its recipients contains the same information as a trap, the message is not sent via UDP. Because the delivery mechanism must be reliable, the message is sent via TCP.

You can define one or more informs for as many alarms as you want. Once you have defined the inform action and enabled an alarm, the inform is sent each time the associated transition occurs. For more information about designing behavior models using the Client, refer to the *Designing and Managing Behavior Models* guide.

Netcool/OMNIBus Configuration Settings

This section explains which Netcool/OMNIBus components are configured to integrate with NerveCenter and what should be configured for each component.

Netcool/OMNIBus configuration involves specifying the type of information you want processed when Netcool receives a NerveCenter inform. You can also specify how the Netcool/OMNIBus Object Server should classify and distribute alerts and any automated actions to be performed.

The \$OMNIHOME directory mentioned in this section is the directory where Netcool/OMNIBus is installed. For example, the default \$OMNIHOME directory for Solaris 2.x and HP-UX 10.x is /opt/Omnibus.

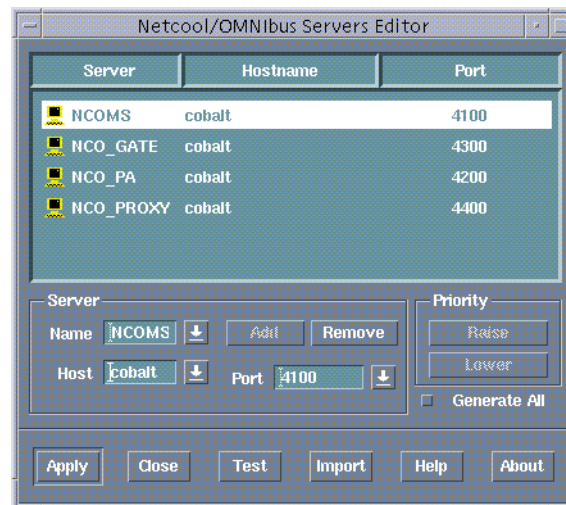
Complete procedures for entering these settings are included in the guides that are shipped with Netcool/OMNIBus. The Micromuse documentation also includes information about the rights and privileges required for access.

Object Server Data Management Settings

After installing Netcool/OMNIBus, you create an interfaces file that specifies the server name, host name, and port for the Object Server, the proxy server, the gateway if installed, and the process control. Without this file, neither the probe nor any other component can communicate with the Object Server.

Any changes to your configuration are made to this file using the Servers Editor (run \$OMNIHOME/bin/nco_xigen).

Figure 14. Servers Editor



NerveCenter Probe Settings

The NerveCenter probe transfers data between the NerveCenter Universal Platform Adapter and the Netcool/OMNIBus Object Server. A probe has two associated files that determine the probe's behavior:

Table 4. NerveCenter Probe Files

File	Description
nervecenter.props	Identifies the probe and displays the location of associated files, the port that connects with the NerveCenter platform adapter, and other information about the probe.
nervecenter.rules	Defines the precise set of information relayed to the Object Server.

Both files are stored in the \$OMNIHOME/probes/*platform* directory. As with the probe, both files are written in a Micromuse proprietary scripting language.

You can design *.props and *.rules files of your own and command the probe to use those files when you start the probe. Refer to the Netcool/OMNIBus *Probe and Gateway Reference* manual for more information about probes. After making changes to these files, you must stop and restart the probe before it can recognize the changes you made.

Nervecenter.props file

The properties file contains default settings for the NerveCenter probe. You can change the properties using the Properties Editor (\$OMNIHOME/bin/nco_xprops). If preferred, you can run the Properties Editor when the Object Server is not running.

Say you wanted to raise the severity level of messages logged from the NerveCenter probe. Messages are logged based on the message and a logging level. There are five message levels:

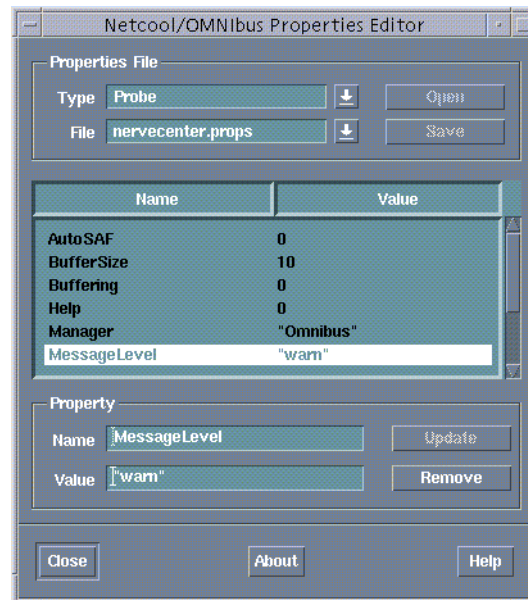
- ♦ Fatal
- ♦ Error
- ♦ Warning

- ◆ Info
- ◆ Debug

To change the minimum severity level of messages that are logged, you could change the message level value from “Warn” to “Error”.

The following illustration shows the nervecenter.props values in the Properties Editor.

Figure 15. Properties Editor



If all the properties are not listed in the Properties Editor, you can open the nervecenter.props file directly using your text editor. In the text editor, you must uncomment a line before any changes you make to the line take effect.

Caution The nervecenter.props file lists the port used by the probe to communicate with the NerveCenter Universal Platform Adapter. For NerveCenter to communicate with Netcool/OMNIBus, this port number must match the -nport setting defined for the NerveCenter Universal Platform Adapter. If they don't match, you must change the adapter setting. See *Universal Platform Adapter Settings* on page 13 for details.

You can set or override some probe property settings from the command line when you start the probe. This is described in the Netcool documentation.

Nervecenter.rules file

The nervecenter.rules file defines the precise set of information relayed to the Object Server. You can modify the rules file to specify how the probe maps NerveCenter events to Netcool/OMNIBus alerts. Prior to reading the file, the probe maps the message's attributes to the fields of an event and creates a list of all attributes and values for insertion into the Object Server's status table. The rules allow you to supersede and change this preformed alert.

The rules file uses tokens to indicate variables, such as the node that caused the message to be sent. A token is identified by the \$ symbol. The @ symbol identifies field values that are transferred to the alerts table in the Object Server database. You can define your own tokens and fields in the rules file.

The default rules file is divided into two main sections, each section is part of an *if .. else* statement. In the first section, the probe generates its own ProbeWatch events to monitor the status of the probe and display messages contingent with the particular case.

