HP OpenView Operations Developer's Toolkit

Application Integration Guide

Software Version: A.08.10

UNIX

Manufacturing Part Number: B7492-90009

September 2004

Legal Notices

Warranty.

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices.

Trademark Notices.

Adobe® is a trademark of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Intel386, Intel80386, Intel486 , and Intel80486 are U.S. trademarks of Intel Corporation.

Intel Itanium ™ Logo: Intel, Intel Inside and Itanium are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries and are used under license.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

MS-DOS® is a U.S. registered trademark of Microsoft Corporation.

Netscape™ and Netscape Navigator™ are U.S. trademarks of Netscape Communications Corporation.

OpenView® is a registered U.S. trademark of Hewlett-Packard Company.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

OSF, OSF/1, OSF/Motif, Motif, and Open Software Foundation are trademarks of the Open Software Foundation in the U.S. and other countries.

Pentium® is a U.S. registered trademark of Intel Corporation.

SQL*Plus® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of the Open Group.

Windows NT® is a U.S. registered trademark of Microsoft Corporation.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

1. An Introduction to Integrating Partner Applications with OVO

2. Integrating Solutions with OVO

3. Using the Integration Capabilities of the OVO Motif-based GUI

5. Integrating with Java GUI

6. Integrating with Service Navigator

7. Integration Facilities of the HP OpenView NNM Core Platform

8. Creating and Distributing an Integration Package

A. Syntax Used in OVO Configuration Files

B. Symbols for Application Integration

C. About OVO Man Pages

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. the manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

First Edition: October 1996

Second Edition: August 1997

Third Edition: February 1999

Fourth Edition: June 2000

Fifth Edition: January 2002

Sixth Edition: April 2002

Sixth Edition: May 2004

Seventh Edition: September 2004

Conventions

The following typographical conventions are used in this manual.

Table 1 **Typographical Conventions**

Font	Meaning	Example
Italic	Book or manual titles, and man page names	Refer to the *OVO Administrator's Reference* and the *opc(1M)* manpage for more information.
	Emphasis	You *must* follow these steps.
	Variable that you must supply when entering a command	At the prompt, enter **`rlogin`** ***`username`***.
	Parameters to a function	The *`oper_name`* parameter returns an integer response.
Bold	New terms	The **HTTPS agent** observes...
`Computer`	Text and other items on the computer screen	The following system message displays: `Are you sure you want to remove current group?`
	Command names	Use the `grep` command ...
	Function names	Use the `opc_connect()` function to connect ...
	File and directory names	`/opt/OV/bin/OpC/`
	Process names	Check to see if `opcmona` is running.
	Window/dialog box names	In the `Add Logfile` window ...
	Menu name followed by a colon (:) means that you select the menu, then the item. When the item is followed by an arrow (->), a cascading menu follows.	Select `Actions: Filtering -> All Active Messages` from the menu bar.

Table 1 **Typographical Conventions (Continued)**

Font	Meaning	Example
`Computer Bold`	Text that you enter	At the prompt, enter **`ls -l`**
Keycap	Keyboard keys	Press **Return**.
`[Button]`	Buttons in the user interface	Click `[OK]`.

OVO Documentation Map

HP OpenView Operations (OVO) provides a set of manuals and online help that help you use the product and understand the concepts underlying the product. This section describes what information is available and where you can find it.

Electronic Versions of the Manuals

All manuals are available as Adobe Portable Document Format (PDF) files in the documentation directory on the OVO product CD-ROM.

With the exception of the *OVO Software Release Notes*, all manuals are also available in the following OVO web server directory:

`http://<management_server>:3443/ITO_DOC/<lang>/manuals/*.pdf`

In this URL, <*management_server*> is the fully qualified hostname of your management server, and <*lang*> stands for your system language, for example `C` for English and `japanese` for Japanese environments.

Alternatively, you can download the manuals from the following website:

`http://ovweb.external.hp.com/lpe/doc_serv`

Watch this website regularly for the latest edition of the OVO Software Release Notes, which gets updated every 2-3 months with the latest news such as additionally supported OS versions, latest patches and so on.

OVO Manuals

This section provides an overview of the OVO manuals and their contents.

Table 2 OVO Manuals

Manual	Description	Media
OVO Installation Guide for the Management Server	Designed for administrators who install OVO software on the management server and perform initial configuration. This manual describes: • Software and hardware requirements • Software installation and de-installation instructions • Configuration defaults	Hardcopy PDF
OVO Concepts Guide	Provides you with an understanding of OVO on two levels. As an operator, you learn about the basic structure of OVO. As an administrator, you gain insight into the setup and configuration of OVO in your own environment.	Hardcopy PDF
OVO Administrator's Reference	Designed for administrator's who install OVO on the managed nodes and are responsible for OVO administration and troubleshooting. Contains conceptual and general information about the OVO DCE/NCS-based managed nodes.	PDF only
OVO DCE Agent Concepts and Configuration Guide	Provides platform-specific information about each DCE/NCS-based managed node platform.	PDF only
OVO HTTPS Agent Concepts and Configuration Guide	Provides platform-specific information about each HTTPS-based managed node platform.	PDF only
OVO Reporting and Database Schema	Provides a detailed description of the OVO database tables, as well as examples for generating reports from the OVO database.	PDF only
OVO Entity Relationship Diagrams	Provides you with an overview of the relationships between the tables and the OVO database.	PDF only

Table 2 OVO Manuals (Continued)

Manual	Description	Media
OVO Java GUI Operator's Guide	Provides you with a detailed description of the OVO Java-based operator GUI and Service Navigator. This manual contains detailed information about general OVO and Service Navigator concepts and tasks for OVO operators, as well as reference and troubleshooting information.	PDF only
Service Navigator Concepts and Configuration Guide	Provides information for administrators who are responsible for installing, configuring, maintaining, and troubleshooting the HP OpenView Service Navigator. This manual also contains a high-level overview of the concepts behind service management.	Hardcopy PDF
OVO Software Release Notes	Describes new features and helps you: • Compare features of the current software with features of previous versions. • Determine system and software compatibility. • Solve known problems.	PDF only
OVO Supplementary Guide to MPE/iX Templates	Describes the message source templates that are available for MPE/iX managed nodes. This guide is not available for OVO on Solaris.	PDF only
Managing Your Network with HP OpenView Network Node Manager	Designed for administrators and operators. This manual describes the basic functionality of HP OpenView Network Node Manager, which is an embedded part of OVO.	Hardcopy PDF
OVO Database Tuning	This ASCII file is located on OVO management server on the following location: `/opt/OV/ReleaseNotes/opc_db.tuning`	ASCII

Additional OVO-related Products

This section provides an overview of the OVO-related manuals and their contents.

Table 3 Additional OVO-related Manuals

Manual	Description	Media
HP OpenView Operations for UNIX Developer’s Toolkit If you purchase the HP OpenView Operations for UNIX Developer’s Toolkit, you receive the full OVO documentation set, as well as the following manuals:		
OVO Application Integration Guide	Suggests several ways external applications can be integrated into OVO.	Hardcopy PDF
OVO Developer’s Reference	Provides an overview of all available application programming interfaces (APIs).	Hardcopy PDF
HP OpenView Event Correlation Designer for NNM and OVO If you purchase HP OpenView Event Correlation Designer for NNM and OVO, you receive the following additional documentation. Note that HP OpenView Event Correlation Composer is an integral part of NNM and OVO. OV Composer usage in the OVO context is described in the OS-SPI documentation.		
HP OpenView ECS Configuring Circuits for NNM and OVO	Explains how to use the ECS Designer product in the NNM and OVO environments.	Hardcopy PDF

OVO Online Information

The following information is available online.

Table 4 OVO Online Information

Online Information	Description
HP OpenView Operations Administrator's Guide to Online Information	Context-sensitive help system contains detailed help for each window of the OVO administrator Motif GUI, as well as step-by-step instructions for performing administrative tasks.
HP OpenView Operations Operator's Guide to Online Information	Context-sensitive help system contains detailed help for each window of the OVO operator Motif GUI, as well as step-by-step instructions for operator tasks.
HP OpenView Operations Java GUI Online Information	HTML-based help system for the OVO Java-based operator GUI and Service Navigator. This help system contains detailed information about general OVO and Service Navigator concepts and tasks for OVO operators, as well as reference and troubleshooting information.
HP OpenView Operations Man Pages	Manual pages available online for OVO. These manual pages are also available in HTML format. To access these pages, go to the following location (URL) with your web browser: `http://`*`<management_server>`*`:3443/ITO_MAN` In this URL, the variable <*management_server*> is the fully qualified hostname of your management server. Note that the man pages for the OVO HTTPS-agent are installed on each managed node.

About OVO Online Help

This preface describes online documentation for the HP OpenView Operations (OVO) Motif and Java operator graphical user interfaces (GUIs).

Online Help for the Motif GUI

Online information for HP OpenView Operations (OVO) Motif graphical user interface (GUI) consists of two separate volumes, one for operators and one for administrators. In the operator's volume, you will find the HP OpenView OVO Quick Start describing the main operator windows.

Types of Online Help

The operator and administrator volumes include the following types of online help:

❏ **Task Information**

Information you need to perform tasks, whether you are an operator or an administrator.

❏ **Icon Information**

Popup menus and reference information about OVO icons. You access this information with a right-click of your mouse button.

❏ **Error Information**

Information about errors displayed in the `OVO Error Information` window. You can access context-sensitive help when an error occurs. Or you can use the number provided in an error message to perform a keyword search within the help system.

❏ **Search Utility**

Index search utility that takes you directly to topics by name.

❏ **Glossary**

Glossary of OVO terminology.

❏ **Help Instructions**

Instructions about the online help system itself for new users.

❏ **Printing Facility**

Printing facility, which enables you to print any or all topics in the help system. (An HP LaserJet printer or a compatible printer device is required to print graphics.)

To Access Online Help

You can access the help system in any of the following ways:

❏ **F1 Key**

Press **F1** while the cursor is in any active text field or on any active button.

❏ **Help Button**

Click [Help] in the bottom of any window.

❏ **Help Menu**

Open the drop-down Help menu from the menu bar.

❏ **Right Mouse Click**

Click a symbol, then right-click the mouse button to access the Help menu.

You can then select task lists, which are arranged by activity, or window and field lists. You can access any topic in the help volume from every help screen. Hyperlinks provide related information on other help topics.

You can also access context-sensitive help in the Message Browser and Message Source Templates window. After selecting Help: On Context from the menu, the cursor changes into a question mark, which you can then position over the area about which you want help. When you click the mouse button, the corresponding help page is displayed in its help window.

Online Help for the Java GUI and Service Navigator

The online help for the HP OpenView Operations (OVO) Java graphical user interface (GUI), including Service Navigator, helps operators to become familiar with and use the OVO product.

Types of Online Help

The online help for the OVO Java GUI includes the following information:

❑ **Tasks**

  Step-by-step instructions.

❑ **Concepts**

  Introduction to the key concepts and features.

❑ **References**

  Detailed information about the product.

❑ **Troubleshooting**

  Solutions to common problems you may encounter while using the product.

❑ **Index**

  Alphabetized list of topics to help you find the information you need quickly and easily.

To View a Topic

To view any topic, open a folder in the left frame of the online documentation window, then click the topic title. Hyperlinks provide access to related help topics.

To Access Online Help

To access the help system, select `Help: Contents` from the menu bar of the Java GUI. A web browser opens and displays the help contents.

NOTE To access online help for the Java GUI, you must first configure OVO to use your preferred browser.

1 An Introduction to Integrating Partner Applications with OVO

Why Integrate with OVO?

A successful system management solution must satisfy a customer’s requirements for unified management across all platforms and all applications in a distributed environment. These requirements can seldom be satisfied by a single vendor, making partnerships essential to extend the functions and scope of a system management solution. With the **HP OpenView HP OpenView Operations Developer’s Toolkit** you have a powerful tool at your disposal to integrate your network solutions into HP OpenView HP OpenView Operations. By employing the standard integration capabilities of OVO, and the extended capabilities of the OVO Developer’s Toolkit, you can create a solution that addresses a wider range of requirements, and that the customer perceives as a single, unified product.

The standard HP OpenView HP OpenView Operations (OVO) product provides operations and problem management for multivendor distributed systems, and combines:

- ❑ Management of databases, applications, and networks;
- ❑ Detection of events occurring on managed nodes or SNMP devices;
- ❑ Filtering mechanisms to separate relevant events from irrelevant events;
- ❑ Generation of meaningful messages that include automatic and operator-initiated actions, and instructions for operators;
- ❑ Sophisticated Motif-based GUIs for operators and administrators, as well as a Java-based operator GUI.

For more information about the standard integration capabilities of OVO, see Chapter 3, “Using the Integration Capabilities of the OVO Motif-based GUI,” on page 87.

In addition to the standard functionality of OVO, the Developer's Toolkit provides a powerful C-library of **Application Programming Interfaces (APIs)**, including:

- ❑ Operator APIs to operate on OVO messages, message events, and applications responses, for example to own or disown a message.

  Interface API to access OVO by opening an instance to the following interfaces:

  - Server Message Stream Interface
  - Agent Message Stream Interface
  - Legacy Link Interface
  - Application Response Interface
  - Message Event Interface

- ❑ Configuration APIs to configure OVO data directly in the database. The functions allow you, for example, to configure new OVO templates or managed nodes, or to modify existing applications or users. In addition, functions are available to control access to OVO data, and to distribute configuration changes to the managed nodes.

For more information about the OVO User APIs, see Chapter 4, “Using the OVO Application Programming Interfaces,” on page 165.

The OpenView Windows API and SNMP API of Network Node Manager are also available for use in generating an integrated solution. For information on these APIs, see Chapter 7, “Integration Facilities of the HP OpenView NNM Core Platform,” on page 247.

These features make OVO ideally suited as an integration framework for other applications or solutions which address the system and network management market. Integration with OVO is especially attractive to partners who provide solutions in the following areas:

- Other system management functional areas, such as backup, spooling, job scheduling, security, or accounting.
- Problem management for specific applications, for example, database systems.
- Problem management for platforms on which OVO intelligent agents are not available.
- Enhanced problem handling, such as event correlation, helpdesk systems, and trouble-ticket systems.
- Service management to monitor business-relevant services.

HP OpenView Partnerships

The major benefit resulting from an integration with OVO is the increased customer value of the integrated solution. OVO is the industrial standard for problem management and supports a wide range of platforms which have either been developed internally, or by partners. When you integrate a solution with OVO, it becomes part of a comprehensive management solution which meets customers' requirements for a unified system management approach. This increases the value your solution provides to customers, making it attractive to market segments that it couldn't previously address. A **partner program** has been established by Hewlett-Packard to support your integration efforts.

Integrations created by solution partners can be validated and certified by Hewlett-Packard to achieve the status of **HP OpenView Premier Partner**. Validation ensures that the integration is well-behaved and does not conflict with other integrated solutions. As an HP OpenView Premier Partner, your solution is recommended by HP sales channels, you can leverage from the well-established HP OpenView brand name, and you receive immediate market exposure for your solution through HP market awareness and selling tools.

For more information about the HP OpenView partner programs, see our web site at `http://www.openview.hp.com`, and select `partners`.

HP OpenView Developer Assist

HP OpenView Developer Assist support that increases the speed, ease, and cost effectiveness of integrating with OVO. For additional documentation and ordering information, see our web site at `http://www.openview.hp.com`, select `partners`, `developers' and third-party applications`, and `developer support services`.

A Word about Licenses

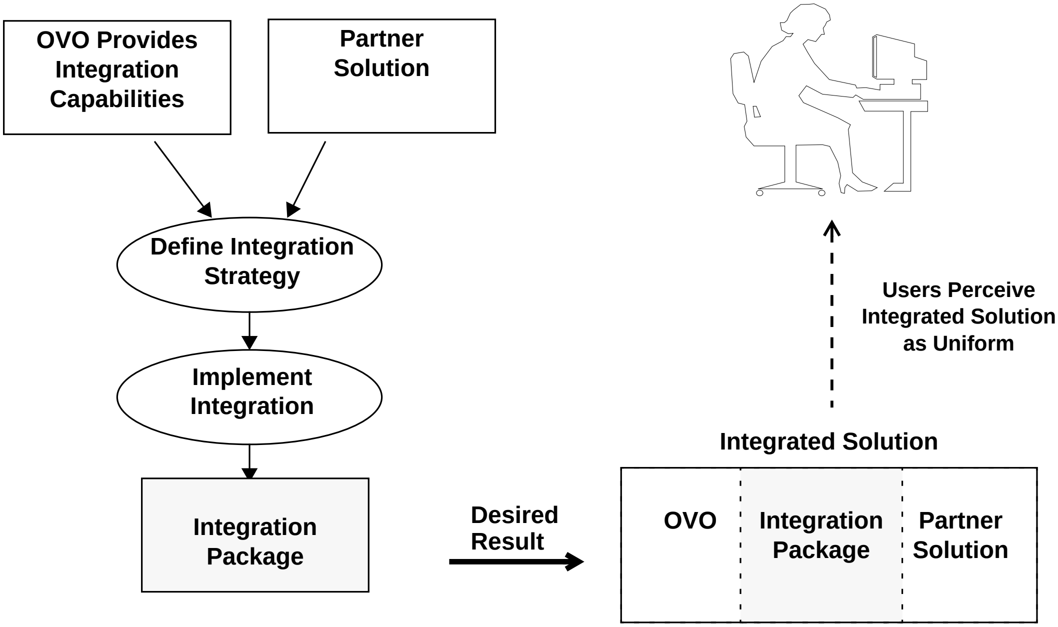
The Development Kit license contains a limited OVO management server license with five nodes and one user. NNM can manage a maximum of 25 objects with this license.

Integrating Partner Solutions with OVO

The ultimate goal of any integration must be to create an **integration package** that enables OVO and the partner solution to work so closely together that they are perceived by the customer as one powerful, integrated solution.

Figure 1-1 shows an overview of the integration process. It starts by analyzing the OVO functionality and integration capabilities available, and the characteristics of the partner solution. You can then design and implement an integration strategy based on this analysis. As a result of this activity an additional product part is created, referred to as the **integration package**. An integration package may consist solely of configuration files, or it may include new code for additional processes.

Figure 1-1 Integration with HP OpenView HP OpenView Operations



We use the term **tight integration** if the capabilities offered by OVO are fully exploited to maximize the uniformity of the integrated solution. For example, consider integrating a business solution with OVO. To achieve tight integration status, OVO capabilities should be employed to ensure that the application is constantly monitored so that OVO operators are immediately notified of problems related to the application. Whenever possible, corrective automatic and operator-initiated actions should be provided, and instruction text should help operators to solve any problems. All mechanisms underlying OVO functionality are highly configurable to enable a high degree of customization, and to provide unique opportunities for the tight integration of partner solutions.

OVO also provides many capabilities to help the integration process and to allow different integration strategies to be tailored to the type of solution, including:

❑ Event integration using messages.

  - Message generation is based on numerous message sources, including logfiles, SNMP traps, threshold monitor values, MPE/iX console messages, calls to opcmsg(1|3), Event Correlation Services (ECS), etc.
  - Instructions for operators, and automatic- and operator-initiated actions can be associated with messages.
  - Messages can be flagged to be forwarded to trouble-ticket and notification services.

❑ Powerful and versatile threshold monitoring and graphing capabilities.

❑ Predefined interfaces to trouble-ticket and notification systems.

❑ Integration of tools for operators and administrators in the OVO application desktop, menu bar, submenus, or toolbar.

❑ APIs and command line interfaces to the agents and to the management server.

❑ Access to NNM maps, submaps, symbols and the SNMP event system from either ASCII files or programs.

To make integration as straightforward as possible, most definitions can be done by way of the OVO GUIs. Tools to download configuration data required for the integration, and then upload it at the customer's site are also available.

OVO Conceptual Overview

This section introduces the key concepts behind the operation of OVO, to help clarify more detailed discussions of the OVO integration capabilities in later chapters.

The HP OpenView Product Family and OVO

HP OpenView is a family of integrated network and system management solutions for managing the complete information technology enterprise, including networks, distributed systems, applications, databases, and services. OVO is one of the key components of the HP OpenView Solution Framework that has become a leading Integrated Network and System Management (INSM) solution.

OVO Concept and Key Features

The most important tasks of operations management are to monitor the use of all systems and contributing resources, and to keep them under surveillance and operational control. Operations management is the central integration point for any INSM solution and includes the detection and reporting of problems, and the actions required to recover from these problems.

IT staff can use OVO to control the following elements and resources:

- Servers and clients
- Networks
- Operating systems
- Middleware
- Applications
- Databases
- Business services (with HP OpenView Service Navigator)

To achieve efficient operation and problem management, information must be gathered from the controlled elements and resources. OVO Agents are installed on managed nodes throughout the management domain which gather status information, messages, and monitoring values from a range of sources. SNMP agents, hosted on any system or IP device, are also fully supported by OVO. Message filters and thresholds are used to ensure that only the relevant information is sent to the management server and presented to the responsible OVO operators.

After collecting data, OVO combines all events received from the managed environment into a single procedural flow, or **message stream**. In addition to conditional event filtering and the suppression of unwanted messages, the **HP OpenView Event Correlation Services (ECS)** can tap the message stream on both the management server and the intelligent agents to reduce the volume of messages reaching the OVO Message Browser. This guarantees the maximum possible efficiency in local and central event analysis and handling. OVO provides multiple mechanisms, such as automatic actions, predefined operator-initiated actions, or problem-specific help text and instructions, to help the operator resolve critical conditions. The agent can even initiate and execute corrective actions without any involvement from the management server.

Besides configuration data, all status information gathered, including records that document completed actions, are stored in a central SQL database. The database offers an excellent starting point for future audits and analyses, and provides a central configuration of remote OVO domains.

You can easily adapt OVO to fit into existing IT infrastructures and adjust the managed environment at any time. The fully-customizable environment ensures a match of different skills, tasks, and responsibilities, and provides opportunities for task delegation. This enables multiple OVO operators to work together simultaneously in the same computing environment, without a duplication of effort.

In the area of network management, OVO provides powerful features for the discovery and mapping of networks which enable OVO operators to view the complete managed environment in the following ways:

- ❑ a logical (system) view
- ❑ a topological (network) view
- ❑ a service view (with the HP OpenView Service Navigator)

The implementation of OVO can be scaled from the management of smaller workgroups, running business critical applications, to the management of world-wide distributed computing environments with thousands of systems. OVO also supports competitive management approaches, for example, **follow-the sun**.