Probe Watch Code

The following sample shows this section:

```
if( match( @Manager, "ProbeWatch" ) )
{
  switch(@Summary)
  {
    case "Running ...":
      @AlertGroup = "probestat"
      @Type       = 2
    case "Going Down ...":
      @AlertGroup = "probestat"
      @Type       = 1
    default:
  }
  @AlertKey = @Agent
  @Summary  = @Agent + " probe on " + @Node + ": " + @Summary
}
```

Message Code

The second section of the rules file manages the information transmitted when a connection is established or rejected, when the connection is terminated, and when an inform message is received from NerveCenter.

The following sample shows default statements that determine what appears in the Event List when NerveCenter sends an inform:

```
case "Inform Netcool":
  @Identifier = $MessageType + $ServerID + $NodeName + $IPAddr + $OSN
  + $OSS + $DSN + $DSS + $TrapGN + $TrapSN + $TrapEID
  @Node = $NodeName
  @NodeAlias = $IPAddr
  @Summary = "See details"
  @Severity = 2
```

The **Identifier** field maps to the data sent by NerveCenter for each inform and uniquely identifies the inform event. The **Identifier** field also enables the elimination of duplicate alerts. If the Netcool Object Server receives two informs with the exact same identifier values, only the first inform is forwarded to the Event List. Netcool processes the duplicate inform but does not display it as a separate event instance.

Suggestions for Customizing the Rules File

You can customize the rules file supplied by Micromuse to optimize your own network management strategy. The Netcool/OMNIbus *Probe and Gateway Reference* manual describes a host of statements and functions that help you manage network data sent to the Object Server and relayed to the Event List. To modify inform data, you would first define any tokens and fields you need and then add the statements and functions to the code residing within the Inform Netcool case.

For example, you can filter unnecessary events using the Netcool *Discard* function. When used prudently, this function is an effective way to prevent an inform from being sent to the Object Server. Other functions enable you to recover discarded alarms, compare variables with strings, extract portions of strings or fields, perform mathematical operations, and insert information into an event using a table format consisting of keys and values.

Caution Make a backup copy of the default `nervecenter.rules` file before modifying that file. Changes you make to the rules file affect the probe's compatibility with incoming NerveCenter inform data. As with customizing any software, you should know both the Netcool and NerveCenter products thoroughly and test each change you make to the rules file before proceeding with further changes.

When making changes to a rules file, you must follow the established Netcool syntax. If the syntax in a rules file is incorrect, the probe cannot be started. Netcool/OMNIBus includes a syntax probe that you can use to test the syntax of a rules file.

You may want to make the following changes to the Inform Netcool section of the rules file:

- ♦ Add a comment symbol (#) in front of the line that contains the following text:
`details($*)`

This line of code passes to the Object Server a set of variables (\$) and their values for each inform. These variables are then displayed in the Alert Details portion of an alert in the Event List. While this information is useful for the development and debug of a rules file, the extra data can overload the Object Server once you start processing large numbers of events. (Commenting this line has no affect on the @ field values sent to the Object Server.)
- ♦ If you choose to keep the `details()` statement for one or more types of detected conditions, you may want to change the message associated with the details. By default, messages received from NerveCenter display as "See Details" in the Event List. You can associate more meaningful messages with events by replacing the @Summary value with some other text string or with a variable, such as `$MessageType`.
- ♦ Define a class of alerts that you can later use to group NerveCenter informs. To do this, you associate an arbitrary number with the class you define, for example:

`@Class = number`
- ♦ Change the severity level associated with informs. There are a total of five possible levels. To change the severity from Warning to Major, for example, you would replace the @Severity value with 4.
- ♦ Make more information available to the Event List by adding new fields to the code. The following example would enable you to filter or group alerts by NerveCenter Server:

`@AlertKey = $ServerID`

or, to filter or group alerts by the specific inform number you provided when creating the inform action in NerveCenter, you could enter the following:

`@AlertKey = $MsgID`

- ♦ Change the data associated with informs by changing the variables listed in the @Identifier field.

When changing the @Identifier field values, it's important to make sure the identifier is specific enough to filter unwanted duplicate alerts without overloading network traffic. For example, adding more variables to the identifier code above (see *Message Code* on page 19) would lessen the probability of exact duplicates, resulting in fewer deletions. This change, however, would also generate more messages managed by the Object Server. This may overload the Object Server with events relating to conditions that may in fact be redundant.

On the other hand, an identifier that doesn't contain enough fields might filter out important, non-duplicated events.

For example, say you were to change the identifier definition to the following:

```
@Identifier = $MessageType + $ServerID + $NodeName+ $IPAddr
```

The above code excludes the variables that identify origin or destination state for informs. As a result, the Object Server sends an alert to the Event List upon receipt of an alarm instance for a managed node. The Object Server does not, however, forward alerts for subsequent transitions of the same alarm instance to different destination states. The subsequent informs have data identical to the first instance and are therefore filtered out.

- ♦ Finally, you can enhance or change the summary information for the probe status by changing or adding text in the Probe Watch section of the files.

For a list and description of all the variables that are sent with NerveCenter informs, see *Inform Messages* on page 29. *Sample Rules File* on page 33 contains a sample rules file you can use as a reference.

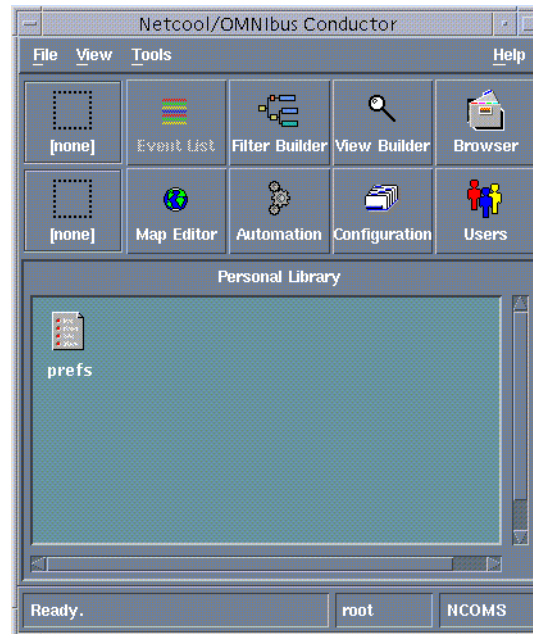
For other ways to configure the rules file, refer to the Netcool/OMNIBus *Probe and Gateway Reference* manual.

Desktop settings

When you start the Netcool/OMNIBus desktop tools, the Conductor is the first window you see. From the Conductor, you can access tools that filter events, customize how the events are displayed in the Event List, associate informs with a particular class, automate commands and actions for informs, and generate service-level views for specified geographical regions.

From the Conductor, you can customize the Event List, create filters, configure classes, and automate actions for incoming NerveCenter alerts.

Figure 16. Netcool/OMNibus Conductor



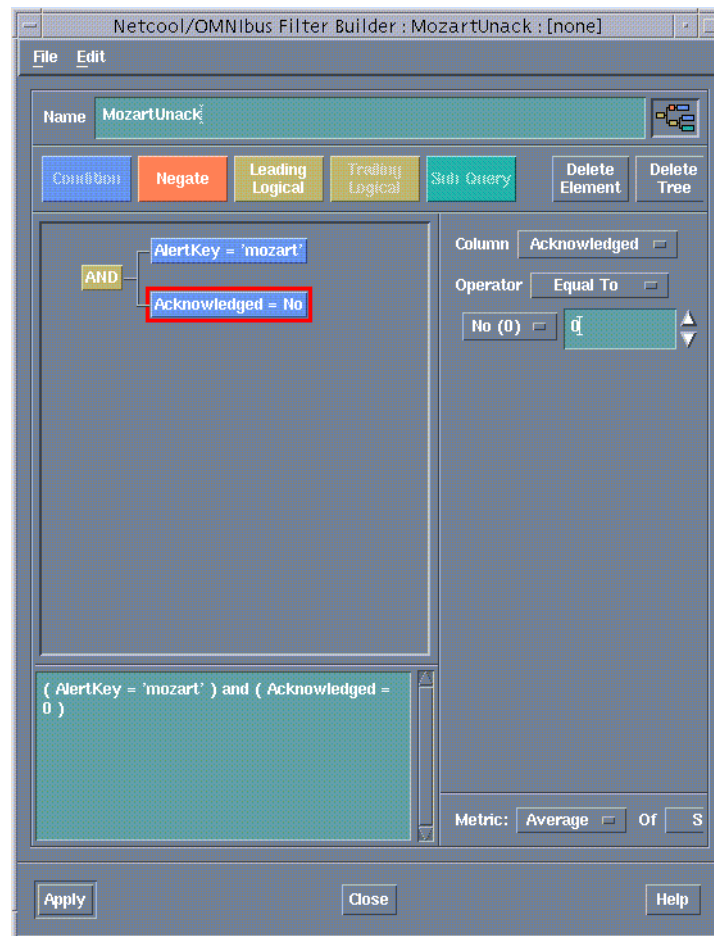
The following sections describe the different ways you may want to customize your desktop for NerveCenter events. Refer to the *Netcool/OMNibus Administration Guide* for complete information about these and any other settings.

Filtered Event Lists

You may want to create a new event list or modify an existing list to display only informs received from NerveCenter. Creating or modifying a list involves creating a new filter that displays certain types of alerts—in this case, informs sent from NerveCenter. Filters limit the information that you receive on your desktop.

The Filter Builder enables you to filter incoming alerts according to the values defined for your informs. Figure 17 shows the Filter Builder used to create filters.

Figure 17. Filter Builder



By stringing together logical OR or AND conditions, you can generate filters based on different combinations of Event List fields.

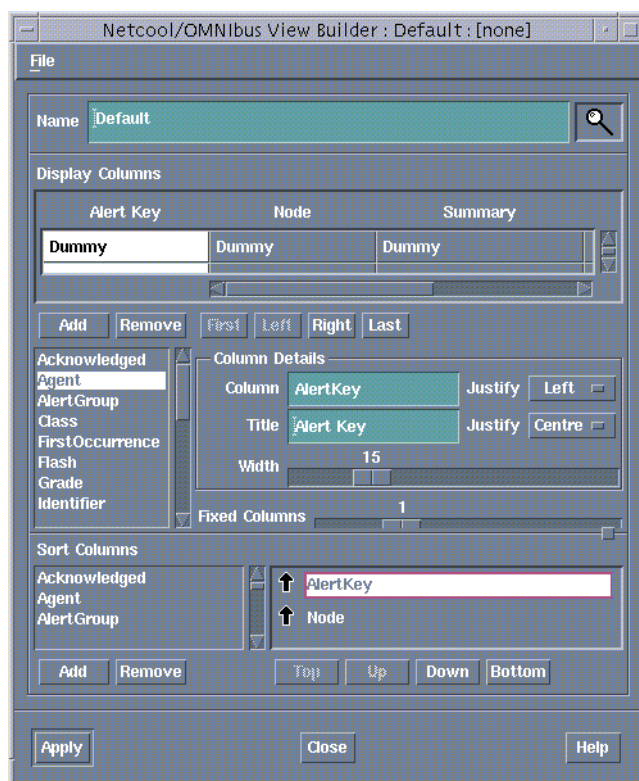
Custom Views

Use the View Builder to customize the columns displayed in the Event List, change the column headers, and organize events by a defined sorting order.

To customize the fields, you choose from the available fields the ones you want displayed in the list. You can include fields, for example **AlertKey**, that you defined at the probe level in the `nervecenter.rules` file.

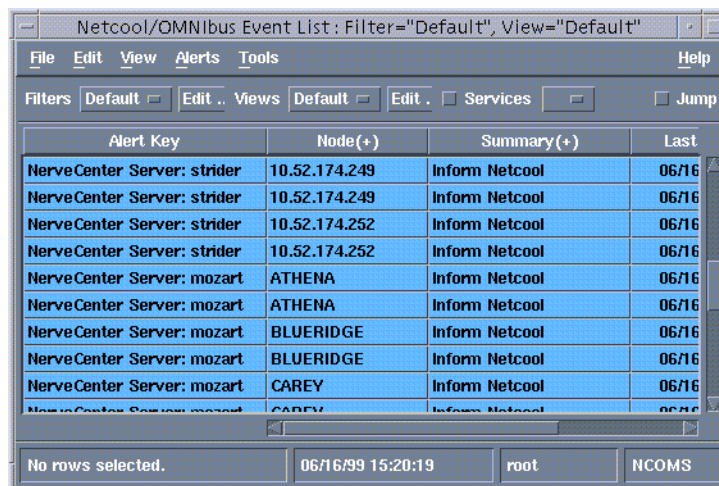
Figure 18 shows sample settings in the View Builder. The settings displayed here add the **AlertKey** field to the Event List and sort by this field, with a secondary sort on the node name.