Open APIs and external interfaces enable the seamless integration of products such as database management modules, and customized application integration packages. Other IT infrastructure components, for example, trouble-ticket systems, pagers, and help desks are available on both the management server and agents.

To summarize, OVO is a core INSM component that can improve the effectiveness and productivity of any IT organization by increasing the availability of computing resources and reducing the time required to resolve problems.

OVO Implementation

OVO is a distributed client-server software solution and its architecture adheres to the manager/agent concept. Within a computing environment managed by OVO one, or several, suitable systems are selected as central **management servers**. The management servers receive and present management information, initiate actions and activate the agents, among other tasks. Other computer systems in the environment, connected by either LAN or WAN to the management servers, can be made **managed nodes**, running the **OVO agent software**. The OVO agent on the managed node collects and processes management information, forwards pertinent information to the appropriate management server(s), and starts local actions.

OVO can also monitor intelligent network devices such as bridges, hubs and printers which can submit **SNMP traps** if a fault or other event occurs.

Communication between the OVO management server and the managed nodes is based on DCE remote procedure calls (RPC) which enables bidirectional communication. In comparison with pure SNMP-based communication, the use of RPC allows true **management-by-exception**. This means that instead of the management server polling its managed nodes at regular intervals to obtain status information, it is contacted by the OVO intelligent agent only when a problem is detected. This minimizes the network traffic and increases the performance of the management server.

Problem Management with OVO

Regardless of the message source, OVO gathers the elements of daily operations and problem management by employing the following procedures:

❑ **Collecting**

Gathering information about the status of the computing environment.

❑ **Processing**

Selecting important or critical status information and making it available on the central system in a consolidated fashion.

❑ **Presenting**

Overview; highlighting of problems; definition of a problem resolution strategy.

❑ **Acting**

Performing planned activities and corrective actions, storing information and action logs (audit).

These various approaches are described on the following pages.

Collecting Management Information

OVO provides extensive collection services for management information. The agents gather management information originating from a variety of sources and when an exception is detected, they generate messages from the collected information.

All messages and events are intercepted by the agent at each managed node, filtered and classified, and then forwarded to the responsible management server. In an environment with multiple servers, the responsible server is determined by the manager-of-manager (MoM) configuration template. SNMP traps can also be intercepted on managed nodes if a trap template has been assigned to the nodes.

Logfiles and SNMP Traps

Important message sources include application and system logfiles, and SNMP traps. The OVO logfile encapsulator extracts important events from logfiles, and the event interceptor intercepts SNMP traps broadcast by components of the network. Multiple character sets for logfiles are supported, and conversion routines (e.g., for binary logfiles) can be applied to consolidate the message format, improve the problem text, set event attributes, etc. System messages normally displayed on an MPE/iX system console are an additional source of OVO messages.

Agent Message API

Management information, generated by applications or customer programs or scripts, can even be sent directly to the OVO agent on any managed node by way of the Agent Message API, see Chapter 4, “Using the OVO Application Programming Interfaces,” on page 165.

Threshold Monitors

The threshold monitoring capability of OVO is also a source of messages.

It enables you to manage nodes more proactively by tracking the development of potential problems. When the predefined threshold for a monitored object is exceeded, a message is generated.

OVO can collect monitoring information for basic system variables by accessing the SNMP Management Information Base (MIB). This service can be extended to any SNMP variable and to user-defined objects provided by your own monitoring applications.

Monitor values from external applications or scripts can be sent directly to the OVO agent on any managed node by the Agent Monitor API, to be locally checked against predefined thresholds, see Chapter 4, “Using the OVO Application Programming Interfaces,” on page 165.

Performance metrics are collected by the embedded performance component that is part of the OVO agents. The performance component collects performance counter and instance data from the operating system.

Legacy Link Interface API

To integrate hardware platforms that are not, or not yet, supported by OVO, the **Legacy Link Interface API** is provided to receive and pass on management information, see “Overview of the Legacy Link Interface” on page 180.

Processing and Consolidating Information

OVO offers extensive tools for the management of messages. Messages collected at the managed nodes are automatically forwarded to an appropriate management server.

To minimize network traffic, and to avoid overloading the user with irrelevant messages, filter conditions can be specified. All messages, including suppressed ones, can be logged on the originating node for future analysis. Each message can be assigned to a particular severity level (critical, major, minor, warning, normal) to show the relative importance of the event. If no severity class has been assigned, then the message is treated as belonging to class **unknown**. However, it is recommended to avoid the “unknown” message status because it is useful to know the severity of different events. Messages sent to OVO by way of the `opcmsg(1)` command are assigned a severity level of **normal** if no other severity level has been specified.

Messages which are considered to belong together, for example, if they are related to the same kind of managed objects or to a certain problem domain, can be grouped together into **message groups**. For example, all messages from a backup or spooler application might be grouped together. OVO provides several default message groups; see the *OVO Administrator's Reference* for a complete list. Message groups can be added, modified, or deleted in the `OVO Message Group Bank`. Note that the message groups `Misc` and `OpC` have special functions and must not be used for integration; they cannot be deleted.

You can configure message source templates at the management server and then download them to the managed nodes using the OVO GUI or the command `opctmpldwn(1)`. This process is independent of the location of the managed node. The monitoring of services at the managed nodes helps to reduce the network traffic. OVO also monitors its own processes to guarantee complete and continuous availability of its services.

Presenting the Information to the User

The typical working environment for an OVO user consists of the following main windows in the Motif-based GUI:

❑ IP Map

❑ Managed Nodes

❑ Message Groups

❑ Message Browser

❑ Application Desktop

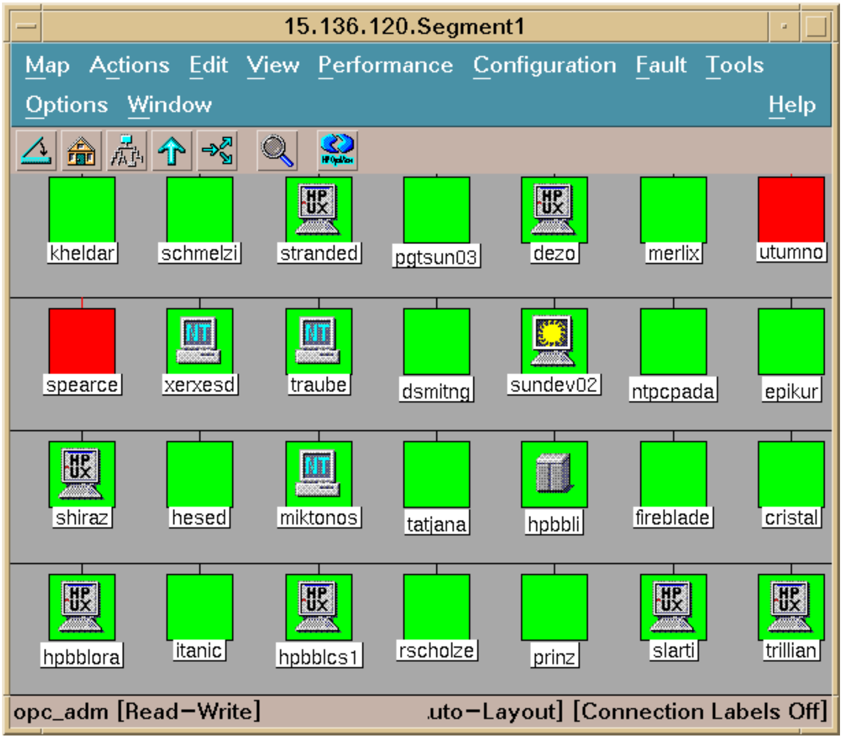
The content of these windows depends on the tasks and responsibilities assigned to a particular user; users only see the objects and messages for which they are responsible and can access only those applications needed to perform their allocated tasks.

The entire working environment of OVO can be configured to match the skills and responsibilities of the individual operator in terms of management information supplied and capabilities granted. The result is a task-oriented working environment. The internal notification service of OVO brings critical events to the user's notice by changing the color of the affected icons. In addition, external notification services such as pager, email, warning light, or telephone call initiation can be activated.

The powerful features of OVO are complemented by integrated partner solutions. Examples of tightly integrated solutions are HP OmniBack, and HP OmniStorage.

IP Map Window OVO includes the full network management features of HP OpenView Network Node Manager. These include network discovery and mapping, and the presentation of critical network events occurring in the managed computing environment.

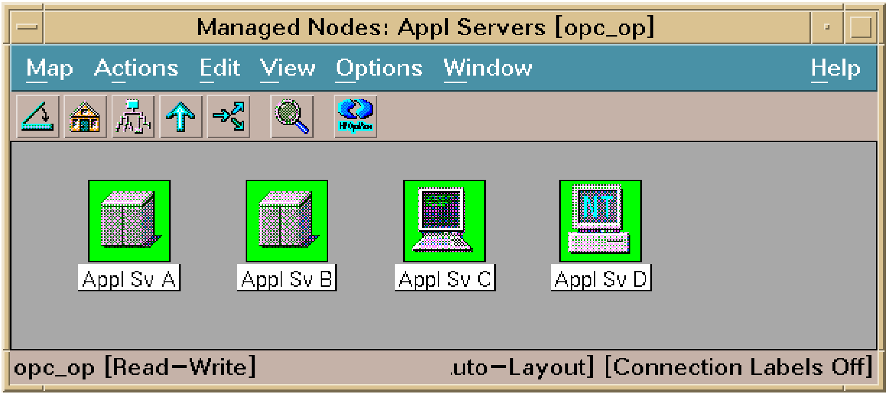
Figure 1-2 **IP Map Window**



The IP Map window provides the operator with a topological view of the network. This can be either a view of the entire network or of a specific submap. Information about the status of the network is essential for the efficient management of complex and distributed computing environments.

Managed Nodes Window The Managed Nodes window is the logical or system view of the managed environment for which the operator is responsible. Each node (or group of nodes) is represented by an icon. OVO changes the color of these icons to reflect the current status of the node according to the messages received into the Message Browser.

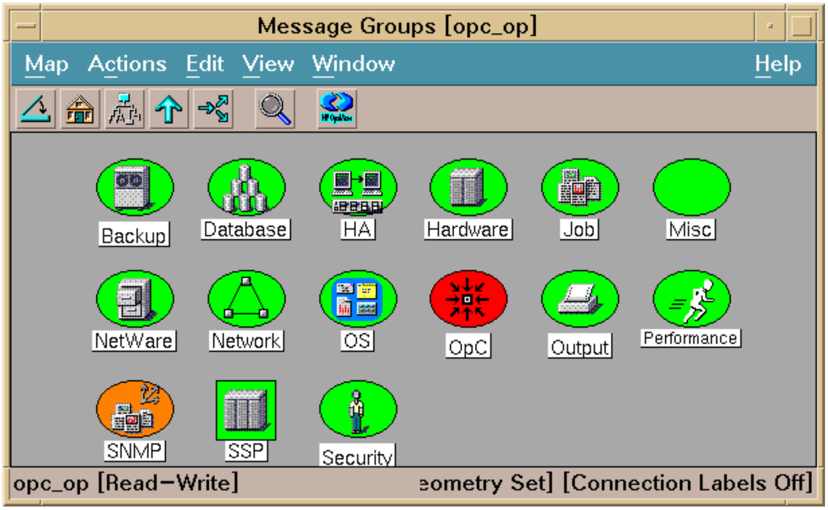
Figure 1-3 Managed Nodes Window



You can arrange the nodes in the Managed Nodes window in many different ways. For example, you might group nodes based on their geographical location (countries, cities, buildings, etc.) or based on logical components like routers, systems, mainframes, etc. Different node hierarchies can be defined to ensure that even large computing environments, with thousands of managed objects, can be controlled easily. By clicking on an icon in the Managed Nodes window, operators can locate the managed object that is the source of a message or event.

Message Groups Window The Message Groups window displays the message groups for which an operator is responsible. Messages are usually grouped by function, location, application, or other logical classification.

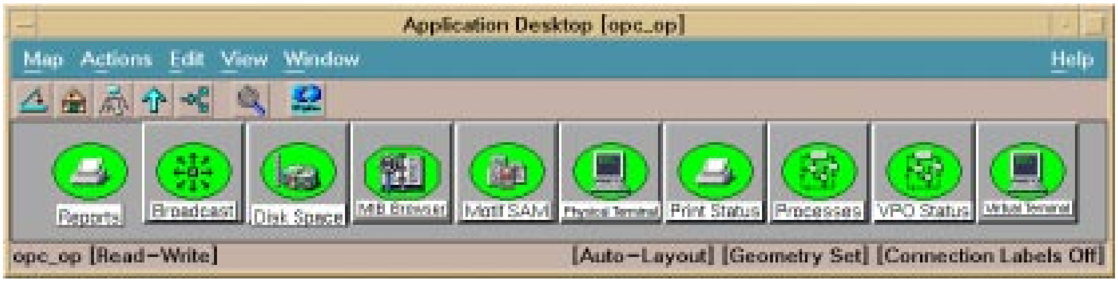
Figure 1-4 Message Groups Window



OVO changes the color of these icons to reflect the current status of the message group according to the active messages in the Message Browser.

Application Desktop Window The Application Desktop window displays icons for each management application that can be accessed by a particular operator. From this window, an operator can start daily problem management tasks. Some applications can be customized to provide either full or partial functionality.

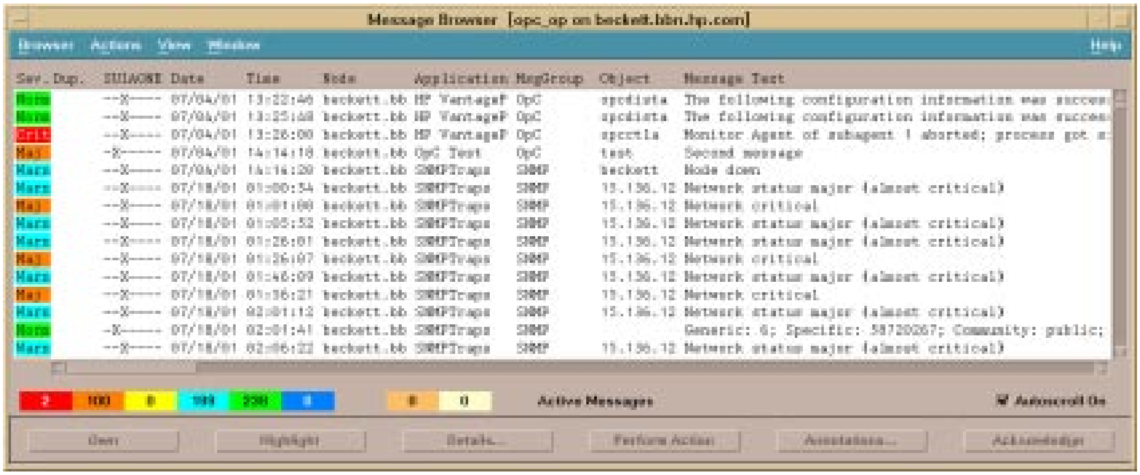
Figure 1-5 Application Desktop Window



As in other OVO main windows, you can group similar applications into their own sub-windows, for example, when applications address similar tasks, or provide multiple functional entry points.

Message Browser The `Message Browser` displays all messages received from the operator’s managed environment, including all nodes for which the operator is responsible, and which belong to message groups assigned to the operator. An operator can access detailed information about messages, find instructions to resolve problems, start/stop operator-initiated actions, review the status of automatic actions, modify message attributes, review or write message annotations, and highlight problem locations.

Figure 1-6 Message Browser



Message Browser [opc_op on beckett.bbn.hp.com]

```
Sev. Dup. SUIAONE Date     Time     Node        Application MsgGroup Object     Message Text
Norm      --X---- 07/04/01 13:22:46 beckett.bb HP VantageP OpC      opcdista   The following configuration information was succes
Norm      --X---- 07/04/01 13:25:48 beckett.bb HP VantageP OpC      opcdista   The following configuration information was succes
Crit      --X---- 07/04/01 13:26:00 beckett.bb HP VantageP OpC      opcctla    Monitor Agent of subagent 1 aborted; process got s
Maj       -X----- 07/04/01 14:14:18 beckett.bb OpC Test    OpC      test       Second message
Warn      --X---- 07/04/01 14:16:20 beckett.bb SNMPTraps   SNMP     beckett    Node down
Warn      --X---- 07/18/01 01:00:34 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
Maj       --X---- 07/18/01 01:01:00 beckett.bb SNMPTraps   SNMP     15.136.12 Network critical
Warn      --X---- 07/18/01 01:05:52 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
Warn      --X---- 07/18/01 01:26:01 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
Maj       --X---- 07/18/01 01:26:07 beckett.bb SNMPTraps   SNMP     15.136.12 Network critical
Warn      --X---- 07/18/01 01:46:09 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
Maj       --X---- 07/18/01 01:36:21 beckett.bb SNMPTraps   SNMP     15.136.12 Network critical
Warn      --X---- 07/18/01 02:01:12 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
Norm      -X----- 07/18/01 02:01:41 beckett.bb SNMPTraps   SNMP               Generic: 6; Specific: 58720267; Community: public;
Warn      --X---- 07/18/01 02:06:22 beckett.bb SNMPTraps   SNMP     15.136.12 Network status major (almost critical)
```

2 100 0 199 208 0   0 0   Active Messages   ☑ Autoscroll On

The `Message Browser` is a powerful source of information for system management and problem management tasks. If an operator is not able to solve a problem, the event can be escalated to an expert working at a different OVO site according to configured escalation rules, for example, based on date/time and message attributes.

Acting on the Information Provided

OVO offers several mechanisms for responding to events. When an event occurs that requires a non-interactive, corrective action, you can configure OVO to run the configured action automatically. These actions can be activated either from the management server and/or directly by the agents. For other significant events, OVO can provide event-specific instructions to guide operators during problem resolution. You can set up operator-initiated actions that are offered to an operator when a particular problem is reported in the operator's `Message Browser`.

All other events and management activities are handled within the `Application Desktop` window. From this window custom scripts, programs and management applications can be started and console windows opened on managed nodes.

The console login is under OVO control and can be configured to meet specific operating policies. If the network or remote system is down, a direct connection over a separate line to the physical console port of the managed node can be established. For similar management tasks that have to be performed on multiple managed nodes, OVO also provides a broadcast facility.

OVO allows you to track the steps taken to address a specific event. A facility to add annotations, and an interface to external trouble-ticket systems are provided. Records documenting the resolution of a problem provide a base for changing and creating message instruction text, defining enhanced problem resolution instructions, and developing more automatic actions.

Customizing OVO

OVO provides a wide range of elaborate customization capabilities so that it can be easily adapted to manage diverse IT environments. It can be configured to collect messages and SNMP traps from any source, and to monitor any variable of interest. Once the management information is collected, all follow-on activities can be configured to suit your IT requirements.

OVO also fully meets the needs of many different users. IT organizations often define individual management responsibilities for each member of the operating staff. Using the administrator's GUI you can specify the managed nodes and message groups for which each user is responsible. Only the messages and alerts that are the responsibility of a particular user appear in the Message Browser of that user.

Messages can be buffered if they arrive outside of configured service hours, or can be suppressed during scheduled outages. In addition, messages can be sent to different management servers depending on time and/or message attributes. For example, you might choose to send all messages to an OVO server in London between 8 AM and 5 PM, and to an OVO server in New York at all other times. You might even choose to send all messages indicating a database problem to your database expert center in Paris. For more information about configuring an OVO environment with multiple management servers, refer to the *OVO Concepts Guide*.

In addition to redirecting messages, which is automatically done by the OVO agent, operators can also **escalate** messages to another user, or to a user on a different management server. Using the **message forwarding** feature, messages can also be transferred between management servers.

The OVO administrator also controls the management tasks assigned to each user. Only those icons representing applications and control programs that are the responsibility of a particular user are displayed in `Application Desktop` of that user.

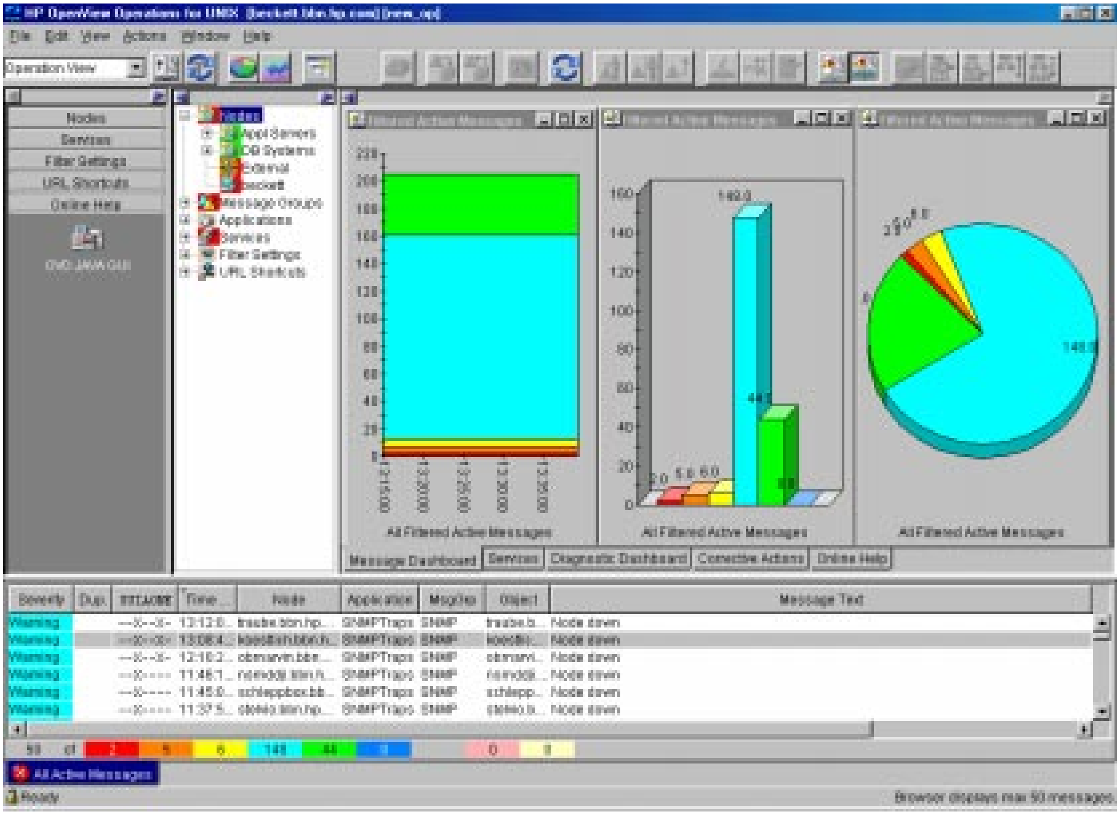
OVO also provides secure operations. Each user has a password to ensure that only authorized people can access OVO. User profiles defined by the OVO administrator restrict the activities of each user on the management server and the managed nodes. All actions can be controlled because activities are initiated only from the Application Desktop or Message Browser which are tailored to the responsibilities of the user.