Figure 18. View Builder



The results of these view settings are shown in Figure 19 in the Event List:

Figure 19. Configured and Sorted Event List



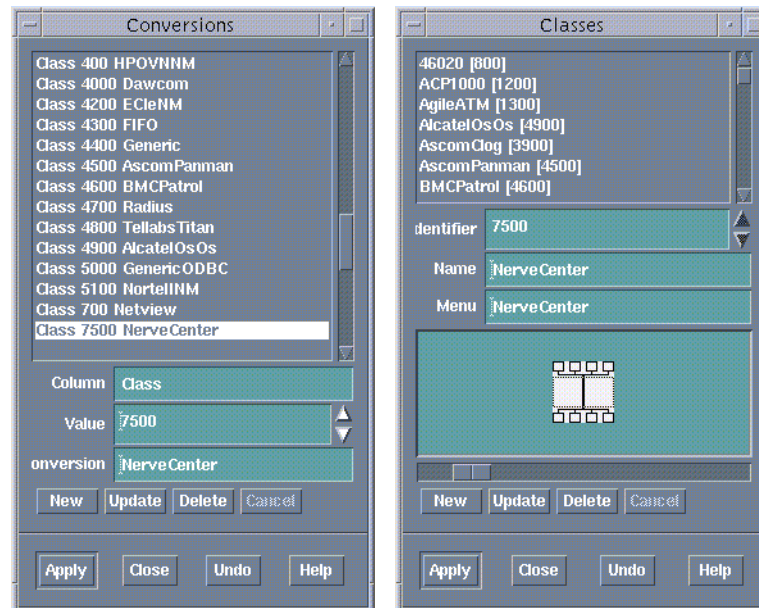
Alert classes

You can tag NerveCenter events with a class value, which is assigned in the nervecenter.rules file. Once you have associated NerveCenter alerts with a specific class, you can create and associate custom menus with the NerveCenter class of alerts. This allows you to automate actions or commands for these events. For menus associated with alerts, the menu options can include commands that reference fields in the alert.

For example, to define a class for NerveCenter events, you would first enter in the nervecenter.rules file the class value, such as `$Class = number`, where *number* is an arbitrary numerical value that you assign. For this example, we'll use the number 7500. Afterward, from the Configuration Manager, you define a conversion for the class and then define the class itself and link it with a menu.

Figure 20 shows the dialogs used to define a conversion and a class.

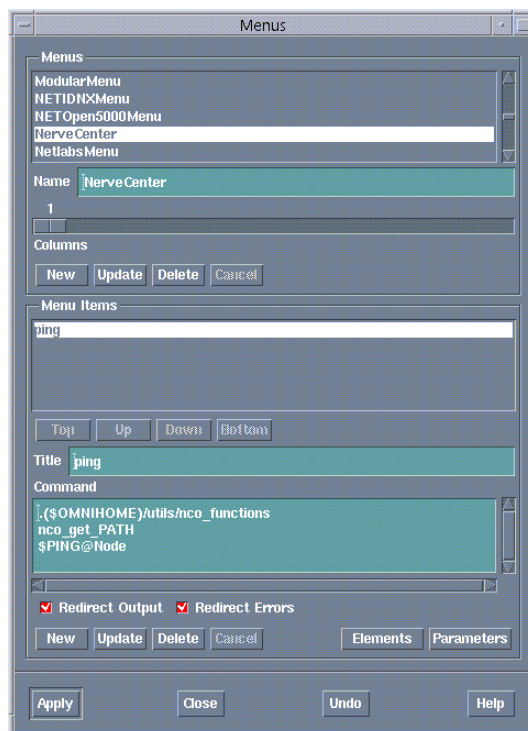
Figure 20. Defining a Conversion and a Class



After defining a class, you can create a menu specifically for that class. If there's a particular action, such as a ping that may need to be performed for NerveCenter alerts, you can define the action as a menu item.

Figure 21 shows the Menus dialog box used to define menu items for a class.

Figure 21. Associating Menu Items with the Class



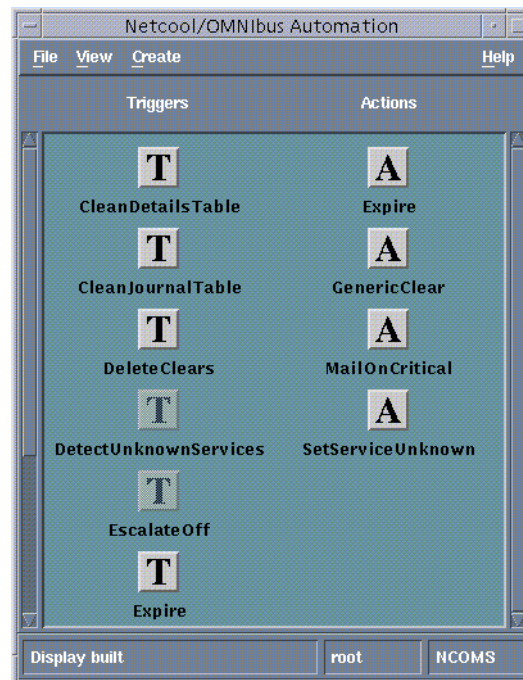
When administrators receive NerveCenter informs tagged with the class you create, they can select the menu items defined for the class from the **Tools** menu of any window in which alerts are displayed.

Automated Actions

The Automation Builder allows you to automate actions or commands for certain types of events. For example, you can notify an administrator of critical events after a specific period of time has elapsed. You do this by creating triggers to detect particular states, for example, an alert severity value of 4, and actions that define the responses to those states. The triggers and states are stored in the Object Server and are created using Object Server SQL.

Figure 22 shows the Automation Builder dialog box used to create and associate actions. You can customize the predefined actions listed in the box or create new actions.

Figure 22. Automation Builder



NerveCenter also provides automated actions that can be performed when an alarm transitions. Besides the inform action described in this paper, NerveCenter has 21 other actions designed for notification, logging data, or correcting a condition.

Objective View Map

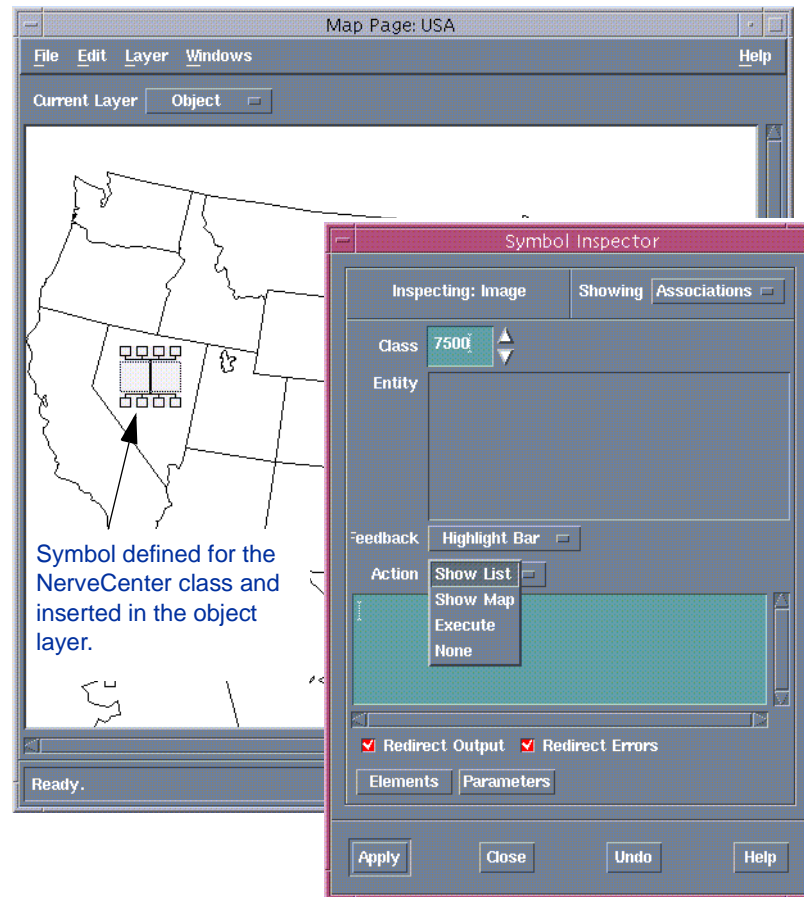
With the Objective View Map Editor, you can map services and devices to geographical regions. This gives you a service-level view of managed objects.

The Objective View displays map books, each containing a number of map pages with graphical objects called symbols. Each page of a book might include symbols representing management sites for a different region of a country or sites in different countries. From these symbols, you can display the status of your services and devices for the region or country.

When creating map pages, layers make it easy to create background-layer images, second-layer annotations, and object-layer editable symbols that can be dynamically manipulated to show status and associations.

Figure 23 shows a page containing a U.S. map in the background layer and a symbol in the object layer. This particular symbol was associated with the NerveCenter class in the Classes dialog box. Double-clicking the symbol displays a dialog in which you can edit the appearance of the symbol and define associations.

Figure 23. Map Page Symbol Associated with a Class and an Action



The action that you associate with a symbol determines what happens when you or someone else double-clicks the icon in the Objective View map. For example, if you select the **Show List** option, double-clicking the icon in the Objective View starts the Event List with the filter and view of the associated entity.

The class you associate with a symbol determines the appropriate **Tools** menu for the symbol when viewed in the Objective View.

Appendix A

Inform Messages

Inform messages, like all other NerveCenter and Netcool/OMNIbus communications, use the TCP protocol. For every inform, the probe establishes a secure socket connection with the NerveCenter Universal Platform Adapter. All messages share a format consisting of a timestamp, a common header, and message data specific to the type of message.

NerveCenter has predefined variables that it sends to the probe when processing an inform action. You can run the probe in debug mode to confirm that the probe is receiving the correct data from NerveCenter.

This appendix contains the following sections:

- ♦ *Inform Data Sent from NerveCenter* on page 29
Lists and describes the variables sent with NerveCenter inform messages.
- ♦ *Debug Probe Output* on page 31
Describes how you can confirm that the probe is correctly processing NerveCenter informs.

Inform Data Sent from NerveCenter

The Inform Netcool message is the mechanism that NerveCenter uses to send events to Netcool/OMNIbus. The following table describes the data sent with NerveCenter informs. The first three items are contained in the inform header. The remaining items are listed alphabetically.

Note Variables that are included by default in the **Identifier** field of the `nervecenter.rules` file are listed as a Default Identifier value..

Table 1. Inform Data

Variable	Description
\$LDT	Local date timestamp.
\$MessageType	The type of message being sent is Inform Netcool. There are seven possible types of messages: <ul style="list-style-type: none">♦ Connection Request♦ Connection Accepted♦ Connection Rejected♦ Exit Notification♦ Heartbeat Query♦ Heartbeat Response♦ Inform Netcool
\$ServerID	The unique identifier for the NerveCenter Server that manages the current alarm transition and alert. The identifier consists of hostname or IP address. This identification becomes important when there are multiple NerveCenter Servers sending informs to the same Universal Platform Adapter.
\$AlrmDN	The alarm definition name for the alarm that transitioned and generated the inform.

Table 1. Inform Data (continued)

Variable	Description
\$AlrmProp	The property assigned to the alarm that transitioned and generated the inform.
\$BOI	The base object instance for the interface that triggered the transition and generated the inform. If the base object is associated with an interface and, therefore, is listed as a table in the MIB .ASN1 file, the instance corresponds to a row in that table.
\$BON	The base object name for the SNMP base object that triggered the transition and generated the inform.
\$DSN	Default Identifier value. The destination state name (the name of the state to which the alarm transitioned).
\$DSS	Default Identifier value. The destination state severity (the severity level associated with the state to which the alarm transitioned).
\$IPAddr	Default Identifier value. The IP address of the node that caused the event. If the node has more than one IP address, the number provided denotes the IP address associated with the event.
\$MesgID	Default Identifier value. The specific number that you enter into the NerveCenter Client alarm definition when you define the Netcool/OMNIbus inform action. If no number is entered, the message ID defaults to the value 1000 for any transition whose destination state has a severity less than 9 (Warning). For severity levels of 9 or greater, the \$MesgID defaults to the value 1000 plus the destination state's severity level.
\$NodeName	Default Identifier value. The name of the node that caused the event to be sent. The name consists of hostname or IP address.
\$NPG	The node property group assigned to the node associated with the inform.
\$OSN	Default Identifier value. The originating state name (the name of the state from which the alarm transitioned).
\$OSS	Default Identifier value. The originating state severity (the severity level associated with the state from which the alarm transitioned).
\$ROCom	The read only community string of the node associated with the inform.
\$RWCom	The read-write community string of the node associated with the inform.
\$TrapEID	Default Identifier value. The trap's enterprise ID. If the transition was caused by a trigger fired from an SNMP trap, the enterprise ID is included here. If the transition was not from a trap, this variable is empty.
\$TrapGN	Default Identifier value. The trap's generic number. If the transition was caused by a trigger fired from an SNMP trap, the generic trap number is included here. If the transition was not from a trap, this variable defaults to -2.
\$TrapSN	Default Identifier value. The trap's specific number. If the transition was caused by a trigger fired from an SNMP trap, the specific trap number is included here. If the transition was not from a trap, this variable defaults to -2.
\$TrigName	The name of the trigger that caused the state transition.