The OVO Java-based Operator User Interface

In addition to the standard Motif-based operator GUI, OVO also provides a Java-based operator GUI. The Java-based GUI provides nearly the same functionality as the Motif-based GUI—see the OVO Software Release Notes for information about the differences between the two GUIs.

If you are using the Java-based user interface, your working environment looks similar to the one in Figure 1-7. See the online documentation supplied with the Java-based GUI for more information about how to perform tasks in this environment.

Figure 1-7 The Java-based Operator GUI



Severity	Dup.	SUIAONE	Time	Node	Application	MsgGrp	Object	Message Text
Warning		--X--X-	13:12:0...	trasbe.bbn.hp...	SNMPTraps	SNMP	trasbe.b...	Node down
Warning		--X--X-	13:08:4...	kaedlish.bbn.h...	SNMPTraps	SNMP	kaedlis...	Node down
Warning		--X--X-	12:10:2...	obmarvin.bbe...	SNMPTraps	SNMP	obmarvi...	Node down
Warning		--X----	11:46:1...	nomdqi.bbn.h...	SNMPTraps	SNMP	nomdqi...	Node down
Warning		--X----	11:45:0...	schleppbox.bb...	SNMPTraps	SNMP	schlepp...	Node down
Warning		--X----	11:37:5...	stenio.bbn.hp...	SNMPTraps	SNMP	stenio.b...	Node down

You can operate with Java-based GUI remotely from other Java applications using the Java GUI Remote APIs. For more information, refer to Chapter 5, “Integrating with Java GUI,” on page 219.

Integration Benefits to Partners

OVO is a leading framework for **Integrated Network and System Management (INSM)** so when integrating with OVO, a partner solution becomes a component of an INSM solution used to manage a complete IT environment. Integration into OVO increases the customer's perceived value of a partner solution and makes it attractive to market segments that it might be unable to address on its own.

OVO as an INSM Framework

OVO is the leading INSM framework for problem and operations management. It provides its users (administrator, template administrators, and operators) with a complete view of the IT environment, including:

❏ Low-level network devices such as bridges, routers, hubs and printers;

❏ Computer systems in a heterogeneous environment;

❏ Software such as operating systems, databases, and applications (including distributed systems).

As the availability of distributed systems depends on all components of the IT environment working smoothly together, a complete view of the managed IT environment is required before you can analyze the root cause of a problem. OVO can collect problem information from all levels and present it to operators in a consistent way. Operators do not need to switch between different interfaces. Considerable management information can accumulate from a large IT environment that must be filtered and distributed to multiple operators, in order to cope with the complexity of problem management.

OVO is not automatically aware of developing or already existing problems in the managed IT environment. It is the responsibility of the solution partners either to develop ready-to-use OVO configuration packages that provide the specialized knowledge for problem detection and resolution, or to extend partner solutions so that they work smoothly together in the OVO INSM framework.

Specific Benefits for Integrators in the NIM, NSM, and INSM Markets

The following discussion is based on the popular classification of the IT management market into the following segments:

❏ Network Infrastructure Management (NIM)

❏ Networked System Management (NSM)

❏ Integrated Network and System Management (INSM)

❏ Service Management

NIM Market Segment

Management solutions providing problem management for this segment deal with low-level network devices such as bridges, routers, hubs, printer and network connections of computers. They typically monitor the state of these devices and query or set device parameters by accessing SNMP MIB values. Typically, these solutions can be configured to receive and act on SNMP traps.

Management solutions addressing this segment are typically integrated into HP OpenView Network Node Manager (NNM). They use the NNM capabilities to access their functionality from a central console. For example, a solution might provide an operator with a view to the backplane of a router. An operator might then observe the traffic passing through the router from the central console. These solutions might integrate into OVO in a similar way as into NNM. This integration would still benefit from features that are standard in OVO. This type of solution can easily be migrated from NNM to an OVO integration so that they become part of an INSM solution.

NSM Market Segment

Management solutions for this market segment deal with computer systems, databases and applications connected over a network. This includes solutions for print and storage management, for configuration management solutions, or for database and application management solutions.

Solutions for the NSM segment integrate into OVO to prepare a management solution to become part of an INSM solution. These solutions can take full advantage of the OVO-specific integration capabilities from event integration via messages, to using the various OVO APIs.

To integrate applications or databases for which no satisfactory problem management solutions are available, OVO provides mechanisms that allow the straightforward development of a powerful management solution. Usually it is sufficient to use the integration capabilities that can be configured in the OVO administrator's GUI and to provide some additional shell scripts.

INSM Market Segment

The following solutions should also consider OVO integration:

- ❑ Solutions that address both network and system level problems
- ❑ Solutions that focus on other aspects of problem management, for example, trouble ticket and helpdesk systems or event correlation engines
- ❑ Solutions that provide problem management for platforms not directly supported by OVO.

These solutions may benefit considerably from an integration into OVO, first, because they make their solution ready for inclusion in a full INSM solution, and second, because they can access the large amount of problem-related messages that are collected and managed by OVO.

These solutions can take full advantage of the OVO-specific integration capabilities from event integration via messages to using the various OVO APIs. Especially for trouble ticket and help desk systems and for event correlation engines, OVO provides powerful interfaces (trouble ticket and external notification) as well as specific APIs including the Legacy Link Interface API, Agent Message Stream Interface API, Server Message Stream Interface API, Server Message API.

Service Management Market Segment

The following solutions should also consider OVO integration:

❑ Service providers

Integration Facilities Provided by OVO

OVO is designed to provide the maximum flexibility for integrators, so the mechanisms underlying OVO functionality are configurable to a high degree. This enables you to customize OVO installations to the specific needs of your customers. In addition, OVO provides powerful and straightforward tools that make the tight integration of partner solutions possible, such as APIs to the management server and managed nodes. See the OVO Software Release Notes for more information about the new features provided with this release of OVO.

Integrating Events Using Messages

OVO intercepts and collects messages generated by diverse components of the network so that it is informed of events occurring throughout the environment. Messages may be generated in the following circumstances:

- ❑ When a new entry is written to a system or application logfile on the managed nodes.
- ❑ When an SNMP trap is sent from an SNMP device.
- ❑ When the threshold monitoring capability of an OVO agent detects that a monitor threshold has been exceeded.
- ❑ When functions of the Agent Message API or Agent Monitor API are called.
- ❑ When an MPE/iX console message is written to the console of an MPE/iX agent.

Message generation, regardless of the message source, is controlled by **message templates** that have a similar structure for all types of message source.

Threshold Monitoring

The preceding sections described the integration of events of which OVO could read some type of text, for example, text written to a logfile or associated with an SNMP trap. In addition, OVO provides a **threshold monitoring** capability to deal with numeric object properties, for example, resource allocation values such as CPU load, disk space usage, or any other value relevant to the managed objects.

The importance of not generating too many messages has already been stated; this also applies to numeric values. A first step towards reducing the number of generated messages is to recalculate the monitored values only at regular time intervals. These time intervals are user-defined and can be different for each monitored value. In general, however, it is not desirable to have a message generated every time the monitored value is recalculated. To prevent this, OVO uses thresholds so that a message is only generated when the monitored value exceeds a threshold.

Note that monitor values are not only obtained by the intelligent agent recalculating the monitor value, they can also be passed to OVO agents directly. The functions of the Agent Monitor API allow other applications to pass monitor values to an OVO intelligent agent.

You can monitor any object property that can be expressed numerically, including:

❑ **Application and system values**

  Compare important application or system-specific values with their expected “normal” values.

❑ **Database values**

  Use the database SQL language and database administration tools to monitor specific values, for example, table sizes, number of locks, etc., and compare these values with a predefined set of normal values.

❑ **Processes**

  Use scripts to monitor whether important processes like daemons are running. Check the important process specific values, for example, the number of running processes.

❑ **Files and/or Filesystems**

Check the existence and/or the sizes of important files or file systems. The script might return the used or available disk space and OVO checks it against predefined upper or lower limits.

❑ **Performance metrics**

Use the embedded performance component to collect performance counter and instance data. The platform-generic metrics can be used to answer most questions about a system's global configuration, CPU, disk, swap, and memory usage. The typical metrics vary by platform but are available on most platforms and are generally useful for drill down and diagnosis on a particular system.

❑ **Management Information Base (MIB) variables**

An alternative mechanism for polling MIB variables is also provided by OVO, referred to as **MIB data collection**. In contrast to the threshold monitors, you can configure this mechanism to store the collected values in a more efficient format. The stored MIB data can be used to analyze trends in monitored variables by graphing the values over time. This type of monitor also supports threshold values, but does not have the sophisticated filtering capabilities of the OVO threshold monitors.

Working with Message Templates

You can use message templates to filter and suppress messages, reformat message text, and link actions and instructions with a message. You can define message templates using the `Message Source Templates` window that is accessible to both the OVO administrator and the OVO template administrator.

By configuring OVO templates, you can specify which message source is to be used, and how it is employed. You can choose which events detected through the message source are relevant to justify a message to be generated and sent to the management server, or which are irrelevant and can be suppressed. If a message is generated for an event, the template specifies the composition of the message, its attributes, and the instructions, annotations, and actions to be associated with the message.

For more information about configuring message templates, see the *OVO Concepts Guide* and the *OVO Administrator's Guide to Online Information*.

❑ **Basic Template Properties**

These include the template name and description, a specification of the message source to be used and how to use it. For example, the name of a logfile to read, the program used to preprocess the logfile, the time interval for checking the logfile, etc.

In addition, the basic template properties include the defaults that are used to set the attributes for generated messages if no additional information is specified, for example, default severity, default message group, etc.

❑ **Template Conditions**

A list of **conditions** belongs to each message template. You can choose either **suppress matched conditions** or **suppress unmatched conditions** to filter out irrelevant events, or choose **message conditions** to extract relevant events and forward them to the Message Browser. Both message and suppress conditions can be placed in any order, and they are processed by OVO in the order in which they are listed. The first matching condition determines how OVO reacts to an event, so the sequence of conditions in the condition list must be carefully considered.

A condition consists of a **match condition** part that determines to which events detected in the message source the condition applies. Among other message attributes, the message text can be used in the match condition. You can specify patterns using regular expressions which the message text must match.

If a condition is a **suppress matched condition**, it is used to suppress message generation for events that match the condition, whereas if it is a **suppress unmatched condition**, it is used to suppress message generation for events that do not match the condition. Consequently, for a suppress condition, all you need to specify is the match condition. For a message condition that triggers the generation of a message, you also need to define the following:

- Attributes of the generated message, for example, what message text to use, the severity level, the service name, etc.
- Custom message attributes of the generated message. These are attributes that you can set to provide your operators with more relevant information about a message.
- Instructions, annotations, automatic and operator-initiated actions to be associated with the message
- Whether the message is to be forwarded to a trouble ticket system or should trigger an external notification.

❑ **Template Options**

At the template level, you can specify which messages to log locally on the managed node. In addition, you also need to decide how to handle entries in the message source that match neither a message condition nor a suppress condition.

❑ **Advanced Options**

You can set the defaults for advanced options at both the template and message condition level. If different values for the advanced options are set at the message condition level, these will overwrite the template defaults. The advanced options specify whether duplicate or similar messages are to be suppressed and whether to copy or divert messages to processes which use the server or agent message stream interface API.

Adding Instructions, Annotations, and Actions to a Message Template

The previous section described how you can use conditions to filter messages so that operators are not overwhelmed by a flood of messages of varying importance. A further function of OVO lets you completely rephrase the text of a message so that it can be easily understood by an operator. This feature can help OVO operators to resolve problems that are already manifest or to proactively avoid problems if a situation that might cause a problem is detected.

For many problem situations additional information is available that might help operators to resolve a problem. For other problems, the action necessary to solve the problem is known in advance. In these cases, you can configure an **automatic action** to reduce the workload of the operator. You can also use an **operator-initiated action** if the solution to a problem requires the operator to do an action. You can even combine all these capabilities to create a powerful problem-solving tool.

For example, consider a scenario in which a file system problem occurs. You might use an automatic action to get more information about the status of the file system, then present this information to the operator as an annotation to the message announcing the problem. The operator might then initiate a predefined command, provided as an operator-initiated action, to resolve the problem.

OVO provides the following mechanisms to support problem resolution and self-attended operation:

❑ **Message Instructions**

These help an operator to solve the problem at hand. They describe the available automatic and operator-initiated actions, give advice about manual steps for solving the problem, and provide any additional explanation that might be useful to an operator.

You can define a default instruction text at the template level, and then define more specific instructions for each message condition of a template.

❑ **Message Annotations**

You can associate any text with a message as an annotation. In contrast to message instructions, annotations are intended to be extended as a message is processed. For example, you might provide the output of automatic and operator-initiated actions to the operators as an annotation.

An operator can later add comments about how the problem was solved to share this information with colleagues who might search the message history log if they encounter a similar problem.

❑ **Automatic Actions**

OVO can invoke actions automatically as a reaction to specific events. An action can be any shell command line, with parameters if necessary, that does not require human interaction.

Automatic actions can either be performed on the node on which the message was generated (**local automatic actions**) or on any other node. Local automatic actions do not require communication with the OVO management server and thus are performed even if the network is down or the OVO management server is not available.

You can even specify that successful completion of the automatic action automatically acknowledges the related message, i.e. the message is no longer displayed in the operator’s message browser.

- ❑ **Operator-initiated Actions**

  OVO provides a mechanism to offer predefined actions that must be started by an operator.

  This is useful if an action requires human interaction, or if an action must be tailored to the detected event, or if human judgement is required. As with automatic actions, on the successful completion of an action, the related message can be acknowledged automatically.

While you can define instruction text at a template and message condition level, annotations and actions are always associated with message conditions. At this level, you can also define two other functions that are closely related to actions. You can decide:

- ❑ Whether a message matching a condition should be forwarded to a trouble ticket system
- ❑ Whether a message matching a condition should call an external notification service.

You can define all these mechanisms using the OVO administrator's GUI.

Integrating Events using Trouble Ticket and Notification Services

OVO includes interfaces to trouble-ticket and notification systems which enable communication from the OVO management server to a trouble ticket system or a notification system, for example, beepers or e-mail systems. The forwarding of messages to a trouble ticket system or to a notification system is defined in the message conditions of templates. This enables you to define exactly which messages are to be forwarded.

By using a notification schedule, you can define which of the available notification systems to use, depending on the day of the week and time.

NOTE OVO does not provide trouble ticket or notification services, but it does support an interface to export event-specific details to an external trouble ticket service or/and to external notification services.

Integrating Applications into the Application Bank

Any tools that can help operators and administrators in their assigned tasks should be integrated into the **OVO Application Bank**. This is the utility used to define the capabilities of operators. The OVO administrator can set up different Application Desktops for each OVO operator. You can group similar applications together into **Application Groups** to avoid cluttering up the desktop.

An operator can invoke applications simultaneously on several nodes, or assign different parts of an application to different operators depending on their responsibility. The latter feature, however, depends on whether the application provides several entry points.

Most applications can be integrated into the Application Bank. Applications with an X-window user interface appear as usual in an X-window environment, others appear in terminal windows.

To integrate an application, you can also add your own entries into the OVO menu structure. The highest-level menus describe generic functionality; submenus describe more specific functionality. Menus can be enabled or disabled based on selection rules which specify the type and number of nodes that must be selected in a map window before a menu item becomes active. Inactive menus are automatically grayed-out. Toolbars provide a quick, intuitive means of invoking actions. OVO provides a default set of toolbars for invoking actions such as panning or selecting the root map. When an application is added to an OVO environment, it can add icons into existing toolbars, or create window-specific toolbars and icons. See “Integrating External Applications into the OVO GUI” on page 146 for more information.

Integrating via APIs

OVO provides a set of APIs which can be grouped according to their functions as follows:

❑ **OVO Operator APIs**

  This group of APIs enable certain actions to be immediately performed on OVO data. These APIs include:

  - OVO Data API
  - OVO Interface API

    This group of APIs enables external applications to register with OVO to receive information. When the requested information is available, OVO forwards it to the requesting interface. The following interfaces to OVO are available:

    — Server Message Stream Interface API

    — Agent Message Stream Interface API

    — Legacy Link Interface API

    — Application Response Interface API

    — Message Event Interface API

  - Server Message API
  - Agent Message API
  - Agent Monitor API

❑ **OVO Configuration APOVO**

This group of APIs enables applications to connect to the OVO database and configure OVO object directly in the database without using the GUI. These APIs include:

- Connection API
- Application Configuration API
- Application Group Configuration API
- Message Group Configuration API
- Message Regroup Condition Configuration API
- Node Configuration API
- Node Hierarchy Configuration API
- Template Configuration API
- User Profile Configuration API
- User Configuration API
- Distribution API
- Server Synchronization API

For information about the API functions, see the OVO Developer's Reference.

The OVO APIs are designed to ease the integration of partner solutions. Examples for the use of these APIs include:

❑ Full integration of legacy systems for which OVO agent software is not available.

❑ Connection of event correlation engines to process messages from the internal message stream of the management server or agent.

❑ Implementation of bidirectional communication between trouble ticket systems and OVO.

Command line functions to acknowledge messages are also available, for example `opcackmsg(1M)` on the OVO management server and `opcmack(1M)` on the managed nodes.

NNM Integration Through the OVO GUI

OVO features a default integration with NNM, providing the operator with the ability to start ovw applications from the OVO GUI. This functionality is available in both the local case, where the NNM system is installed on the OVO management server, and in the remote case, where NNM is installed on a server other than the OVO management server.

NOTE OVO provides different functionality, depending on whether the operator is working from the Motif UI or the Java UI.

Command line functions for the Java UI are also available, opcctrlovw (1m), a tool for controlling an associate ovw process, and opcmapnode (1m), which automatically determines which NNM nodes are available on the domain in order to start an ovw process.

Integrating via NNM Functionality

The HP OpenView Network Node Manager (NNM) consists of two main parts: HP OpenView Windows (ovw) and the HP OpenView SNMP Event System.

The **HP OpenView Windows API** supports the creation and manipulation of maps, submaps and symbols, and allows the generation of dialogs for user interaction. It is intended for the development of applications that need to use these features. New symbols can also be registered by ASCII Symbol Registration Files.

The HP OpenView Windows API also supports access to the HP OpenView Windows object database which stores information about network objects. An object is an internal representation of a logical or physical entity or resource that exists somewhere in a computer network. Objects have certain fields which can be seen as the attributes of an object, for example an object representing a computer in a network will have fields for the name and the IP address of that node. New fields can be added to objects by either using the HP OpenView Windows API or Field Registration Files.

The **HP OpenView SNMP Event System** provides a command and a programmer's interface to SNMP communication functions, an API for SNMP configuration purposes and it stores information about the network topology in a topology database.

2 Integrating Solutions with OVO

Deciding Which Integration Capabilities to Use

The different integration capabilities provided by OVO are summarized in the previous chapter. This chapter discusses the advantages and disadvantages of different integration capabilities to help you to plan and design an **integration package**.

It is important to decide whether the integration capabilities to be used are available on the management server or on the managed nodes. It is preferable, whenever possible, to choose integration capabilities on the managed nodes rather than on the management server. Should you choose to access the management server directly, you may encounter the following disadvantages:

- ❏ Since filtering is done on the management server, any event causes additional network load which in many cases turns out to be unnecessary, especially if the event is filtered out and no message is generated.
- ❏ Additional load on the management server might cause performance problems for the server; it is better to take advantage of the distributed CPU power available on the managed nodes.
- ❏ The threshold monitoring functions of the agent are not available.
- ❏ If the management server is unreachable at the time information is sent, for example, if there is a problem with the network, data may be lost.
- ❏ The ability of the agent to perform immediate local automatic actions is not available.

As a general rule, try to exploit the capabilities available on the managed nodes in preference to those available only on the management server.

There is no restriction on the number of templates for a particular message source that can be active at a given time. Templates for each source (logfiles, SNMP traps, OVO Message API, MPE/iX console, etc) are evaluated in parallel.

When integrating a partner solution it is recommended that you create a new template for that solution whenever possible. This prevents any possibility of the conditions defined for the new solution conflicting with previously defined conditions. In some circumstances, however, this may result in the generation of similar messages from different sources. To prevent the Message Browser from being flooded by similar messages, it is recommended to switch off the `Forward Unmatched Messages` option under normal working conditions.

A major goal of the certification program is to ensure that conflicts between message conditions are avoided. This can be achieved, for example, with SNMP traps by always including the enterprise-specific trap IDs in the match condition. For other message sources the unique application name should be included in any match condition.

These recommendations can be summarized as follows:

1. Try to exploit the functionality of the OVO agent whenever possible.
2. Prefer capabilities handled on the managed nodes to those handled on the management server only.
3. Define a new template for each integration and/or message source; do not append new conditions to an existing message source template.

Table 2-1 shows where the different message sources are handled, and lists any limitations:

Table 2-1 **Message Source Management**

Message Source	**Managed On**
Logfile Encapsulation	Managed Nodes
API or command line interface to managed nodes: passing a message by way of **opcmsg(1|3)** or **opcagtmsg_send(3)**	Managed Nodes
Threshold Monitoring This applies to all types of threshold monitors, both for monitors for which values are determined by the OVO agent, and for monitors for which values are passed to the agent by a call to **opcmon(1|3)** or **opcagtmon_send(3)**	Managed Nodes
SNMP Traps	Managed Nodes (for HP-UX, Solaris, AIX, Windows NT/2000, and Novell NetWare agent platforms only) or Management Server
MPE/iX Console Messages	Managed Nodes (MPE/iX only)
Calls to functions of management server APIs: Legacy Link Interface API, Server Message Stream Interface API, Server Message API, etc.	Management Server
Calls to functions of managed node: Agent Message Stream Interface API, Agent Message API	Managed Nodes

Defining an Integration Strategy

This section describes how to select a suitable integration strategy for your solution, based on your integration starting point:

❑ Starting with an existing integration into OVO (aimed at the NSM market segment).

❑ Starting with an existing integration into HP OpenView Network Node Manager (NNM) (aimed at the NIM market segment).

❑ Starting from scratch

  No previous HP OpenView integration available.

❑ Obtaining coexistence of NNM and OVO

  The final part of this section provides some hints to integrators who want to integrate into both NNM and OVO (aimed at the INSM market segment).

Adapting an Existing OVO Integration for OVO A.08.10

NOTE

If you need to integrate into OVO a partner solution that was integrated into a previous (and recent) version of OVO, any integration points introduced with OVO version A.08.10 will not be used. In addition, you may have to recompile the application with the libraries from the latest version of the OVO software, if those libraries have been changed.

The recommendations for adapting an existing OVO integration depend on the evaluation of the integration. Two scenarios can be distinguished:

❑ An existing integration was designed for the OVO version A.06.xx release and takes full advantage of the integration capabilities offered by this release for a tight integration to be achieved.

❑ An existing OVO integration does not take advantage of the integration capabilities offered by the OVO version A.06.xx release.

In the second case, in which the integration does not take full advantage of OVO A.06.xx integration features, a complete verification of the integration is recommended.

If you can leverage from a tight OVO A.06.xx integration, you might use this integration as a starting point. Before starting, establish whether you can enhance the integration by using features of the latest version of OVO. Refer to the *OVO Software Release Notes* for more information about the new features of OVO.

Leveraging From an Integration into NNM

If you currently use NNM, it is worth remembering that, as an OVO network operator, you can work in a way that is similar to the role of the typical NNM operator. OVO provides the preconfigured network operators **netop** and **itop** to whom the IPMap application, and the Network and SNMP message groups are assigned by default.

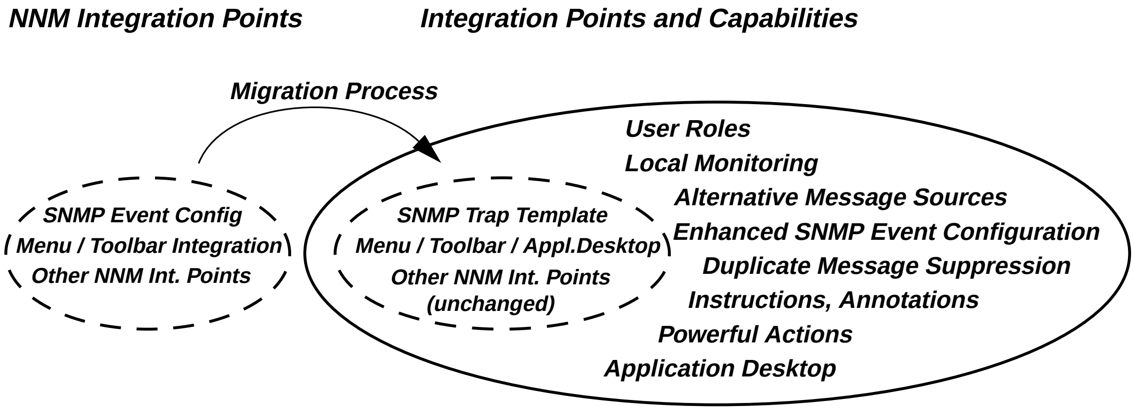
All NNM applications, and additional applications that can be used for configuration purposes are assigned to the **itop** operator, whereas only those NNM applications not used for configuration purposes are assigned to the network operator **netop**. An operator can invoke NNM applications using the menu items in the NNM menu bar. Alternatively, network operators can invoke the same applications from their personal application desktop.

The nodes from which the OVO network operator receives SNMP traps can be configured by means of the **external nodes** symbol which defines source nodes in addition to the nodes in the Managed Node window.

Applications integrated into NNM that are not part of the standard NNM product can be easily integrated into OVO.

OVO provides all the NNM integration points in addition to its own integration points. You can further enhance an integration by exploiting these additional OVO-specific integration points, as shown in Figure 2-1.

Figure 2-1 SNMP Event Configuration


NNM Integration Points
Integration Points and Capabilities
Migration Process
SNMP Event Config
Menu / Toolbar Integration
Other NNM Int. Points
SNMP Trap Template
Menu / Toolbar / Appl.Desktop
Other NNM Int. Points
(unchanged)
User Roles
Local Monitoring
Alternative Message Sources
Enhanced SNMP Event Configuration
Duplicate Message Suppression
Instructions, Annotations
Powerful Actions
Application Desktop

SNMP Event Configuration

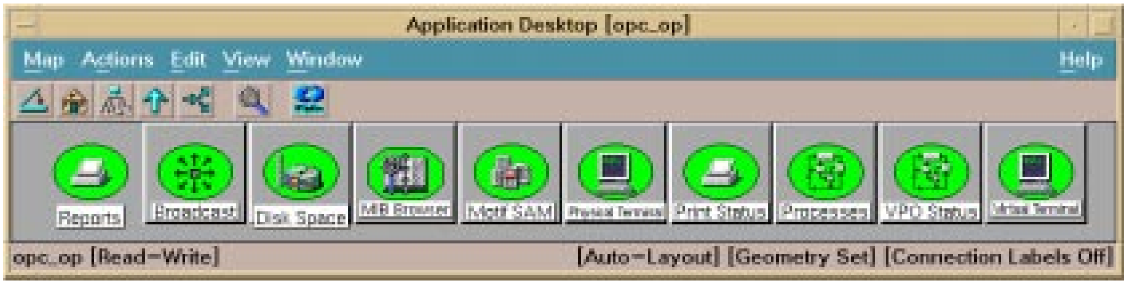
SNMP traps are configured using the SNMP Trap Templates. An SNMP Trap Template is automatically generated by a conversion command that takes the NNM SNMP Event Configuration as input. By using the SNMP Trap Template, the following additional features are available for customers migrating from NNM:

- ❑ OVO provides automatic and operator-initiated actions which can be started on demand by authorized operators. Actions can be performed on the management server or on any OVO managed node running the agent software. The status of an action is visible in the OVO message browser (running, failed or successful) and the output of the action can be attached to the message as an annotation.
- ❑ Instructions can be attached to an OVO message which can either be statically configured for each condition or template, or dynamically created using the Instruction Text Interface. For a detailed description, see “Adding Instructions, Annotations, Automatic- and Operator-initiated Actions” on page 136.
- ❑ Messages are stored in the central database, and both standard and user-customized reports can be generated.
- ❑ A Duplicate Message Suppression function supports the following:
  - Suppression configuration at a template and/or condition level
  - Suppression of identical messages or messages matching the respective condition
  - Duplicate message re-transmission based on time intervals and/or counter
- ❑ The Server or Agent Message Stream Interface (MSI) API enables messages to be diverted or copied to external applications like event correlation systems.

Powerful GUI Application Integration

In NNM, applications can be integrated into menus or the toolbar by way of Application Registration Files (ARFs). The same mechanism is also available in OVO. See "Integrating External Applications into the OVO GUI" on page 146 for more information.

Figure 2-2 Application Desktop Window



opc_op [Read-Write]

[Auto-Layout] [Geometry Set] [Connection Labels Off]

OVO applications can also be integrated using a GUI registration method, and are displayed as icons in the operator's Application Desktop. The integration method for OVO applications has the following advantages over the method for NNM applications using ARF files:

- ❑ Application groups provide structuring of applications;
- ❑ Applications can be executed remotely under any preconfigured user ID;
- ❑ Startup of applications can be preconfigured by the OVO administrator.

Consequently, the Application Desktop is the recommended integration facility to integrate applications into OVO. Applications newly added to OVO should always be integrated as OVO applications.

Monitoring Facilities

The **Data Collection and Threshold Monitoring** facility of NNM can be used to monitor numeric MIB values. Alternatively, you can configure monitor templates for OVO managed nodes to use the OVO threshold monitoring facility. This has the following advantages:

- ❑ With MIB data collection, polling is always done by the management server which sends a request for data to the managed objects at regular intervals, and receives the monitored value as the response. With the OVO threshold monitors, polling is done by the OVO agent. This means that there is no additional network traffic when polling MIB variables. The only additional network traffic occurs if a problem is detected that causes a message to be sent to the management server. This can be useful for monitoring application MIBs that are located on nodes running the OVO agent.
- ❑ Even if an object for which a MIB variable is monitored is not running an OVO agent, the threshold monitor can reduce network load. In this case, it is recommended to use an agent running on a machine located in the vicinity of the monitored object. In this case, any additional network load caused by MIB variable polling is restricted to a segment of the network, reducing overall network load and the workload of the management server.
- ❑ All features previously mentioned for the SNMP Trap Template also apply to the Monitor Template.

The MIB Data Collection facility allows the collection of SNMP-related data and it is provided by both NNM and OVO. It is recommended to use the MIB Data Collection facility when:

- ❑ the agent is not running on the node to be monitored, and there is no OVO managed node in the immediate vicinity
- ❑ the monitored values are to be collected and graphed at a later time.

OVO offers additional capabilities to achieve a tighter integration and many additional features in the areas described in the following subsections:

Alternative Message Sources

SNMP traps are only one of several different message sources for OVO. Logfile templates can be used to generate messages on OVO managed nodes in which a powerful pattern-matching mechanism is applied locally to forward relevant messages to the management server.

Network traffic is only necessary when a filtered message is transmitted by way of a reliable remote procedure call (RPC) connection. In comparison, SNMP traps use the unreliable UDP and can be lost if the trap interception process on the management server is not running.

NOTE The OVO agent buffers all messages which it cannot send if the management server is not accessible for any reason.

User Role Concept

The **user role concept** of OVO means that new operators can be configured with individually assigned message groups, applications, and managed nodes. Operators, therefore, have a customized view of their managed environment and see only the information from systems, devices and objects for which they are responsible.

A message can be owned by an operator, allowing only that operator to perform actions, or to escalate or acknowledge the message.

Advantages of an INSM Solution

OVO provides:

- ❑ Reflection of node status in IP Map windows
- ❑ Central configuration of distributed collection stations for topology detection
- ❑ HP OpenView Service Navigator integration on the management server

Starting from Scratch

To integrate a partner solution that is not currently integrated into either NNM or OVO, first make sure that you have a complete understanding of the way in which operators use the GUIs, and the underlying concepts of the two OpenView products. Based on this knowledge, you can start to define the functional specification of the integrated solution. You will need to decide what operators will see of the partner solution and how they are to access the additional functionality.

After defining what you want to show to operators (for example, which messages and actions) it is usually quite straightforward to decide on the capabilities to implement this functionality. As a general guideline, capabilities that are configured by way of the administrator's GUI are the preferred choice.

Obtaining Coexistence of NNM and OVO Integrations

If you plan to integrate your product into both NNM and OVO, you can choose between the following two strategies:

1. Develop an integration package for NNM and transform this into an OVO integration package.

   The OVO integration package resulting from this approach is not improved.

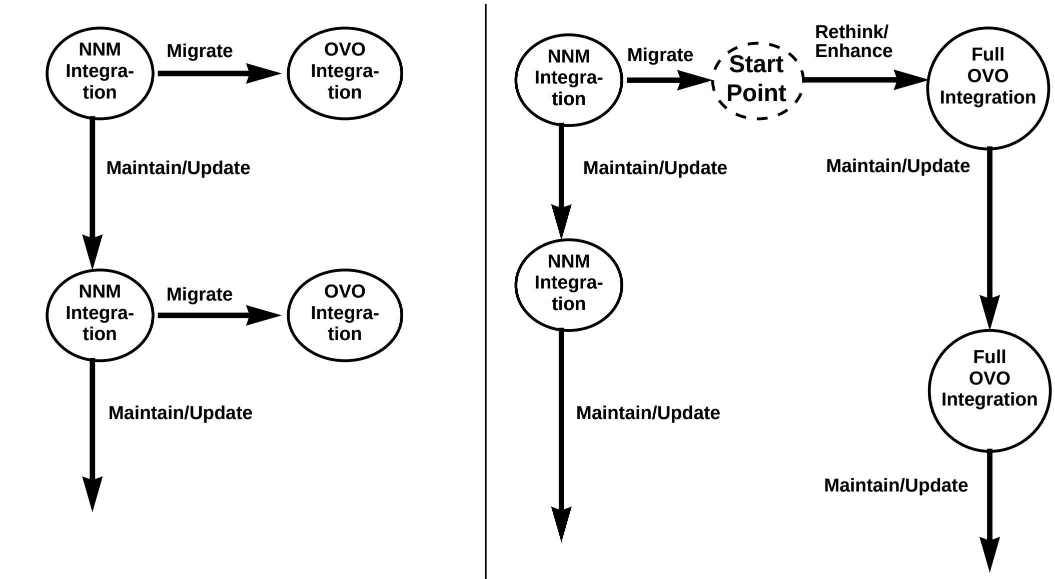
2. Develop an integration package for NNM and develop a separate integration package for OVO.

   It is always possible to develop the NNM integration package first, transform this to an OVO integration package, and then use the result as a starting point for an independent OVO integration package.

Figure 2-3 on page 80 illustrates the two strategies. Using strategy 1 only the NNM migration package needs to be maintained; the corresponding OVO integration package is always generated from the NNM integration package. You will need to use strategy 2 if you want to achieve a tight OVO integration that takes full advantage of the OVO features, you will need to use strategy 2. A tight integration provides maximum benefit to customers and is a prerequisite for achieving OVO certification.

Figure 2-3 on page 80 also shows that there are no other strategies available, as soon as you start to modify the OVO integration resulting from the migration process, you are using strategy 2. Consequently, the modified OVO integration can no longer be generated from the NNM integration and needs to be maintained separately. It is not possible to design an OVO integration first and to leverage from this for an NNM integration.

Figure 2-3 Comparison of Integration Strategies



Strategy 1:

Develop an integration package for NNM and transform this into an OVO integration package.

The Operations integration resulting from the migration process is not enhanced or improved.

Strategy 2:

Develop an integration package for NNM and develop a separate integration package for OVO.

As a starting point for the separate OVO integration package, the result of migrating a NNM integration might be used.

The following subsections summarize the advantages and disadvantages of the two strategies.

Strategy 1: Use Transformed NNM Integration in OVO

This strategy has the following advantages:

- ❏ Requires little integration effort

  You don't need all the advanced features of OVO, and you don't need to identify new integration capabilities.

- ❏ The OVO integration will work in the same way as the NNM integration.

  This means an easy switchover for operators that used NNM and will continue with OVO.

- ❏ Easy to maintain

  You need only maintain the NNM integration package.

  The OVO integration package is always generated from an NNM integration package, using the recommended migration process.

This strategy has the following disadvantages:

- ❏ You do not achieve a tight OVO integration as OVO integration points and features are not exploited.
- ❏ It is unlikely that this type of integration will satisfy the requirements for OVO certification because of the large amount of unused OVO functionality.

Strategy 2: Develop a Separate OVO Integration

This strategy has the following advantages:

- ❏ Use OVO features to increase the customer benefit of your integration, for example, by providing detailed instructions to operators. For more information, see "Leveraging From an Integration into NNM" on page 73.
- ❏ Tight OVO integration can be achieved. This is also a prerequisite for OVO certification of your integration.

This strategy has the following disadvantages:

- ❏ The resulting OVO integration package must be maintained as a separate package
- ❏ Increased integration effort and planning required

Summary of the Integration Process

The outcome of any integration is an **integration package** that contains OVO configuration information. Depending on the product you are integrating, you may need to modify the software of the partner solution or even implement additional processes, for example, when using the OVO APIs. In this case, the integration package would contain other items in addition to the OVO configuration information.

You may find it useful to answer the following questions as a rough guide to defining an integration strategy:

- ❑ What will an operator using OVO see of the partner solution?
- ❑ Which messages and applications will be available?
- ❑ Will there be any new message groups?
- ❑ Can you include automatic and operator-initiated actions?
- ❑ For which types of message can you provide instructions?
- ❑ Are there any numeric values that can be monitored by OVO?
- ❑ Does the partner solution include applications or tools that can be offered to OVO operators in the application desktop?

The integration process takes the following steps:

1. Select the integration capabilities you need to implement the required functionality.

   You may need to use more than one of the OVO integration capabilities to implement a certain functionality.

   The result of this step is a design plan for the integration.

2. Define the required configuration information using the OVO administrator's GUI. This includes:

   - Templates for the selected message sources: logfiles, SNMP traps, MPE/iX console messages, and calls to the OVO APIs. You will also need to define message and suppress conditions.
   - Instructions for operators, automatic actions, and operator-initiated actions to be coupled with messages.
   - Threshold monitors

- Interfaces to trouble-ticket and notification systems
- Tools for operators and administrators that are to be integrated into the OVO Application Desktop
- Platforms that will be supported

3. If required, modify the partner solution software or implement additional processes.

   You can implement additional processes using programming languages or shell scripts, depending on whether you will use the OVO APIs or the command line interfaces.

NOTE APIs and command line interfaces are available for the agents and the management server.

4. Create the configuration information fileset by selectively downloading the required configuration information in the OVO administrator’s GUI.
5. Add other required files to the configuration information to create the integration package.

   Other required files can be the necessary executables (scripts, commands, actions) and HP OpenView application registration files (ARFs).

6. Bundle the integration package with the partner solution product as a separate fileset or as a separate product.

   Add installation instructions to the partner solution documentation or, if the integration package is a product of its own, create a separate installation guide.

   The installation information must include information to enable customers to customize the integrated solution to meet their needs.

7. Install or restore the integration package at the customer’s sites by uploading the configuration information.
8. Show the customer how to use the OVO administrator’s GUI to do the final customization, and to assign and distribute new or changed templates to the managed nodes.

The Role of Configuration Data in an Integration

Figure 1-1 on page 32 shows an overview of the integration process in which it distinguishes between defining an integration strategy and implementing an integration. The figure shows that the result of an integration process is an **integration package** which enables the partner solution and OVO to work together smoothly.

An integration package always contains OVO configuration information, which includes definitions for some, or all, of the following object classes:

❑ Applications and application groups
❑ Instruction text interfaces
❑ Managed nodes
❑ Message groups
❑ Node defaults
❑ Node groups
❑ Node Hierarchies
❑ Notification services
❑ Templates and template groups
❑ User and user profiles
❑ Action, command, and monitor executables
❑ Administrator configuration
❑ Database maintenance (configuration of OVO internal database)
❑ Escalation manager configuration
❑ Event correlation libraries
❑ Event correlation modules
❑ Management server configuration
❑ Message forwarding configuration
❑ Regroup conditions
❑ Responsible manager configuration
❑ Trouble ticket configuration

Configuration data in the local database determines the operational behavior of OVO. Consequently, the configuration data required by OVO to interact with a partner solution must be incorporated into the OVO internal database using the configuration download and upload utilities of OVO.

To download configuration data from OVO you can either select `Actions:Server->Download Configuration...` from the menu of the administrator's GUI or use the configuration download command `opccfgdwn(1M)` at the command line. Both methods download configuration information into a tree structure of flat files. You can later upload configuration information into the OVO internal database using the `opccfgupld(1M)` command.

Note that service data of the HP OpenView Service Navigator is kept in a separate file and can currently not be downloaded or uploaded.

These configuration utilities enable a partner to define a configuration using the OVO GUI and then to download the configuration to a file. The resulting file can then be shipped and installed with the partner software, or provided as a separate product that can be uploaded into the customer's OVO installation. From the customer's perspective, an "out-of-the-box" integration of the partner solution and OVO is provided.

Figure 2-4 on page 86 summarizes how the OVO configuration download and upload capabilities are used to define a configuration and then to distribute it to customers.

Figure 2-4 shows the following steps:

Step 1: Define the OVO configuration.

Step 2: Download the OVO configuration to the flat files using the GUI or `opccfgdwn(1M)` command.

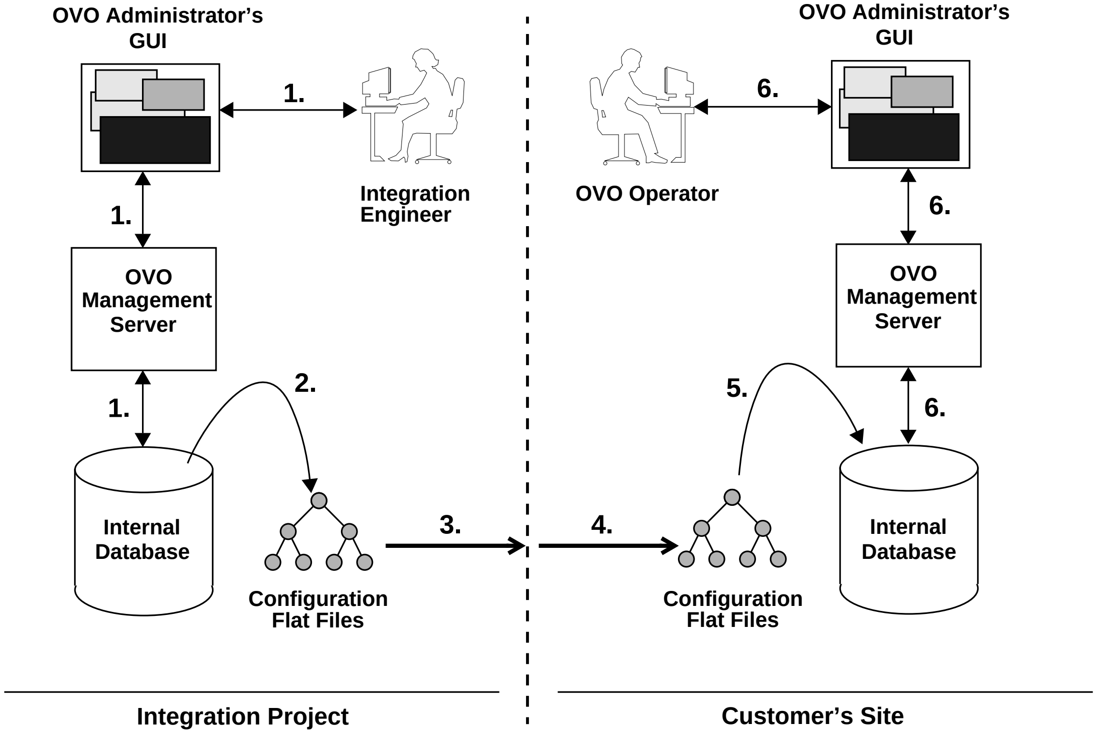
Step 3: Package the configuration with the product and ship to the customer.