Table 1. Inform Data (continued)

Variable	Description
\$VarBinds (n)	Variable binding pair for the nth variable binding, in text format: attribute = value

Debug Probe Output

By running the probe in debug mode, you can confirm that the probe is receiving the correct data from NerveCenter. This helps you establish that:

- ♦ NerveCenter is communicating with the probe—it's especially important to determine this when NerveCenter and Netcool components are installed on different machines.
- ♦ The probe is receiving the correct information from NerveCenter—this helps identify whether you have the correct versions of both the probe and NerveCenter.

The command for running the probe in debug mode is:

```
$OMNIHOME/probes/nco_p_nervecenter -messagelevel debug &
```

This command forces Netcool/OMNIBus to log the parsed values received from the probe to a file named *nervecenter.log*, located in the \$OMNIHOME directory.

The NerveCenter inform variables are logged each time NerveCenter issues an inform to the probe. Each log entry is appended to the *nervecenter.log* file. The log file lists each variable received from NerveCenter along with the variable's current value. You can compare the variables and values against the NerveCenter inform data for each of your inform actions.

Appendix B

Sample Rules File

The `nervecenter.rules` file is the starting point for customizing the Netcool/OMNIbus desktop, eliminating duplicate alerts in the Event List, and associating events with fields that can be manipulated by the Object Server. This appendix contains a customized `nervecenter.rules` file.

The sample rules file was modified to do the following:

- ♦ Define lookup tables for adding information to an event. The tables for this example are located in `$OMNIHOME/probes/platform` and are accessed using the `lookup` keyword.
- ♦ Create generic variable token definitions that can be assigned to database fields.
- ♦ Assign default values to database fields to ensure all fields contain a value. The rules file later passes these fields to the Object Server when the probe receives certain types of events.
- ♦ Create several different types of classes, along with associated token variables. The class value is set depending on the type of event. This field is also used to associate tools and actions in the Netcool desktop tool.
- ♦ Parse the node names, which follow a predefined set of naming conventions, and extract certain values that can be expanded into more readable text.
- ♦ Define separate cases for events and make field assignments for each case. Each case is based on an incoming `$MesgID` value, which the user provides when creating the inform action in NerveCenter.
- ♦ Modify the **Identifier** field and add a value for the **AlertKey** field. The **AlertKey** field forwards to the Object Server the name of the alarm, node, and interface if applicable for an inform.

If you intend to develop your own `nervecenter.rules` file, you might first map out the message you want for a single NerveCenter alarm and then create the case statement for that `$MesgID` value. This makes it easy to format the **@Summary** field to include the information you want displayed to the operator. The rules file can grow quite large; working with one alarm case at a time makes customization easier.

Sample Nervecenter.rules File

The sample that follows shows the complete rules file.

Caution Make a backup copy of the default nervecenter.rules file before modifying it. Changes you make to the rules file affect the probe's compatibility with incoming NerveCenter inform data. As with customizing any software, you should know both the Netcool and NerveCenter products thoroughly and test each change you make to the rules file before proceeding with further changes.

When making changes to a rules file, you must follow the established Netcool syntax. If the syntax in a rules file is incorrect, the probe cannot be started. Netcool/OMNIBus includes a syntax probe that you can use to test the syntax of a rules file.

```
#####
#      Copyright (C) 1998 Omnibus      Transport Technologies Ltd.
#      All Rights Reserved
#      RESTRICTED RIGHTS:
#      This file may have been supplied under a license.
#      It may be used, disclosed, and/or copied only as permitted
#      under such license agreement. Any copy must contain the
#      above copyright notice and this restricted rights notice.
#      Use, copying, and/or disclosure of the file is strictly
#      prohibited unless otherwise provided in the license agreement.
# Ident: $Id: nervecenter.rules 1.1 1998/07/07 09:23:42 nic Development $
#####
table interfaces= "/opt/Omnibus/probes/hpux10/Tables/interfaces.lookup"
    default="NoMatch"
table fr= "/opt/Omnibus/probes/hpux10/Tables/fr.lookup"
    default="NoMatch"
table bgp= "/opt/Omnibus/probes/hpux10/Tables/bgp.lookup"
    default="NoMatch"
table junction= "/opt/Omnibus/probes/hpux10/Tables/junction.lookup"
    default="NoMatch"
table abbreviations = "/opt/Omnibus/probes/hpux10/Tables/abbreviations.lookup"
    default="9999"
table supportdefinitions =
"/opt/Omnibus/probes/hpux10/Tables/supportdefinitions.lookup"
    default="0"
table exceptionscity = "/opt/Omnibus/probes/hpux10/Tables/exceptionscity.lookup"
    default="Unknown"
table exceptionsdevfun = "/opt/Omnibus/probes/hpux10/Tables/exceptionsdevfun.lookup"
    default="Unknown"
#####
# Other Generic Definitions
$DISPLAYON          = 1
$DISPLAYOFF         = 0
$DONOTFORWARDEVENT  = 0
$FORWARDEVENT       = 1
$ACKFORWARDEVENT    = 2
$UNDEFINED_SERVICE  = 999
$UNDEFINED_CLASS    = 9999
$NULL               = 0
$UNKNOWN            = "Unknown"
$UNDEFINED          = "Undefined"
```

```

$CUSTOMER                = "xxx"
$STARTEVENT               = 1
$ENDEVENT                 = 0
#
# Put ProbeWatch Specific messages here, ie to customise Agent names
# !! This is not part of management system event processing
#
if( match( @Manager, "ProbeWatch" ) )
{
    switch(@Summary)
    {
        case "Running ...":
            @AlertGroup = "probestat"
            @Type       = 2
            @Rise       = $STARTEVENT
        case "Going Down ...":
            @AlertGroup = "probestat"
            @Type       = 1
            @Rise       = $ENDEVENT
        default:
    }
    @AlertKey = @Agent
    @Summary  = @Agent + " probe on " + @Node + ": " + @Summary
} else
{
    switch($MessageType)
    {
        case "Connection Accepted":
            @Identifier = $MessageType + $ServerID
            @Summary    = "Connection accepted from " + $ServerID
            @Severity    = 0

        case "Connection Rejected":
            @Identifier = $MessageType + $ServerID
            @Summary    = "Connection rejected from " + $ServerID
            @Severity    = 3

        case "Exit Notification":
            @Identifier = $MessageType + $ServerID
            @Summary    = "Exit notification received from " + $ServerID
            @Severity    = 1

        case "Inform Netcool":
            @Agent      = "NerveCenter-site"
            @Node       = $NodeName
            @NodeAlias  = $IPAddr
            @Summary    = "?: " + $MesgID
            @Manager    = "manager1"
            @Severity    = 1
            @Type       = 1
    }
}

#####
#
# The Informs start here for main rules section (non ProbeWatch alerts).
# !! This IS where management system event processing starts
#
# Default User specific fields