Step 4: Install the product, including the OVO configuration.

Step 5: Upload the OVO configuration from the flat files

Step 6: Use the new, uploaded configuration.

Figure 2-4 Handling OVO Configuration Information



Although configuration files are not encrypted, it is strongly discouraged to edit them as this can lead to problems when uploading the configuration. The configuration should always be defined using the administrator's GUI.

The uploading and downloading capabilities, and the structure of the configuration file tree are described in more detail in Chapter 8, "Creating and Distributing an Integration Package," on page 273.

3 Using the Integration Capabilities of the OVO Motif-based GUI

In This Chapter

This section describes how to use features of the OVO administrator's GUI to integrate applications into OVO.

Event Integration Through Messages

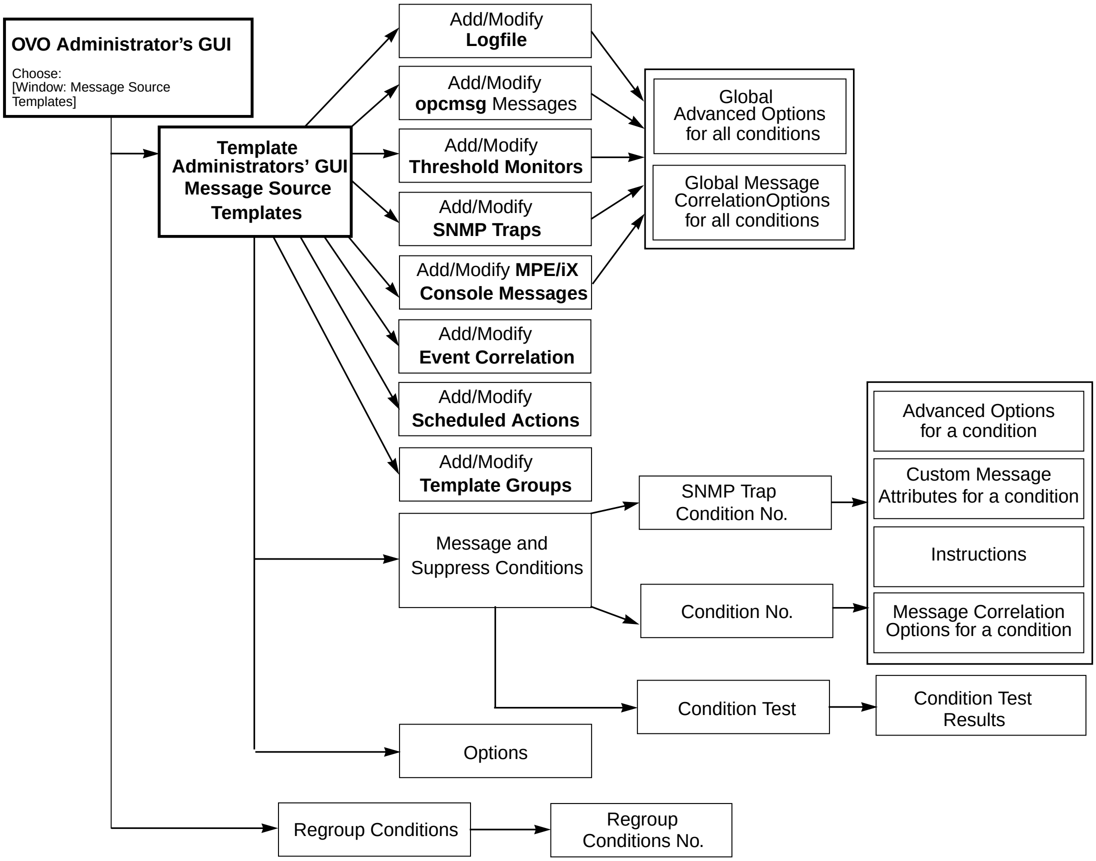
This section describes in more detail the various methods provided by the OVO GUI that can be used to generate an integration package.

Configuring Messages in the Message Source Templates Window

Any problem that causes the partner solution to generate an error message, should also trigger an OVO message. When correctly configured, message templates guarantee that OVO operators are always informed about problems when they occur in the partner solution.

The templates for all message sources, including threshold monitors, are listed in the `Message Source Templates` window. The windows used to create, or modify templates for the various sources can all be accessed from this window, as shown in Figure 3-1. The `Add/Modify <Template>` windows differ slightly for each source, but the concepts involved in defining templates are similar.

Figure 3-1 Accessing the Template Configuration Windows


OVO Administrator's GUI
Choose:
[Window: Message Source Templates]
Template Administrators' GUI Message Source Templates
Add/Modify Logfile
Add/Modify opcmsg Messages
Add/Modify Threshold Monitors
Add/Modify SNMP Traps
Add/Modify MPE/iX Console Messages
Add/Modify Event Correlation
Add/Modify Scheduled Actions
Add/Modify Template Groups
Global Advanced Options for all conditions
Global Message CorrelationOptions for all conditions
Message and Suppress Conditions
SNMP Trap Condition No.
Condition No.
Advanced Options for a condition
Custom Message Attributes for a condition
Instructions
Message Correlation Options for a condition
Condition Test
Condition Test Results
Options
Regroup Conditions
Regroup Conditions No.

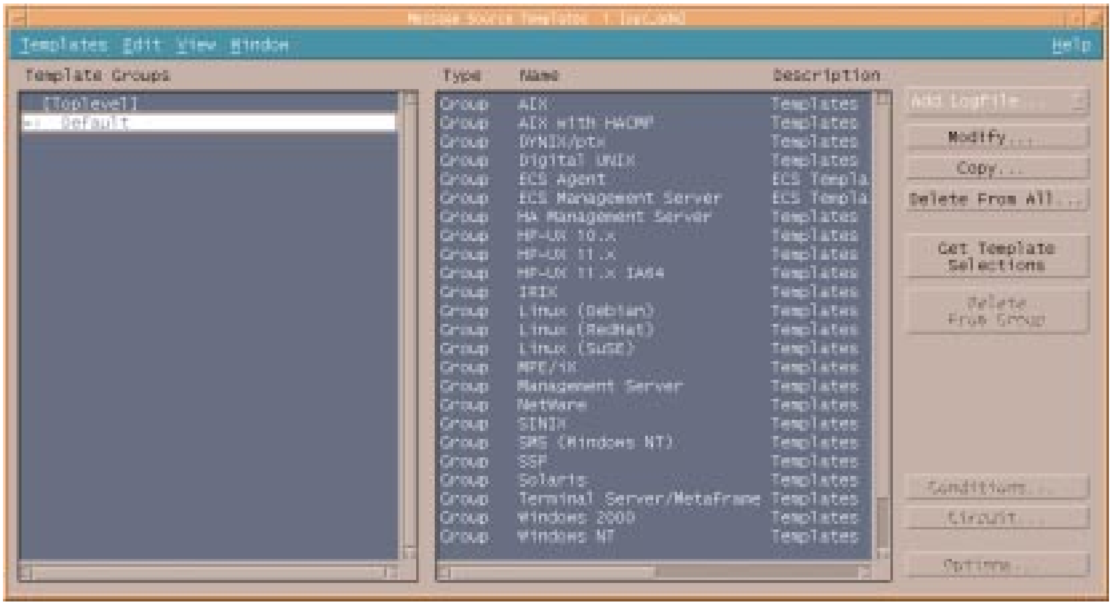
To open the `Message Source Templates` window, you can:

- ❏ Login as OVO administrator

  Select `Windows: Message Source Templates` from the OVO administrator's GUI.

- ❏ Login as the OVO template administrator

Figure 3-2 Message Source Templates Window



Type	Name	Description
Group	AIX	Templates
Group	AIX with HACMP	Templates
Group	DYNIX/ptx	Templates
Group	Digital UNIX	Templates
Group	ECS Agent	ECS Templa
Group	ECS Management Server	ECS Templa
Group	HA Management Server	Templates
Group	HP-UX 10.x	Templates
Group	HP-UX 11.x	Templates
Group	HP-UX 11.x IA64	Templates
Group	IRIX	Templates
Group	Linux (Debian)	Templates
Group	Linux (RedHat)	Templates
Group	Linux (SuSE)	Templates
Group	MPE/iX	Templates
Group	Management Server	Templates
Group	NetWare	Templates
Group	SINIX	Templates
Group	SMS (Windows NT)	Templates
Group	SSP	Templates
Group	Solaris	Templates
Group	Terminal Server/MetaFrame	Templates
Group	Windows 2000	Templates
Group	Windows NT	Templates

The following sections describe the OVO message sources and the ways in which they can be used to integrate with OVO. For more detailed information about the message sources, refer to the OVO Administrator's Guide to Online Information and the *OVO Concepts Guide*.

For a description of event correlation services and scheduled actions shown in Figure 3-1 on page 90, refer to the OVO Administrator's Guide to Online Information.

Defining Templates for Logfile Encapsulation

If an application writes events into a logfile, the OVO **logfile encapsulator** can interpret these events and incorporate them in the OVO Message Browser.

An OVO logfile template specifies the events that are of interest to OVO, for example:

- ❏ the file to be analyzed and the polling interval
- ❏ the defaults to be applied to the generated messages (severity, message group, etc.)
- ❏ a set of options
- ❏ a set of conditions

A condition either specifies which events should generate messages to be forwarded to the management server (**message conditions**) or which events should be suppressed (**suppress conditions**).

NOTE Do not mistake the `Monitoring Options` part of the `Add/Modify Logfile` window, for the threshold monitoring facility of OVO which is described in the section “Defining Templates for Threshold Monitors” on page 100.

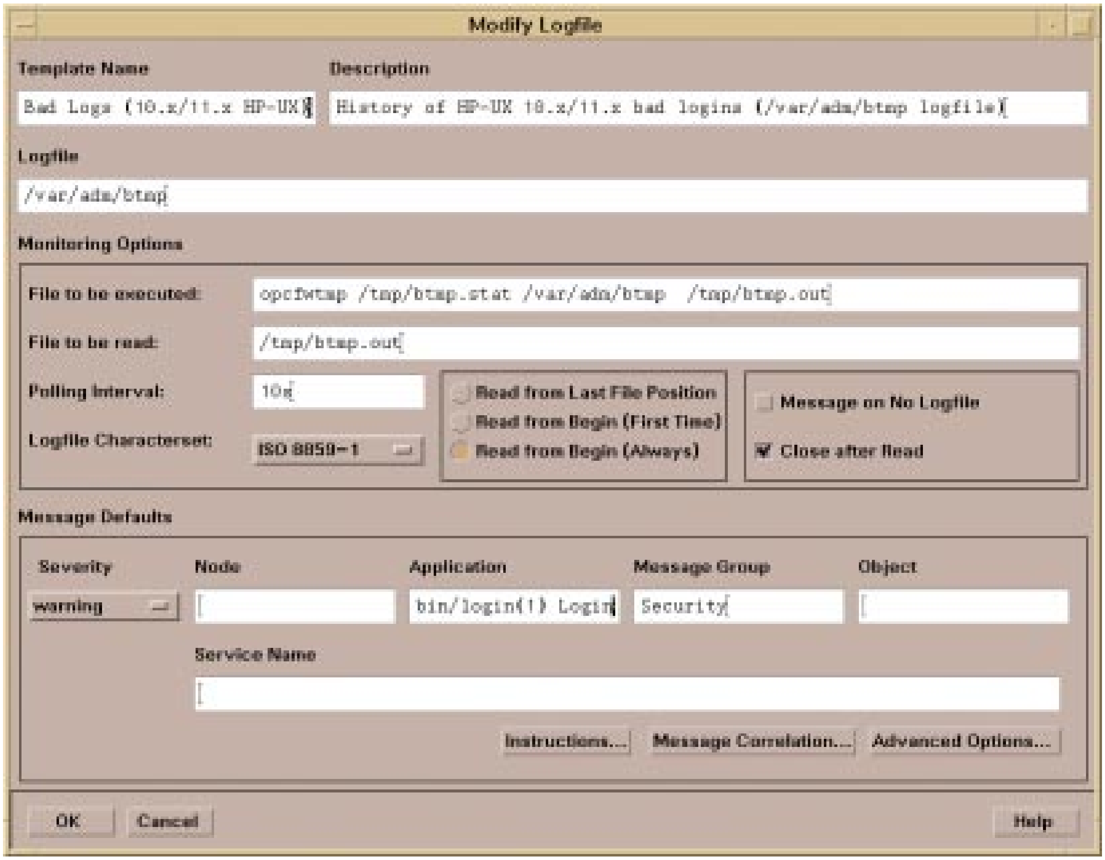
The template defaults are applied in the following cases:

- ❏ When defining a new condition for a given template.
- ❏ When a message is generated that does not match either a message condition or a suppress condition, this is referred to as an **unmatched message**. When defining the template, the options let you choose whether unmatched messages are ignored or forwarded to the OVO management server.

You can always change the defaults for the entire template or for individual conditions as required.

To define a logfile template, click on the `[Add Logfile]` button in the `Message Source Templates` window. The `Add Logfile` window shown in Figure 3-3 enables you to specify the logfile to be encapsulated, the monitoring options for the logfile, and the defaults for the generated message.

Figure 3-3 Add/Modify Logfile Window



Modify Logfile

Template Name: Bad Logs (10.x/11.x HP-UX

Description: History of HP-UX 10.x/11.x bad logins (/var/adm/btmp logfile)

Logfile: /var/adm/btmp

Monitoring Options

File to be executed: opcfwtmp /tmp/btmp.stat /var/adm/btmp /tmp/btmp.out

File to be read: /tmp/btmp.out

Polling Interval: 10s

Logfile Characterset: ISO 8859-1

○ Read from Last File Position
○ Read from Begin (First Time)
● Read from Begin (Always)

□ Message on No Logfile
☑ Close after Read

Message Defaults

Severity: warning

Node:

Application: bin/login(1) Login

Message Group: Security

Object:

Service Name:

For information about the fields in the `Add/Modify Logfile` window, see the OVO Administrator’s Guide to Online Information.

For information about setting Advanced Options for a message source, or using the OVO Instructions Interface see:

- ❑ “Setting Advanced Options for a Message Source Template” on page 116;
- ❑ “Setting Message Correlation Options for a Message Source Template” on page 118.
- ❑ “Adding Instructions, Annotations, Automatic- and Operator-initiated Actions” on page 136.

Using the Logfile Monitoring Options

This section describes how to use the encapsulator options for different types of logfile. The options `Close After Read` and `Message On No Logfile` must be considered in each individual situation.

The options `File to be Read` and `Command to be Executed` enable OVO to observe any type of file, including entire directories or binary files. The logfile encapsulator detects the modification of the file and runs the specified program which converts the required information into a readable form for the encapsulator.

❏ **Binary logfiles**

For a binary logfile, you need a program to convert the binary data into readable ASCII text. When the application generates binary files, the following options are recommended:

- Enter the program required to convert the binary file into the `File to be executed` field.
- Enter the name of the file containing the converted, readable text into the `File to be read` field.
- Select the `Read from begin (Always)` option if the `File to be read` will be completely overwritten.

❏ **Logfiles without a constant name, for example, logfile<number> or logfile<date>**

The logfile encapsulator lets you observe directories that are modified simultaneously if one of the files contained in the directory changes. You need to use a subprogram or script that must discover the modified files and extract the new information into another file to be read by the encapsulator as “File to be read”. For logfiles with a changing name, the following options are recommended:

- Enter the program required to discover the modified files into the `File to be executed` field.
- Enter the name of the file containing the new information into the `File to be read` field.
- Select the `Read from begin (Always)` option if the `File to be read` will be completely overwritten.

- **Logfiles that decrease in size**

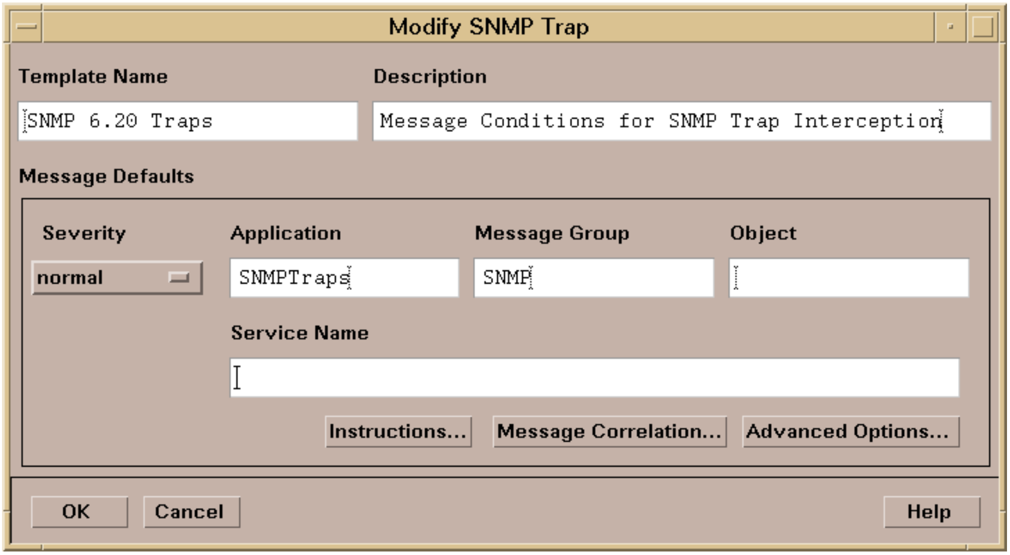
  You may find that some logfiles may get smaller if they are completely overwritten by the writing process. It is not possible to recognize new information as the encapsulator stores only the file status, and not the status of its contents. To read this type of logfile, select the "Read from begin (Always)" option.

Defining Templates for SNMP Trap Interception

If an application generates SNMP traps, OVO can intercept them and incorporate them into the Message Browser.

Figure 3-4 shows the window for adding an SNMP trap template.

Figure 3-4 Add/Modify SNMP Trap Window



Modify SNMP Trap

Template Name: SNMP 6.20 Traps

Description: Message Conditions for SNMP Trap Interception

Message Defaults

Severity: normal

Application: SNMPTraps

Message Group: SNMP

Object:

Service Name:

For information about the fields in the `Add/Modify SNMP Trap` window, see the OVO Administrator’s Guide to Online Information.

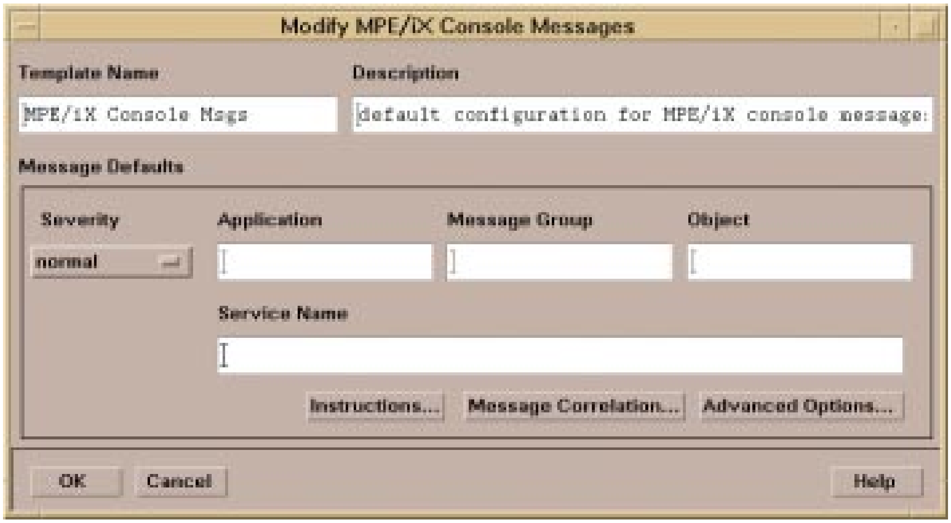
For information about setting Advanced Options for a message source, or using the `Instructions Interface` see:

- ❏ “Setting Advanced Options for a Message Source Template” on page 116;
- ❏ “Setting Message Correlation Options for a Message Source Template” on page 118.
- ❏ “Adding Instructions, Annotations, Automatic- and Operator-initiated Actions” on page 136.

Defining Templates for MPE/iX Console Message Interception

OVO can intercept all MPE/iX console messages, and it uses the default categorization and grouping defined in the templates provided with the OVO software.

Figure 3-5 Add/Modify MPE/iX Console Messages Window



Template Name: MPE/iX Console Msgs

Description: default configuration for MPE/iX console message:

Severity: normal

For information about the fields in the `Add/Modify MPE/iX` window, see the OVO Administrator's Guide to Online Information.

For information about setting Advanced Options for a message source, or using the `Instructions Interface` see:

- ❏ "Setting Advanced Options for a Message Source Template" on page 116;
- ❏ "Setting Message Correlation Options for a Message Source Template" on page 118.
- ❏ "Adding Instructions, Annotations, Automatic- and Operator-initiated Actions" on page 136.

Defining Templates for Messages Sent to the OVO Message Interface opcmsg(1|3)

This section describes how to create or modify a template that intercepts messages sent to the OVO message interface - the function opcmsg(1|3) of the **Agent Message API**.

The partner solution can use the OVO message C library routine `opcmsg(3)` or command `opcmsg(1)` to route messages in the standard OVO format to any OVO agent. Messages sent through the API or command are intercepted by OVO at the managed node on which the command or API are invoked. This allows a distributed handling of messages across the network.