```

```

@RemedyFlag      = $DONOTFORWARDEVENT
@ProcessedFlag= $NULL
@LoggedFlag      = $NULL
@DatabaseFlag    = $DONOTFORWARDEVENT
@AlertClass      = $NULL
@DevFuncCode     = $UNKNOWN
@CustomerCode    = $UNKNOWN
@CityCode        = $UNKNOWN
@Location        = $UNKNOWN
@Class           = $UNDEFINED_CLASS
@DisplayFlag     = $DISPLAYOFF
@Interface       = $UNDEFINED
@Rise            = $STARTEVENT
#
# SupportClass is a reference indicating the Support for a particular event
#
$UNDEFINED_SUPPORT      = "0"
$INTERNAL                = "1"
$REMOTE                  = "2"
$BUSINESS                = "3"
$OPERATIONS              = "4"

@SupportClass = $UNDEFINED_SUPPORT
#
# VendorClass is defined by the division of the Class by the value of 100
# VendorClass table will look like:
#
#      10      Cisco
#      11      Bay
#      99      Other
#
$CISCO                = "10"
$BAY                  = "11"
$OTHER                = "99"

@VendorClass          = $OTHER
#
# AlertClass Definitions
$LINKFAILURE          = "LinkFailure"
$ROUTINGFAILURE        = "RoutingFailure"
$COMPONENTFAILURE      = "CompentFailure"
$ENVFAILURE            = "EnvironmentalFailure"
$NODEFAILURE           = "NodeFailure"
$PERFORMANCE           = "Performance"
$OTHERCLASS= "Other"

# EventType Definitions
$TRAFFICDROP= "TrafficDrop"
$TRAFFICLOAD= "TrafficLoad"
$PACKETLOSS           = "PacketLoss"
$BGPFAILURE            = "BGPFailure"
$SERVICEFAILURE       = "ServiceFailure"
$MEMORYFAILURE         = "MemoryFailure"
$LINKDOWN              = "LinkDown"
$NODEDOWN              = "NodeDown"
$SECURITY              = "Security"
$CPUFFAILURE           = "CPUFailure"

```

```

$LINKERROR                = "LinkError"
$SNMPFAILURE               = "SnmpFailure"

# If the Node Name is NOT an IP address - parse out info from the name
#
if (regmatch(@Node, "^[a-zA-Z][a-zA-Z][a-zA-Z][0-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z]")) {
    @DevFuncCode           = extract(@Node, "([a-zA-Z][a-zA-Z][a-zA-Z]+)")
    $junction= lookup(@Node,junction)
    if (match($junction,"NoMatch")) {
        @SupportClass      = lookup (@DevFuncCode, supportdefinitions)
    } else {
        @SupportClass      = lookup ($junction, supportdefinitions)
    }
    @Class                 = lookup (@DevFuncCode, abbreviations)
    @CustomerCode          = "abc"
    @CityCode              = extract(@Node, ".*([a-zA-Z][a-zA-Z][a-zA-Z])")
    @VendorClass           = int(@Class)/100
    @Location= $RWCom
    @DisplayFlag           = $DISPLAYON
} else

#
if (regmatch(@Node, "^[a-zA-Z][a-zA-Z][0-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z]")) {
    @DevFuncCode           = extract(@Node, "([a-zA-Z][a-zA-Z]+)")
    @CustomerCode          = "abc"
    @CityCode              = extract(@Node, ".*([a-zA-Z][a-zA-Z][a-zA-Z])")
    @Class                 = lookup (@DevFuncCode, abbreviations)
    @VendorClass           = int(@Class)/100
    @SupportClass          = lookup (@DevFuncCode, supportdefinitions)
    @Location= $RWCom
    @DisplayFlag           = $DISPLAYON
} else

#
if (regmatch(@Node, "^[a-zA-Z][a-zA-Z][a-zA-Z][0-9][a-zA-Z][a-zA-Z][a-zA-Z][a-zA-Z]")) {
    @DevFuncCode           = extract(@Node, "([a-zA-Z][a-zA-Z][a-zA-Z]+)")
    @CustomerCode          = "abc"
    @CityCode              = extract(@Node, ".*([a-zA-Z][a-zA-Z][a-zA-Z])")
    @Class                 = lookup (@DevFuncCode, abbreviations)
    @VendorClass           = int(@Class)/100
    @SupportClass          = lookup (@DevFuncCode, supportdefinitions)
    @Location= $RWCom
    @DisplayFlag           = $DISPLAYON
} else

#
if (regmatch(@Node, "^yes[0-9][0-9][0-9]i[0-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z]")) {
    @DevFuncCode           = extract(@Node, "([a-zA-Z][a-zA-Z][a-zA-Z]+)")
    @CustomerCode          = "abc"
    @CityCode              = extract(@Node, ".*([a-zA-Z][a-zA-Z][a-zA-Z])")
    @Class                 = lookup (@DevFuncCode, abbreviations)
    @VendorClass           = int(@Class)/100
    @SupportClass          = lookup (@DevFuncCode, supportdefinitions)
    @Location= $RWCom
    @DisplayFlag           = $DISPLAYON
} else