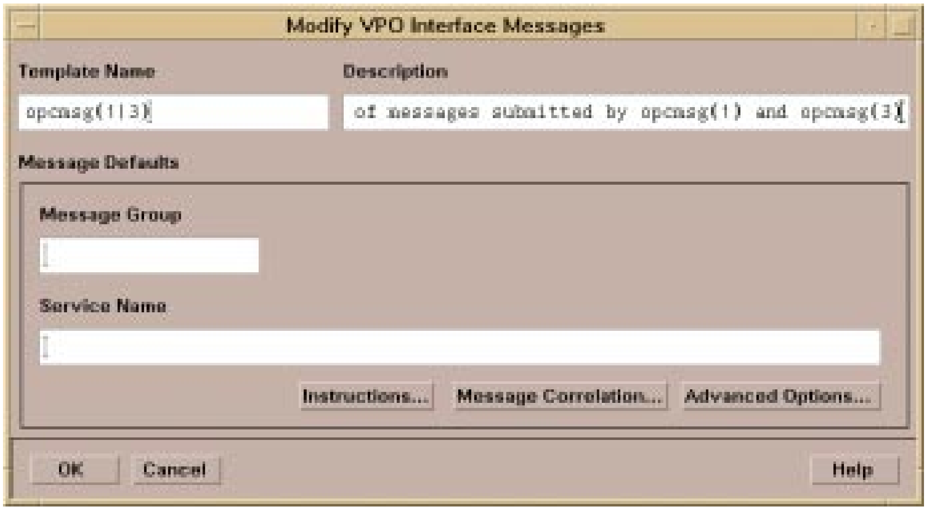
The following example shows how to create an OVO message with the `opcmsg(1)` command:

❏ This example submits a critical message issued by the script `diskwatcher.sh` with the related object `/usr/lib` which belongs to the message group DB:

```
opcmsg sev=critical msg_grp=DB appl=diskwatcher.sh \
node=laurax.bbn.hp.com \
msg_text=”DSET vouchers FULL: No further processing \
on vouchers possible!” \
object=”/usr/lib”
```

For all messages created by the OVO Agent Message API, the same processing is done as for other message sources. You can filter messages, classify them, set attributes, and define actions or other options. The description of the message handling is almost the same as for the other message sources, with only the specification of the message source being different.

Figure 3-6 **Add/Modify OVO Interface Messages Window**



Modify VPO Interface Messages

Template Name: opcmsg(1|3)

Description: of messages submitted by opcmsg(1) and opcmsg(3)

Message Defaults

Message Group:

Service Name:

For information about the fields in the `Add/Modify OVO Interface Messages` window, see the OVO Administrator's Guide to Online Information.

For information about setting Advanced Options for a message source, or using the `Instructions Interface` see:

- ❏ "Setting Advanced Options for a Message Source Template" on page 116;
- ❏ "Setting Message Correlation Options for a Message Source Template" on page 118.
- ❏ "Adding Instructions, Annotations, Automatic- and Operator-initiated Actions" on page 136.

Defining Templates for Threshold Monitors

This section describes the different types of monitors that are available, and how to define your own monitors using the OVO administrator's GUI.

Overview of OVO Monitoring Capabilities

OVO enables the monitoring of numeric values that are determined either by:

❏ **Program**

Calling a program at regular intervals that submits a numeric value to OVO.

❏ **External Program**

Accepting numeric values that are determined by some external mechanism and passed to OVO. These values can be accepted at any time.

❏ **MIB variables**

Polling MIB variables at regular time intervals.

The primary mechanism for monitoring numeric values in OVO is **threshold monitoring**. Threshold monitors can be set up to use any of the above methods to determine new values. Threshold monitors can, therefore, be used to poll MIB variables and to monitor any other values.

Threshold monitors are designed to watch numeric values which might indicate problems, and to generate messages as soon as a potential problem is detected. It is, however, important to avoid overwhelming operators by generating many redundant messages. To prevent such a flood of redundant messages, OVO threshold monitors enable redundant messages to be filtered out, for example, by defining time intervals after which a new message may be generated. As with other OVO message sources, the full set of action options (automatic and operator-initiated actions, automatic acknowledgment, etc.) can be configured for the messages generated by a threshold monitor, together with instructions, external notification, and trouble ticket services. Threshold monitors do not, by default, store the data that has been collected.

A second mechanism, specifically for polling MIB variables and referred to as **MIB Data Collection**, is also provided by OVO. In contrast to threshold monitors, this mechanism can be configured to store the collected data which can be used to analyze trends in monitored variables by plotting the values against time. This type of monitor also supports threshold values but does not provide their sophisticated filtering capabilities. Events can also be generated based on exceeded thresholds and on returning to “normal” values, but it is not straightforward to set up an action to resolving a problem.

Monitoring MIB Variables

When deciding whether to use a threshold monitor or MIB data collection, consider the following points:

❑ To collect and store data for the later plotting of graphs, it is recommended to use MIB data collection.

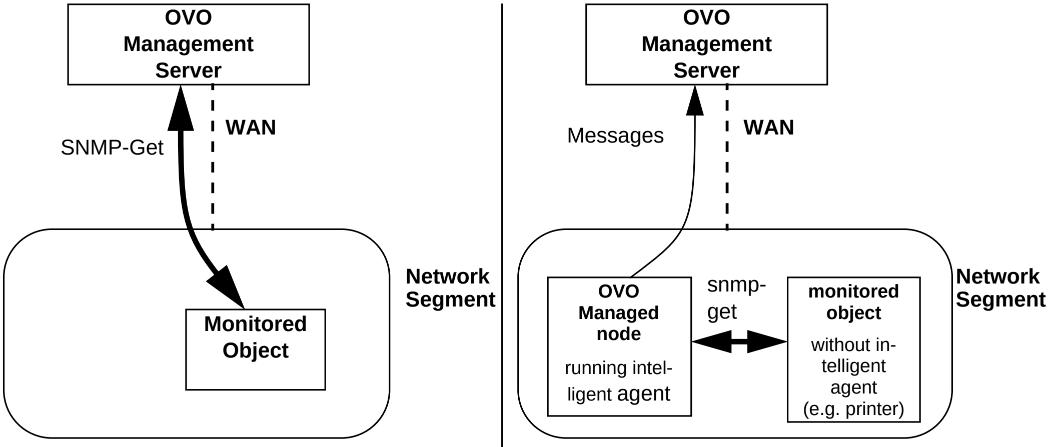
❑ To generate a message as soon as a monitored value indicates a problem, it is recommended to use threshold monitoring.

❑ If the OVO management server is at a location remote from the objects for which MIB variables are monitored, and if network load and management server workload are issues, threshold monitoring might be more suitable for the following reasons:

  - With MIB data collection, the OVO management server polls the managed objects at regular intervals with a request for data and receives a response.
  - With threshold monitoring, the OVO intelligent agent polls the monitored object. If the object is a node running the OVO agent, additional network traffic, due to MIB variable polling, is not generated. The only additional network traffic is caused when a problem is detected, and a message must be sent to the management server.

- Even if the object for which a MIB variable is monitored is not running an OVO intelligent agent, the threshold monitor can still reduce the network load. In this case, it is recommended to use an intelligent agent running on a machine located close to the monitored object. Additional network load caused by MIB variable polling is, therefore, restricted to a small part of the network. This reduces the overall network load and the workload of the management server. If the OVO management server is connected to the managed objects over a WAN, this situation can be especially beneficial in terms of network load.

Figure 3-7 **Comparison of Network Traffic Caused Monitoring Methods**



MIB Data Collection:

Considerable additional network traffic caused between the OVO management server and the object for which the MIB variable is monitored

Threshold Monitoring:

Considerable additional network traffic caused only between the object to be monitored and a nearby agent

Using Threshold Monitoring to Generate Messages

You can configure OVO to monitor managed nodes at regular time intervals, during which the monitored value is calculated and compared against a threshold value. If the threshold is exceeded, whether a message is generated depends on the type of monitor selected and on the “history” of values relative to the reset value.

If a message is generated, you can set message attributes and custom message attributes, and provide instructions, annotations, and actions, in the same way as for any other message generated by matching a message condition. When a message is generated and all message attributes are set, it is sent to the management server. If a local automatic action has been defined, this is also started in parallel. Note that only when the threshold is exceeded does OVO monitoring cause network traffic.

The threshold monitoring scripts, provided by the user, are invoked by the monitoring agent in the configured time interval, except for values monitored externally. The **monitor agent** checks the success of the scripts by interpreting the exit value. If the exit value is not equal to 0, indicating script failure, the monitor agent sends an appropriate message to the Message Agent that forwards the message to the management server.

The script collects one or more current values of the objects to be monitored and sends them to the monitor agent through a C library routine or a command interface provided by OVO, with a call to `opcmon(1|3)`. See the man pages *opcmon(1)* and *opcmon(3)* for more information.

The check of externally monitored objects is not invoked by the monitor agent, however, it checks whether a value from an object has arrived through a call to `opcmon(1|3)` at least every 15 seconds.

The monitor agent checks the received values against the configured thresholds and creates user-defined messages for any exceeded threshold values. This happens for the values of all types of objects (program, external or SNMP MIB) that arrive at the monitor agent. If the monitor agent receives a value of an object which was not configured, it will generate an opcerror log entry with an appropriate warning on the managed node. For example:

File is: `/var/opt/OV/log/OpC/`*`<managed_node>`*`/opcerror`

OVO message: `OpC30-613`

```
08/05/00 10:39 WARNING Monitor Agent(10827):
Can’t find object ‘<Monitored Object>’.
Unknown monitor ’<Monitored Object>’.
Ignoring received value.
```

If a monitored object should be checked by a user-provided script invoked by the monitor agent, and a corresponding name-value pair is not returned, (that is, the script/program does not call opcmon) the monitor agent will generate a warning message and send it to the management server. For example, the message OpC30-608:

```
Can’t retrieve value for monitor <Monitored Object>.
Suppressing further error messages.
```

Message Generation Policies and Message Filtering

Threshold monitoring enables you to choose either a minimum or maximum threshold:

❑ **Maximum threshold monitoring**

As long as the values received from the monitored object are less than the threshold value, no messages are generated. If the values reach or exceed the threshold value, then OVO generates the configured message for this object.

❑ **Minimum threshold monitoring**

As long as the actual values received from the monitored object are above the threshold value, no messages are generated. If they reach or drop below the threshold value, then OVO generates the configured message for this object. The generation of further messages depends on how the message generation type is configured, see below.

To prevent redundant messages from cluttering the operators' Message Browsers, OVO provides three mechanisms you can use to generate further messages after the value of a monitored object crosses a defined threshold:

❑ Message generation on crossing threshold **with Reset**

If the monitored value reaches or crosses the threshold, OVO generates one message. Before OVO generates further messages, the monitored value must either:

- Drop below the reset value for maximum threshold monitoring, or
- Rise above the reset value for minimum threshold monitoring.

❑ Message generation on crossing threshold **without Reset**

If the monitored value reaches or crosses the threshold, OVO generates one message. Before OVO generates further messages, the monitored value must either:

- Drop below the threshold value for maximum threshold monitoring, or
- Rise above the threshold value for minimum threshold monitoring.

❏ Message generation **continuously**

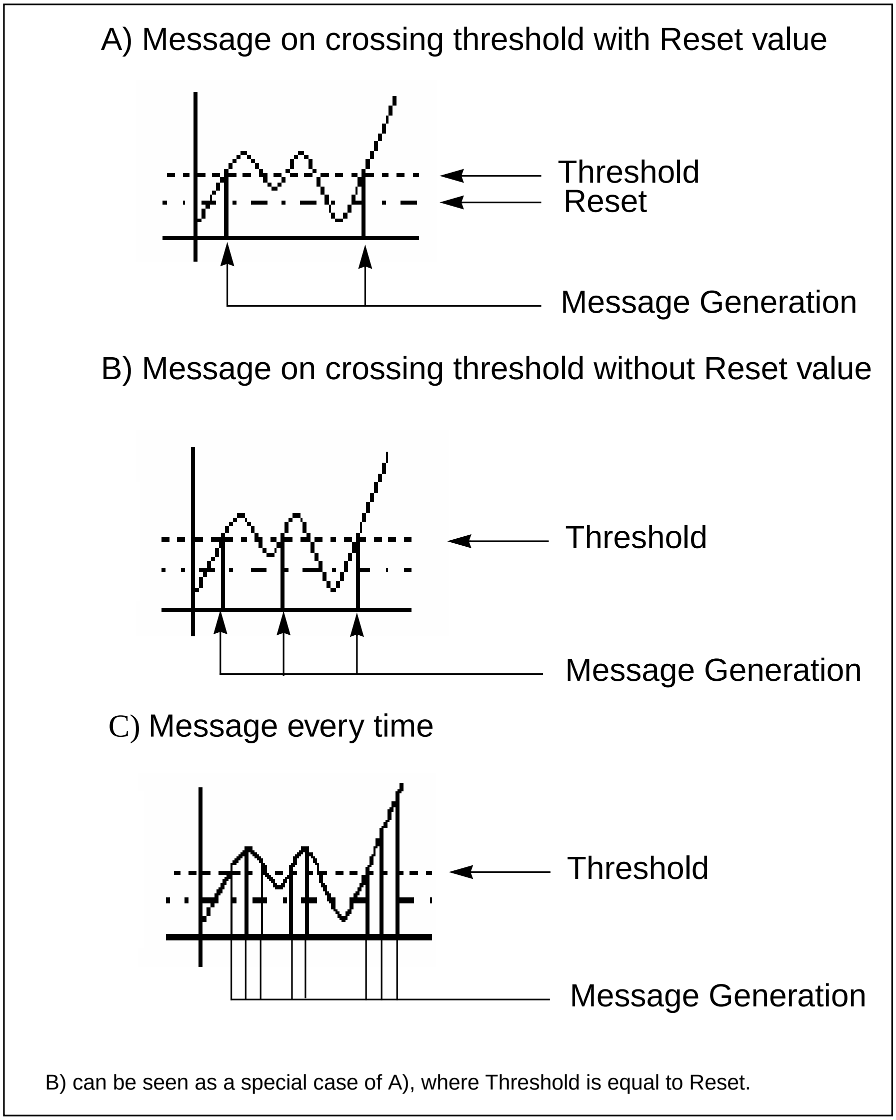
OVO generates a message each time the monitored value is either:

- equal to or greater than the threshold for maximum threshold monitoring, or
- equal to or less than the threshold for minimum threshold monitoring.

The frequency of the message generation in this case depends on the configured polling interval.

It is also possible to specify that a message is only to be generated when the monitored value exceeds the threshold for a longer duration than a defined period. You can do this by setting the `Duration` parameter in the `Add/Modify Threshold Monitor` window.

Figure 3-8 Message Generation Policies For Maximum Thresholds



A) Message on crossing threshold with Reset value

B) Message on crossing threshold without Reset value

C) Message every time

B) can be seen as a special case of A), where Threshold is equal to Reset.

Types of OVO Monitor Available

OVO supports the following types of threshold monitor:

❑ **Monitoring With User-defined Programs or Scripts**

This method uses a program or script provided by the user that is invoked by the OVO Monitor Agent (**opcmona**). It collects the monitored value and transmits it to the OVO management server using the `opcmon(1|3)` C library API or command.

Monitoring Performance Metrics

Performance metrics are collected by way of the embedded performance component. Use the following syntax in the `Monitor Program or MIB ID` field:

OVPERF\\<*data source*>\\<*object*>\\<*metric*>

Where:

<*data source*>

Identifies the data source. When collecting metrics from the embedded performance component <*data source*> must be set to CODA.

<*object*>

Identifies the name of the object class to be monitored.

The performance component collects the following object classes:

- Global (object name: GLOBAL)
- CPU (object name: CPU)
- Network interface (object name: NETIF)
- File system (object name: FS)
- Disk (object name: DISK)

<*metric*>

Identifies the metric to be collected.

The metrics that are currently available for each object class are described in more detail on the following Web page:

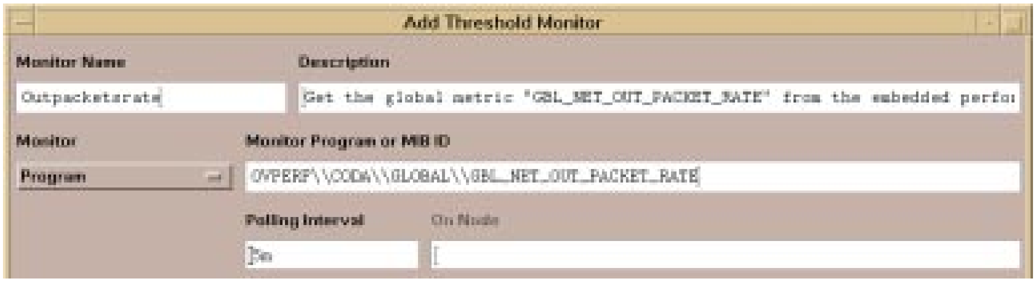
http://<*management_server*>:3443/ITO_DOC/<*lang*>/
manuals/EmbedPerfAgent_Metrics.htm

In this instance, *<management_server>* is the fully qualified hostname of the management server and *<lang>* is one of the following:

`C`	for English environments
`japanese`	for Japanese environments

Figure 3-9, Setting Performance Thresholds, shows how to enter the syntax explained above in threshold monitor templates.

Figure 3-9 Setting Performance Thresholds



Add Threshold Monitor

Monitor Name: Outpacketsrate

Description: Get the global metric "GBL_NET_OUT_PACKET_RATE" from the embedded perfo

Monitor: Program

Monitor Program or MIB ID: OVPERF\\CODA\\GLOBAL\\GBL_NET_OUT_PACKET_RATE

Polling Interval: 5m

On Node:

The performance component constantly collects all platform-generic and typical metrics. The collection interval is by default five (5) minutes and cannot be changed. The data is kept in the data store for up to five (5) weeks. When the database is full after five (5) weeks of data have been collected, the oldest data is rolled out one week at a time and deleted.

See the *OVO Administrator's Reference* for more information about troubleshooting the embedded performance component.

❑ **External Monitoring**

This type of monitoring is similar to the previous method, but it differs in that OVO does not invoke the user's monitor scripts or programs. This means that monitoring is done externally and the OVO monitor agent only checks the values of an externally monitored object against the defined threshold value. The monitor agent checks whether value pairs (for example, object name=<value>) have been received for comparison at least every 15 seconds.

- ❑ **SNMP MIB variables**

  OVO can also monitor SNMP MIB variables. OVO queries SNMP MIB variables on the agents at defined intervals, and checks the output against the threshold values. If the result is above the maximum or below the minimum threshold value, OVO generates a message that is sent to the management server by way of the message agent.

Monitor scripts or programs should be provided by the integrator. The scripts or programs are either assumed to exist on the respective managed nodes (e.g., if they are part of a monitored application) or they can be distributed to the managed nodes by OVO. If it is required to distribute these scripts with OVO, they must be copied into the following directory on the OVO management server:

```
/var/opt/OV/share/tmp/OpC_appl/<app_name>/<lang>\
/EXECUTABLES/...
```

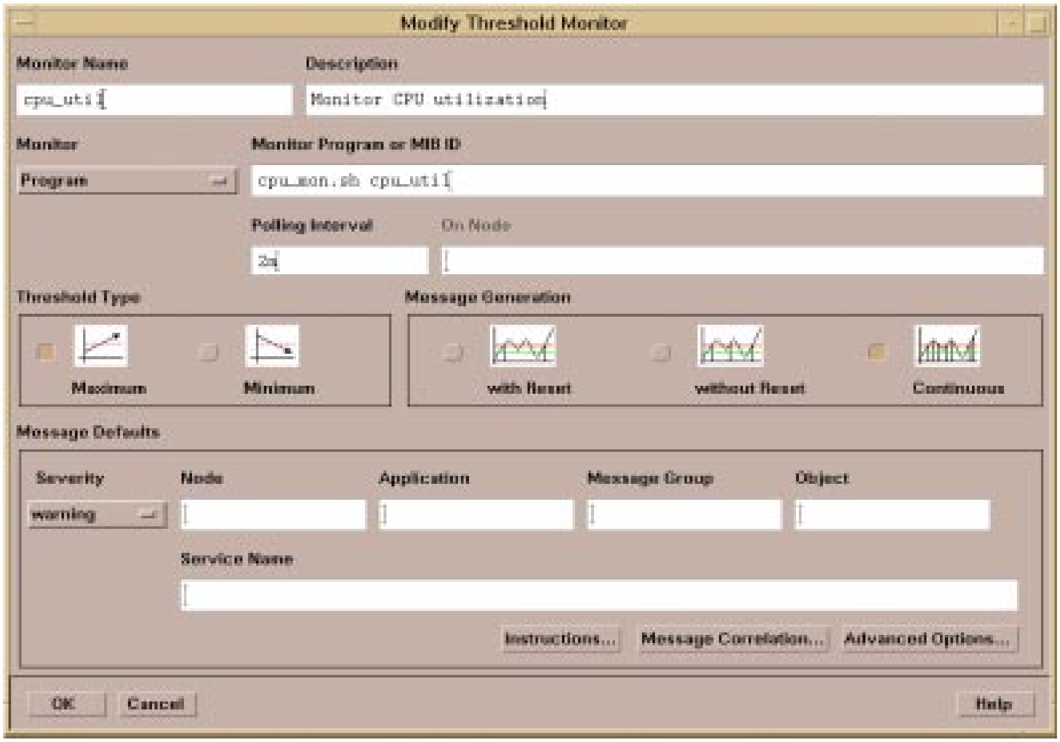
From this directory, the scripts can be downloaded automatically, uploaded to the customer's management server, and distributed to the managed nodes.

Integrating Monitors into OVO

A separate monitor is defined for each type of value to be monitored. You can specify message and suppress conditions for the monitor templates in the same way as for other message sources. To display a list of monitors, select the appropriate group in the `Message Source Templates` window. To open this window, choose `Window: Message Source Templates` from any of the toplevel windows. You can use this window to add new monitors, as well as modify, copy, and delete monitors.

When you choose to add or modify a monitor, the `Add/Modify Threshold Monitor` window is displayed, see Figure 3-10.

Figure 3-10 Add/Modify Threshold Monitor Template Window



Modify Threshold Monitor

Monitor Name: cpu_util

Description: Monitor CPU utilization

Monitor: Program

Monitor Program or MIB ID: cpu_mon.sh cpu_util

Polling Interval: 2m

On Node:

Threshold Type: ● Maximum ○ Minimum

Message Generation: ○ with Reset ○ without Reset ● Continuous

Message Defaults

Severity: warning

Node:

Application:

Message Group:

Object:

Service Name:

For information about the fields in the `Add/Modify Threshold Monitor` window, see the OVO Administrator's Guide to Online Information.

You can set conditions for a monitor template using the `Condition No.` window in the same way as for other message sources. For information about the fields in the `Condition No.` window for monitors, see the OVO Administrator's Guide to Online Information.

As in all condition definition windows, an `[Instructions...]` button enables you to specify help text, instructions, and information about the monitored object and the threshold value, and/or about the configured actions.

Sending Values Over the OVO Monitoring API or Command

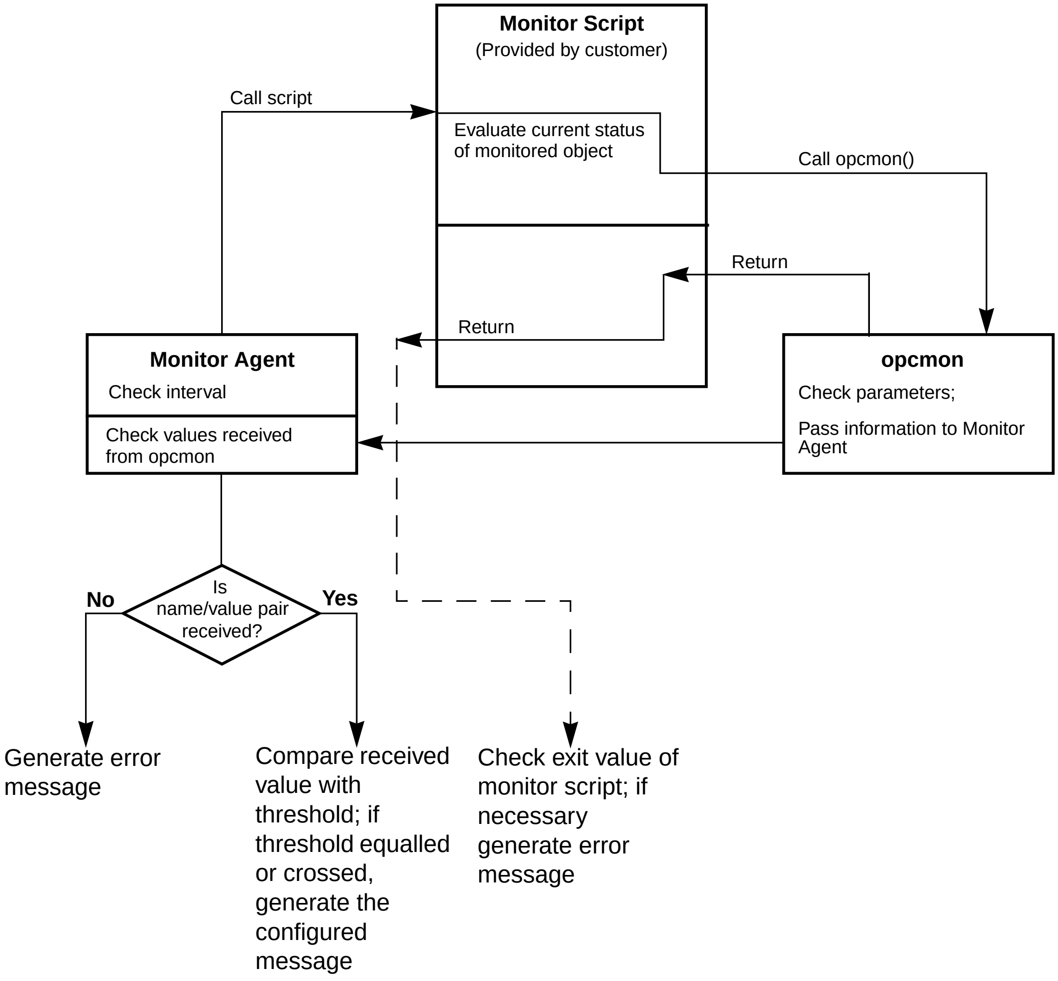
OVO provides an API and a command `opcmon(1|3)` to allow applications to send monitoring values from the managed nodes to the OVO management server. This API/command should be used at the managed nodes for distributed, exception-based management.

The following steps describe the process flow and the interaction between the monitor agent of OVO (`opcmona`) running on the managed node, and the monitoring script or program provided by the user:

1. OVO checks the intervals defined to invoke the monitoring scripts.
2. OVO calls the specified script or program.
3. As part of the script or program, the actual monitoring values are collected.
4. The script calls `opcmon(1|3)` (usually once) to pass the value(s) to the monitor agent of OVO.
5. The monitor agent checks the value received from `opcmon(1|3)` and returns a value to the caller of the script: zero for successful execution, one for a failure.
6. The monitor agent reads the value against the predefined thresholds and generates a message if an exception occurs.
7. The return value of `opcmon(1|3)` is checked in the script and the processing of the script continues.

OVO checks the exit value from the script and sends an error message if the script returns an exit value other than zero.

Figure 3-11 Interaction Between the Monitor Agent and opcmon(1|3)


Monitor Script
(Provided by customer)
Call script
Evaluate current status of monitored object
Call opcmon()
Return
Return
Monitor Agent
Check interval
Check values received from opcmon
opcmon
Check parameters;
Pass information to Monitor Agent
No
Is name/value pair received?
Yes
Generate error message
Compare received value with threshold; if threshold equalled or crossed, generate the configured message
Check exit value of monitor script; if necessary generate error message

MIB Data Collection

Setting up MIB Data Collection is described in detail in the section "Collecting MIB Data" in the *Managing Your Network with HP OpenView Network Node Manager* manual.

Setting Advanced Options for a Message Source Template

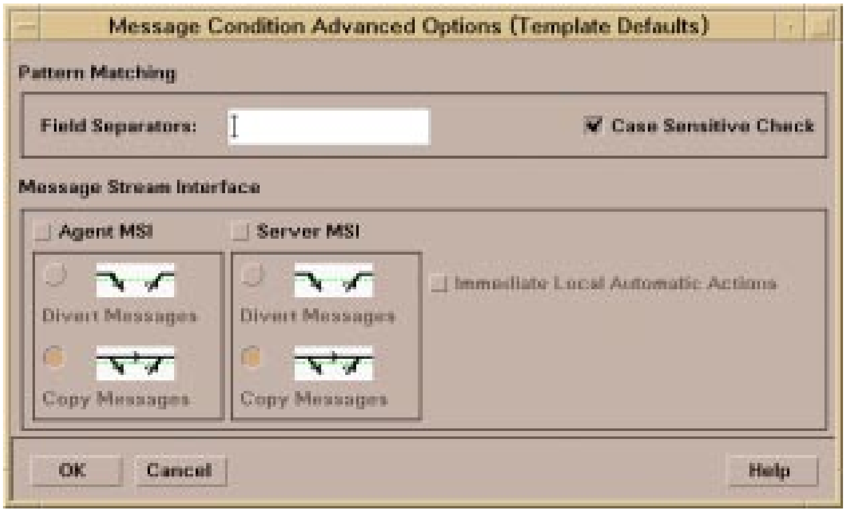
You can specify advanced options for all message sources in the `Advanced Options` window. This includes options for pattern-matching and whether messages are copied or diverted to the Server or Agent Message Stream Interface.

You will use an identical window to set advanced options for individual message conditions, and to set the defaults for an entire template that are applied when a new condition is created. The following description applies to both cases. You can always change the defaults set by the template when specifying individual message conditions.

NOTE

Although you can modify default message conditions for a template at any time, this does not change the conditions associated with templates that you defined previously.

Figure 3-12 Message Condition Advanced Options Window



Message Condition Advanced Options (Template Defaults)

Pattern Matching

Field Separators:

☑ Case Sensitive Check

Message Stream Interface

□ Agent MSI

□ Server MSI

○ Divert Messages

● Copy Messages

○ Divert Messages

● Copy Messages

□ Immediate Local Automatic Actions

For information about the fields in the `Message Condition Advanced Options` window, see the OVO Administrator’s Guide to Online Information.

Output to Agent and Server Message Stream Interface (MSI)

You can choose whether to output messages to a message stream interface so that they can be accessed by external applications, or to disable these features entirely. When disabled, messages matching the message conditions cannot be accessed by processes using the server or agent MSI.

To output messages to either the server or agent MSI, you can choose either to:

❑ Allow a message that matches the condition to be *copied* to the MSI.

  If the message is read by a process using the MSI, then written back to OVO, it will be duplicated.

❑ Allow a message that matches the condition to be *diverted* to the MSI.

  If the message is read by a process using the MSI that does not write it back to the internal message stream, it is lost, which may be the intended behavior. For example, an event correlation mechanism on the management server might compress multiple related messages into a single, more meaningful message.

  Furthermore, if the `Divert Messages` option is enabled, you can also define whether immediate local actions should be allowed. It might be useful to disable local automatic actions if the messages matching the condition are expected to be removed from the message stream at the management server, for example, by a process using the MSI that discards diverted messages. If local automatic actions are disabled, the automatic action is started if the message finally arrives in the database at the OVO management server. This means that an automatic action will not be started if the message is removed from the message stream.

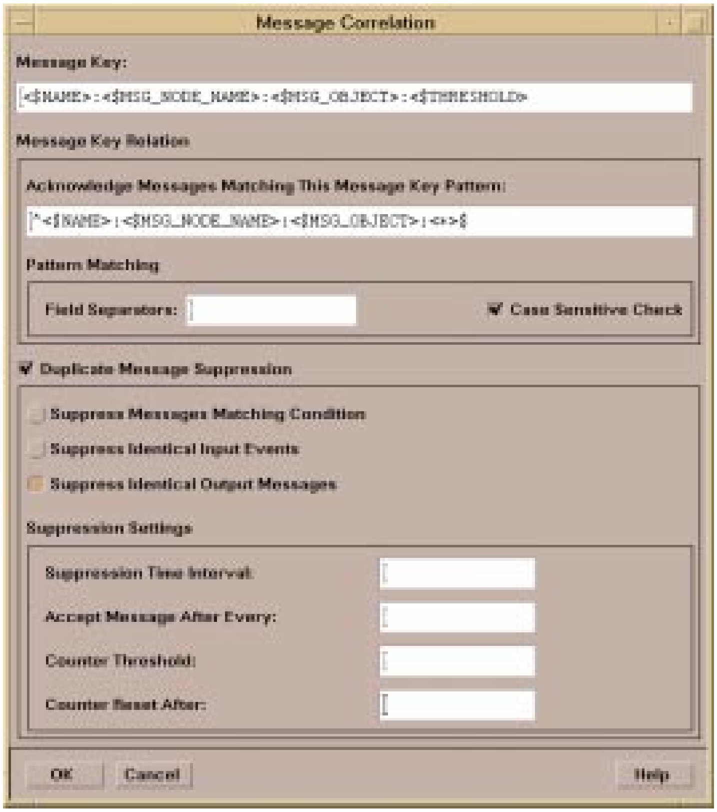
Setting Message Correlation Options for a Message Source Template

You can specify message correlation options for all message sources in the `Message Correlation` window. This includes the setting of message keys and message key relations, and options for the processing of duplicate messages.

You will use an identical window to set message correlation options for individual message conditions, and to set the defaults for an entire template that are applied when a new condition is created. The following description applies to both cases. You can always change the defaults set by the template when specifying individual message conditions.

NOTE Although you can modify default message conditions for a template at any time, this does not change the conditions associated with templates that you defined previously.

Figure 3-13 Message Correlation Window



Message Key: <$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>:<$THRESHOLD>

Message Key Relation

Acknowledge Messages Matching This Message Key Pattern: ^<$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>:<*>$

Pattern Matching

Field Separators:

☑ Case Sensitive Check

☑ Duplicate Message Suppression

○ Suppress Messages Matching Condition

○ Suppress Identical Input Events

○ Suppress Identical Output Messages

Suppression Settings

Suppression Time Interval:

Accept Message After Every:

Counter Threshold:

Counter Reset After:

For information about the fields in the `Message Correlation` window, see the OVO Administrator’s Guide to Online Information.

Suppression of Duplicate Messages

You can set the Duplicate Message Suppression Mode so that OVO suppresses:

- All duplicate messages that match a particular condition.
- All duplicate messages whose input events are identical.
- All duplicate messages whose output messages are identical.

When you activate the suppression of duplicate messages, you can also define either:

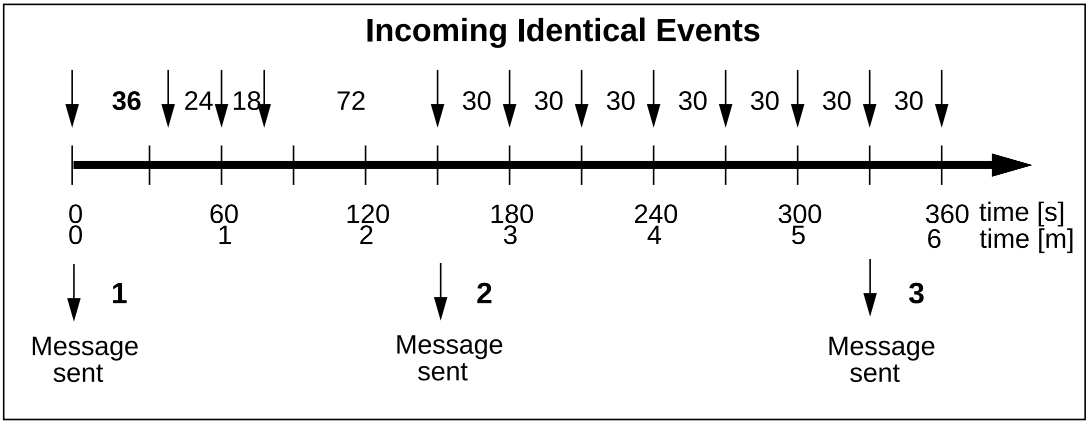
- A time interval over which OVO suppresses duplicate messages, and then retransmits them after this period has elapsed, or
- A threshold for a duplicate message counter; OVO increments the counter until it crosses the threshold and then allows the retransmission of the duplicate message.

Figure 3-14 shows how duplicate messages are suppressed. Assume that the suppress condition has been defined as follows:

❑ Suppress identical messages occurring within:

- A suppression time interval of “1m”
- Accept message after every “3m”

Figure 3-14 Suppressing Duplicate Messages



In Figure 3-14, the first message (1) is sent because it is the first occurrence of the event. The second message (2) is sent because more than 1m (60s) have elapsed between the previous event and the current event. The third message (3) is sent because 3 minutes (180s) have elapsed since the last message was sent and events are still coming.

Setting Options for a Message Source Template

Click on the `[Options]` button in the `Message Source Templates` window to specify options for Local Logging of messages, and for the treatment of unmatched messages.

If a message or trap does not match either a message condition or a suppress condition, the message is flagged as **unmatched**. The `Options` window lets you choose whether to ignore unmatched messages, log them on the local node only, or forward them to the management server. If you forward an unmatched message to the management server, you can choose whether it should be entered directly into a logfile, or that it be displayed in the Message Browser.

Adding Instructions to a Message Source Template

See “Adding Instructions, Annotations, Automatic- and Operator-initiated Actions” on page 136.

Setting Message and Suppress Conditions

This section describes:

- ❏ Setting conditions for incoming messages
- ❏ Setting conditions for incoming SNMP traps

Setting Conditions for Incoming Messages

The `Message and Suppress Conditions` window enables you to specify which messages are to be intercepted from the source and forwarded to the management server, and which messages are to be discarded.

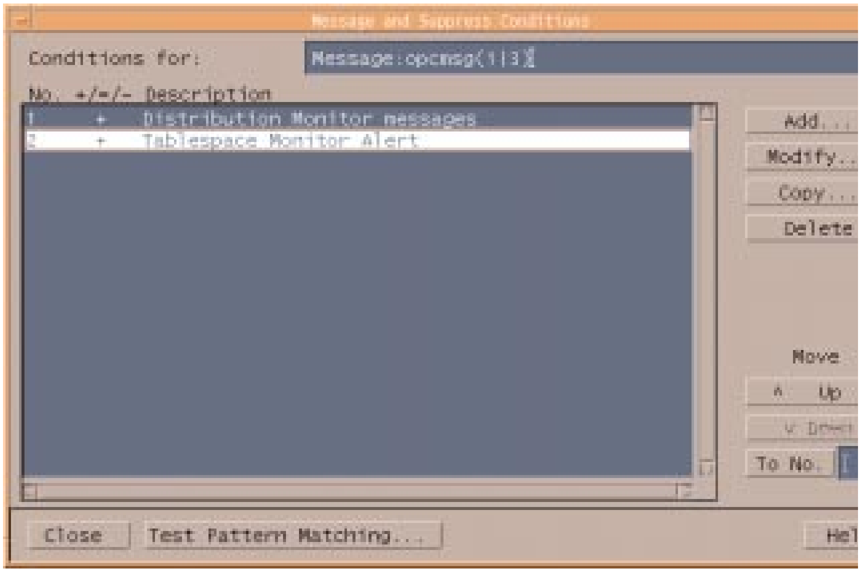
You can specify multiple message and suppress conditions, and freely mix the order and type of conditions. With suppress conditions, you can also choose to suppress either messages that match the match condition, or those that don't match the match condition. Take care when specifying the order of conditions as they are processed in the order of the list. OVO stops processing as soon as a condition matches, so it is recommended to list more specific conditions above the more general conditions. If you don't do this, a general condition may match before the more specific condition is even applied.

If the first condition that matches is a message condition, the message is forwarded to the OVO Message Browser, and any configured actions are invoked. If the message matches a suppress condition, depending on the type of suppression configured, it is discarded. If a message matches neither message conditions nor suppress conditions, it is flagged **unmatched** as previously discussed.

As you can define multiple message and suppress conditions for each template, OVO provides the `Message and Suppress Conditions` window to list the conditions already defined. This window lets you add and delete conditions, or to select specific conditions for modification. You can also change the sequence of conditions in the list.

Figure 3-15 shows the `Message and Suppress Conditions` window. Message conditions are indicated by the plus (+) sign in the second column, suppress matched conditions by the minus (-) sign, and suppress unmatched conditions by the equals (=) sign.

Figure 3-15 Message and Suppress Conditions Window



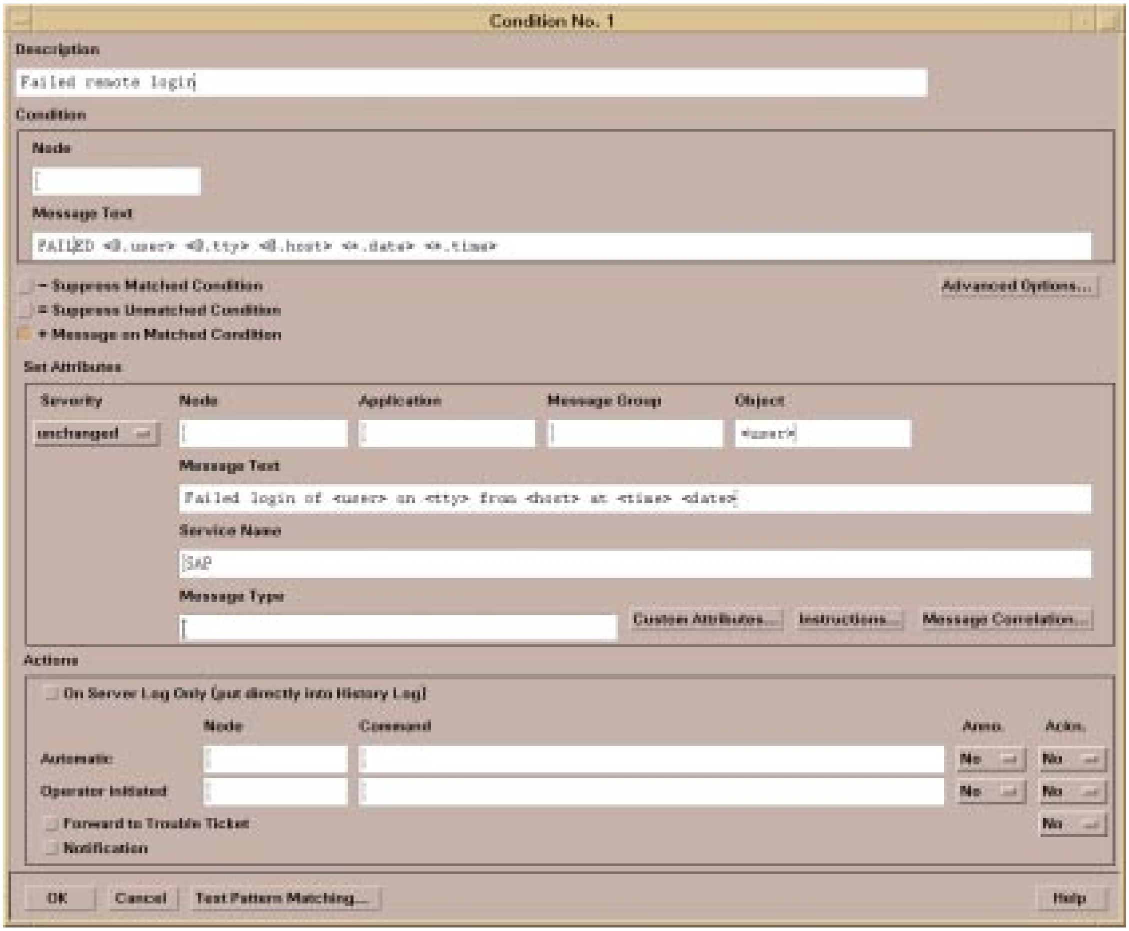
Message and Suppress Conditions

Conditions for: Message:opcmsg(1|3)

No.	+/=/-	Description
1	+	Distribution Monitor messages
2	+	Tablespace Monitor Alert

When you choose to add or modify a condition, the `Condition No.` window shown in Figure 3-16 on page 126 is displayed. This window lets you set the attributes for a single message or suppress condition. The appearance of the window differs, depending on whether you are defining a message condition or suppress condition.

Figure 3-16 Condition No. Window for a Message Condition



Description: Failed remote login

Condition Node:

Condition Message Text: FAILED <@.user> <@.tty> <@.host> <*.date> <*.time>

Severity: unchanged

Object: <user>

Message Text: Failed login of <user> on <tty> from <host> at <time> <date>

Service Name: SAP

Automatic: Anno. No, Ackn. No

Operator Initiated: Anno. No, Ackn. No

Forward to Trouble Ticket: No

The Condition No. window for a suppress condition, contains only a Description field, and fields to specify the different parts of the Match Condition.

For information about the fields in the Condition No. window, see the OVO Administrator’s Guide to Online Information.