```

```

{
#
# Pick up the odd nodes
  @DevFuncCode      = lookup(@Node, exceptionsdevfun)
  $junction= lookup(@Node,junction)
  if (match($junction,"NoMatch")) {
    @SupportClass    = lookup (@DevFuncCode, supportdefinitions)
  } else {
    @SupportClass    = lookup ($junction, supportdefinitions)
  }
  @CustomerCode      = "abc"
  @CityCode          = lookup(@Node, exceptionscity)
  @Class             = lookup (@DevFuncCode, abbreviations)
  @VendorClass       = int(@Class)/100
  @Location= $RWCom
  @DisplayFlag       = $DISPLAYON
}

#
# Check to see if @Class was not set (Normally because not found in any lookup
if (int(@Class) == 0) {
  @Class             = $UNDEFINED_CLASS
  @VendorClass       = $OTHER
}

# Next Enterprise: NetLabs_NerveCenter .1.3.6.1.4.1.78
switch($MesgID)
{
case "3004":
  # NC_alarm1
  @AlertKey = $AlrmDN + $NodeName
  @AlertGroup = $AlrmDN + "FreeBusy"
  @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " CPU Utilization
back to normal"
  @Severity = "2"
  @AlertClass = $COMPONENTFAILURE
  @EventType = $CPUFAILURE
  @Rise = $ENDEVENT
case "3007":
  # NC_alarm2
  details($VarBind1)
  @AlertKey = $AlrmDN + $NodeName
  @AlertGroup = $AlrmDN + "FreeBusy"
  @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " CPU Utilization
    >= 75% <= 90%"
  @Severity = "3"
  @AlertClass = $COMPONENTFAILURE
  @EventType = $CPUFAILURE
  @Rise = $STARTEVENT
case "100000":
  # NC_alarm3
  @AlertKey = $AlrmDN + $NodeName
  @AlertGroup = $AlrmDN + "UpDown"
  @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " unreachable."
  @Severity = 4
  @AlertClass = $NODEFAILURE
  @EventType = $NODEDOWN
}

```



```

        @Rise                = $STARTEVENT
case "100001":
    # NC_alarm4
    @AlertKey = $AlrmDN + $NodeName
    @AlertGroup = $AlrmDN + "UpDown"
    @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " unreachable.
        Problem with network path to node."
    @Severity        = 4
    @AlertClass      = $NODEFAILURE
    @EventType       = $NODEDOWN
    @Rise            = $STARTEVENT
case "100003":
    # NC_alarm5
    @AlertKey = $AlrmDN + $NodeName
    @AlertGroup = $AlrmDN + "UpDown"
    @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " Down."
    @Severity        = 5
    @AlertClass      = $NODEFAILURE
    @EventType       = $NODEDOWN
    @Rise            = $STARTEVENT
case "100004":
    # NC_alarm6
    @AlertKey = $AlrmDN + $NodeName + $INTERFACE
    @AlertGroup = $AlrmDN
    @Summary = "NC: " + $AlrmDN + ": High Error Rate (>5%) on NODE: " +
        $NodeName + " interface " + $INTERFACE + "."
    @Severity        = 4
    @AlertClass      = $PERFORMANCE
    @EventType       = $LINKERROR
    @Rise            = $STARTEVENT
case "100008":
    # NC_alarm7
    details($VarBind1)
    @AlertKey = $AlrmDN + $NodeName + $INTERFACE
    @AlertGroup = $AlrmDN + "UpDown"
    $LookupKey = $NodeName + $INTERFACE
    $SpecialInfo = lookup ($LookupKey, interfaces)
    if(match($SpecialInfo,"NoMatch")) {
        $Exclamation = ""
        $SpecialInfo = ""
        @Severity        = 4
    } else {
        $Exclamation = "Hot!! "
        @Severity        = 5
    }
    @Summary = $Exclamation + "NC: " + $AlrmDN + ": Node " + $NodeName +
        " Interface: " + $INTERFACE + " is Down " + $SpecialInfo
    @Interface      = $INTERFACE
    @AlertClass      = $LINKFAILURE
    @EventType       = $LINKDOWN
    @Rise            = $STARTEVENT
case "100011":
    # NC_alarm8
    details($VarBind1)
    @AlertKey = $AlrmDN + $NodeName + $INTERFACE
    @AlertGroup = $AlrmDN + "UpDown"
    $LookupKey = $NodeName + $INTERFACE

```

```

$SpecialInfo = lookup ($LookupKey, interfaces)
if(match($SpecialInfo,"NoMatch")) {
    $Exclamation = ""
    $SpecialInfo = ""
    @Severity      = 4
} else {
    $Exclamation = "Hot!! "
    @Severity      = 5
}
@Summary = $Exclamation + "NC: " + $AlrmDN + ": Node " + $NodeName +
"Interface: " + $INTERFACE + " is Flapping " + $SpecialInfo
@Interface      = $INTERFACE
@AlertClass      = $LINKFAILURE
@EventType       = $LINKDOWN
@Rise            = $STARTEVENT
case "100016":
    # NC_alarm9
    @AlertKey = $AlrmDN + $NodeName + $INTERFACE
    @AlertGroup = $AlrmDN + "UpDown"
    @Summary = "NC: " + $AlrmDN + ": Node: " + $NodeName + " Session: " +
    $INTERFACE + " is Up."
    @Severity      = 2
    @AlertClass      = $ROUTINGFAILURE
    @EventType       = $BGPFFAILURE
    @Rise            = $ENDEVENT
case "100017":
    # NC_alarm10
    @AlertKey = $AlrmDN + $NodeName
    @AlertGroup = $AlrmDN + "Reboot"
    @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName + " has rebooted."
    @Severity      = 3
    @AlertClass      = $NODEFAILURE
    @EventType       = $NODEDOWN
    @Rise            = $STARTEVENT
case "100063":
    # NC_alarm11
    details($*)
    @AlertKey = $AlrmDN + $NodeName
    @AlertGroup = $AlrmDN + "LowOK"
    @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName +
    "Low Memory! Current " + $VarBind1
    @Severity      = 4
    @AlertClass      = $PERFORMANCE
    @EventType       = $MEMORYFAILURE
    @Rise            = $STARTEVENT
case "100064":
    # NC_alarm12
    details($*)
    @AlertKey = $AlrmDN + $NodeName
    @AlertGroup = $AlrmDN + "LowOK"
    @Summary = "NC: " + $AlrmDN + ": NODE: " + $NodeName +
    "Returned to normal memory utilization. Current memory = " + $VarBind1
    @Severity      = 2
    @AlertClass      = $PERFORMANCE
    @EventType       = $MEMORYFAILURE
    @Rise            = $ENDEVENT
default:

```

```

        @Summary = "Unknown specific trap number (" + $MesgID + ") received for
            enterprise " + $enterprise-name
        @Severity = 1
        # details ($TrapEID, $MesgID, $ServerID, $NodeName, $INTERFACE, $NPG,
            $AlrmDN, $AlrmProp)
    }
#####
#
# This is the end of the main rules section
#
#####
#
# The identifier is built here to take into account the AlertKey
#
@Identifier = $ServerID+": "+$NodeName+": "+$MesgID+": "+@AlertKey+": "+@Rise
@Initial_Severity = @Severity

    default:
    }

# details($*)
}

```