Setting Custom Message Attributes for a Message Condition

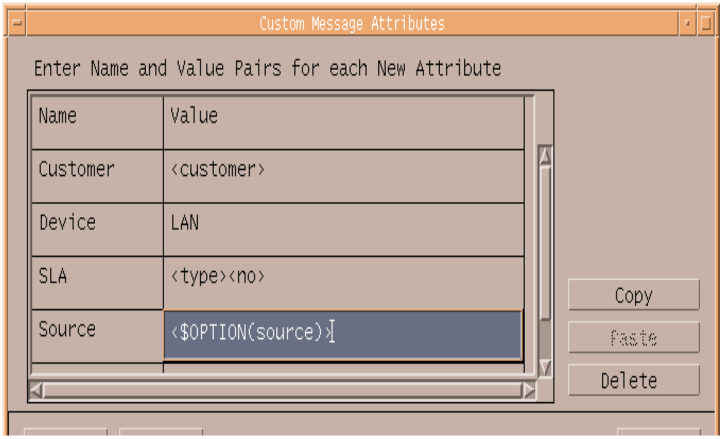
Custom message attributes allow you to add your own attributes to a message. This means that in addition to the default message attributes like severity, date, time, object, or application, you can extend OVO messages with attributes of your choice, for example, the attribute “Customer” or the attribute “SLA” for service level agreements.

Custom message attributes can only be set for message conditions and are only available for logfile, OVO interface, and threshold monitor templates.

Use the `Custom Message Attributes` window to assign attributes of your choice to a message. Figure 3-17 on page 127 shows an example of the `Custom Message Attributes` window. In this example, a message matching this condition would display with four additional columns in the browser windows:

- ❏ Customer
- ❏ Device
- ❏ SLA
- ❏ Source

Figure 3-17 **Custom Message Attributes Window**



Custom Message Attributes

Enter Name and Value Pairs for each New Attribute

Name	Value
Customer	<customer>
Device	LAN
SLA	<type><no>
Source	<$OPTION(source)>

In the `Custom Message Attributes` window, the `Name` fields define the name of the attribute which is then displayed as an additional column in the browser windows. The `Value` fields display the value of the attribute. This value can contain one or more of the following:

❏ Hard-coded text

❏ Variables returned by OVO’s pattern-matching mechanism

❏ Predefined OVO variables

See the *OVO Administrator’s Reference* for a list of predefined variables.

NOTE Custom message attributes are only displayed in the browser and message properties windows of the Java-based operator GUI. The Motif-based operator GUI is not capable of displaying them.

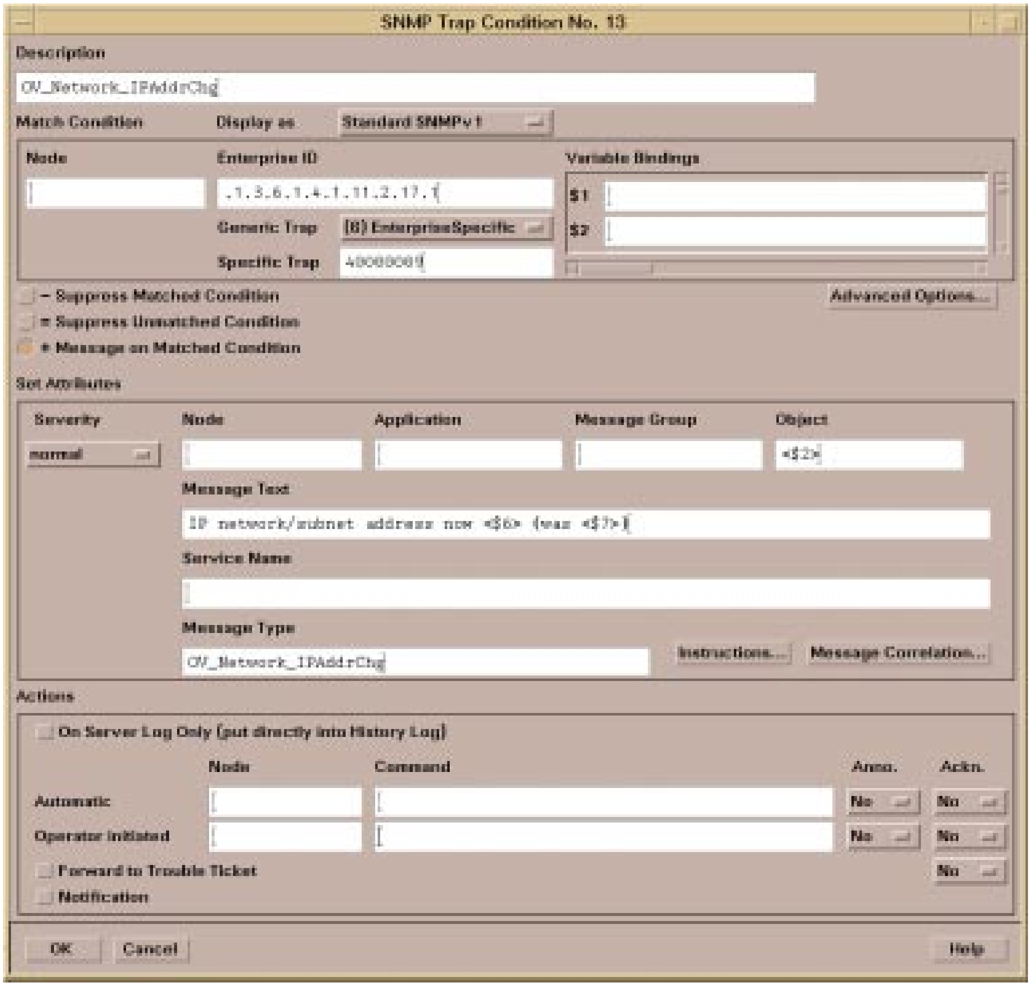
If so configured, custom message attributes are passed to the message stream interface (MSI) on the agent and/or the management server. See the *OVO Developer’s Reference* for more information about available APIs.

Custom message attributes are also passed to trouble ticket systems and/or notification services.

Setting Conditions for Incoming SNMP Traps

The window used to define an SNMP Trap condition, shown in Figure 3-18, differs slightly in the `Match Condition` fields from the one used to define a message condition. Note that when defining a suppress condition, only the `Match Condition` part of the window is displayed.

Figure 3-18 SNMP Trap Condition No. Window



SNMP Trap Condition No. 13

Description: OV_Network_IPAddrChg

Match Condition — Display as: Standard SNMPv1

Enterprise ID: .1.3.6.1.4.1.11.2.17.1

Generic Trap: [6] EnterpriseSpecific

Specific Trap: 40000081

○ − Suppress Matched Condition
○ = Suppress Unmatched Condition
● + Message on Matched Condition

Set Attributes

Severity: normal

Object: <$2>

Message Text: IP network/subnet address now <$6> (was <$7>)

Message Type: OV_Network_IPAddrChg

Actions

□ On Server Log Only (put directly into History Log)

	Node	Command	Anno.	Ackn.
Automatic			No	No
Operator initiated			No	No

□ Forward to Trouble Ticket — No
□ Notification

For information about the fields in the `SNMP Trap Condition No.` window, see the *OVO Administrator's Guide to Online Information*.

Using SNMP Trap Templates Converted from NNM SNMP Trap Configuration Files

In comparison with the NNM SNMP trap configuration files, OVO trap templates provide additional features that can be exploited for the manipulation of SNMP traps.

NNM events are configured by associating the enterprise ID, generic and specific SNMP trap number with an NNM event message, a severity and an event category. The message derived from that NNM event can be logged, displayed in the NNM event browser, or discarded. An automatic action can also be configured for execution on the management server, or any system running the OVO intelligent agent, when the NNM event is received.

OVO provides the same functionality by associating the enterprise ID, generic and specific SNMP trap number with an OVO message, a severity, and a message group. An OVO message can also be logged, displayed in the OVO Message Browser, or discarded.

The OVO user-role concept allows the administrator to set up the operators' working environment so that operators only see messages from systems and message groups under their control. In addition, OVO provides automatic and operator-initiated actions that can be performed on the management server or on any OVO managed node. The status of an action is visible in the OVO message browser (running, failed or successful) and the output of the action can be attached to the message as an annotation. Instructions can also be attached to an OVO message at either the template or condition level.

Converting NNM SNMP Trap Configuration Files to OVO Trap Templates

For NNM integrated solutions, a conversion tool is provided to convert SNMP trap configuration files into an OVO SNMP Trap Template. The conversion tool is part of the OVO product.

In NNM, the SNMP Trap configuration is done using the application **xnmtrap** which maintains the configuration file **trapd.conf**. It contains definitions for the handling of SNMP traps, including how to format their log entries, and what action to take upon their receipt. The event browser **xnmevents** displays the events which are grouped to categories.

After the conversion the same basic functionality is available:

❑ the NNM event definitions (event name, enterprise ID, generic/specific trap number) and the NNM event descriptions are converted

❑ automatic actions previously defined in the NNM SNMP Trap configuration are converted to OVO automatic actions

❑ with OVO, the same six NNM severity levels are supported

❑ OVO supports the same variables for the definition of a message format, so the same message text previously received in the NNM event browser is shown is the OVO message browser

❑ OVO supports the NNM override traps to set the source node, the OV object ID, the severity value or the Event Category (Message Group) value

❑ OVO supports the NNM special traps for setting the status color of an ovw object, display a message in the message browser, pop-up message, ring a bell and highlight source event.

Convert NNM SNMP Trap configuration files to OVO SNMP Trap templates, as follows:

1. Run the command: **ovtrap2opc**

   This creates a new application directory, including the converted trap template file, and uploads the template to the OVO database.

2. Assign the trap template to the management server.
3. Distribute new template to management server.
4. Restart operator sessions.

See also the man page *ovtrap2opc(1M)*.

NOTE The NNM event notification functionality provided through the `Popup Notification` window is covered by means of the Message Browser in OVO. OVO assigns the message group “SNMP” to all NNM events.

Hints and Tips for Event Integration from Message Sources

This section provides a summary of the recommended methods to integrate events from the various OVO message sources.

❏ Use the OVO templates to filter important messages from those that are less acute.

Effective filtering ensures that the Message Browsers of responsible operators do not become cluttered by messages of low severity. Carefully consider whether it is necessary to forward messages of status warning and below to the OVO management server.

❏ Avoid using the severity unknown when defining templates and conditions.

❏ Set the polling interval, used to check the message source, to as long a time period as acceptable.

❏ For OVO performance reasons, define patterns for pattern-matching with care.

- Avoid the use of the asterisk (*) wildcard to match an arbitrary string whenever possible. A comparison against this symbol is the most time-consuming.
- Define the order of message and suppress conditions carefully; this is not only important for the semantics of the template, but also for pattern-matching performance. OVO stops processing when the first message or suppress condition matches the source, for example, an entry in a logfile. You can greatly improve performance by placing an effective suppress condition at the beginning of the list of conditions.

❏ Define new Message Groups for your integration solution whenever possible. Only put new messages into existing message groups when they really belong there.

If you find that there are too many message groups, you can regroup messages by selecting `Actions: Configure Message: Regrouping on Server` from the `Node Bank` menu.

❑ Carefully consider whether to forward all unmatched messages to the OVO management server. The forwarding of unmatched messages has the following associated advantages and disadvantages:

- The analysis of unmatched messages in the design phase can help to improve your definitions of message and suppress conditions so that these messages can be matched in the customer’s environment.
- Increased network traffic and overwhelming of operators can result if the forwarded unmatched messages are found to be unimportant.

It is recommended that you handle unmatched messages as follows:

- In the development phase of your integration, forward all unmatched messages to the management server. You can analyze these messages carefully to improve the templates.
- When your OVO integration is installed at the customer’s site, log all unmatched messages on the originating node and advise the customer to check these at regular intervals to further refine the templates.

❑ It is recommended to enable the output of messages to the agent and server message stream interface (MSI) in the `Message Condition Advanced Options` window for the following reasons:

  - The server MSI can always be globally disabled using the `Configure Management Server` window. By default, the server MSI is disabled.
  - If the server MSI is not enabled, and at a later stage the customer decides to connect an external application to the MSI, they would need to modify all relevant conditions in all templates manually.
  - To prevent an application connected to the server MSI from being swamped with messages, the API can register conditions with the management server, which effectively filters the message stream flowing to the interface.

  Consider the following points when choosing whether to divert or copy messages to the server MSI:

  - If you choose to divert a message, there is a possibility that the application could remove the message from the message stream so that it is never displayed in a Message Browser. If you consider a message to be important, choose to copy it to the MSI.
  - When the copying of messages to the MSI is enabled, not even an event correlation engine can remove a message from the message stream. This result may not be desired if you use an event correlation engine to reduce the number of messages reaching the management server.

❑ Configure templates for your integration so that the OVO operators are always informed about any problems caused by the partner solution. For example, make sure that any known messages of the partner solution trigger OVO messages.

Adding Instructions, Annotations, Automatic- and Operator-initiated Actions

This section describes how to add message annotations and instructions to a message received by the OVO management server, and how to define automatic- and operator-initiated actions for a message.

Adding Instructions for Solving Known Problems

By attaching instructions to a message template or individual message condition, you provide a source of information that can help an OVO operator solve the problems that are causing messages to be generated.

You can either define instructions by entering text in the `Select Instruction Text Type` window, or by specifying an **instruction text interface** listed in the `Instruction Text Interfaces` window. You can supply instructions for all OVO event sources.

You can associate instructions with the following OVO message entities:

- ❑ **Message templates**

  Specify the default instruction text or instruction text interface to be applied to all message conditions created from this point that do not overwrite the template setting.

- ❑ **Individual message conditions**

  Specify the instruction text or instruction interface that is specific to any messages matching the message condition.

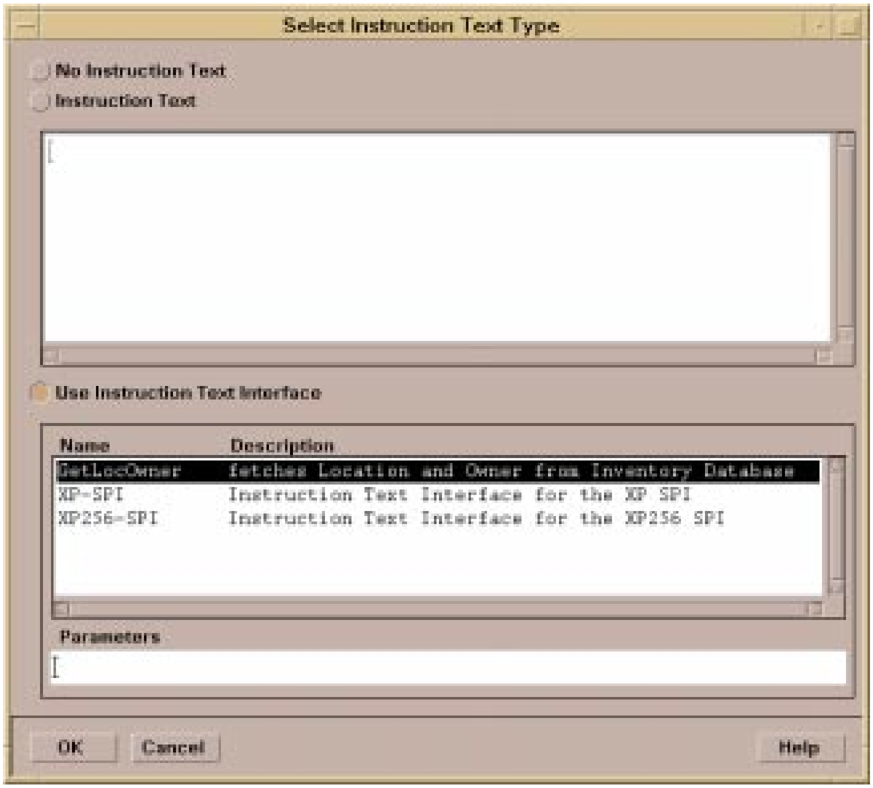
The windows used to define the instruction text options for individual message conditions and to set instruction text defaults for a template are identical. You can always override the defaults when specifying individual message conditions.

NOTE Although you can modify defaults later, this does not affect message conditions defined before the point at which you make the modification.

If you choose to specify an **Instruction Text Interface** (where an external program is called to provide the instruction text) you must already have defined this interface by selecting `Actions: Utilities: Instruction Interfaces...` from the `Node Bank` menu. If you enter instruction text directly into the `Select Instruction Text Type` window, the text applies to a set of messages matching a certain condition, or even to a set of messages generated according to the same template. By using a predefined instruction text interface, you can add instructions that are specific to the individual message.

Figure 3-19 shows the `Select Instruction Text Type` window.

Figure 3-19 Select Instruction Text Type Window



Select Instruction Text Type

○ No Instruction Text
○ Instruction Text
● Use Instruction Text Interface

Name	Description
GetLocOwner	fetches Location and Owner from Inventory Database
XP-SPI	Instruction Text Interface for the XP SPI
XP256-SPI	Instruction Text Interface for the XP256 SPI

Parameters

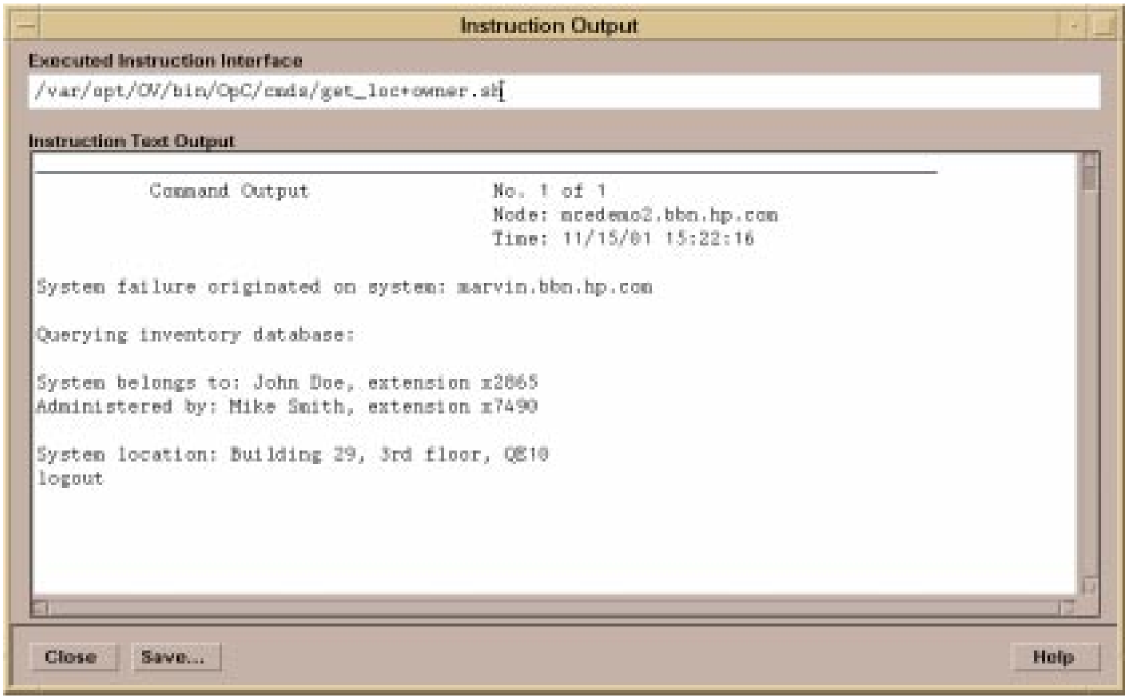
For information about the fields in the `Select Instruction Type` window, see the OVO Administrator's Guide to Online Information.

When providing instruction text, it is useful to use a common, structured text format. Possible sections contained in an instruction text might be:

- ❑ Error number with short error text
- ❑ Problem description
- ❑ Problem solution text
- ❑ Impact
- ❑ Severity

An example layout is shown in Figure 3-20 which contains the output of an external script connected to the OVO Instruction Interface:

Figure 3-20 Example of Instruction Text Output



Executed Instruction Interface

```
/var/opt/OV/bin/OpC/cmds/get_loc+owner.sh
```

Instruction Text Output

```
____________________________________________________________________________
          Command Output                 No. 1 of 1
                                         Node: mcedemo2.bbn.hp.com
                                         Time: 11/15/01 15:22:16

System failure originated on system: marvin.bbn.hp.com

Querying inventory database:

System belongs to: John Doe, extension x2865
Administered by: Mike Smith, extension x7490

System location: Building 29, 3rd floor, QE10
logout
```

Adding Actions and Annotations to a Message

You can choose whether certain actions are performed when a message is received. These actions are either:

- **Operator-initiated actions**

  An operator-initiated action prompts the operator to start certain actions manually to solve the problem generating the message.

- **Automatic actions**

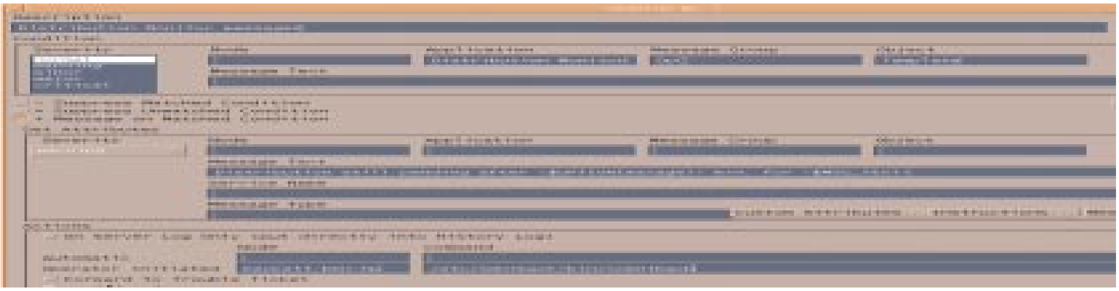
  An automatic action is initiated as soon as the message is received at the management server.

The OVO **annotation** feature can be used to record how and when problems generating messages were resolved. Annotations can serve as a useful source of reference for an operator the next time a similar message is received.

You can choose to add an annotation to a message automatically on the successful completion of an operator-initiated or automatic action. An annotation records the starting time, output, exit value, and finishing time of the action. If an operator-initiated or automatic action fails, an annotation is automatically supplied.

You can define actions and annotations in the lower part of the `Condition No.` window for all message sources, see Figure 3-21.

Figure 3-21 **Actions Part of the Condition No Window**



For information about the fields in this part of the `Condition No.` window, see the OVO Administrator's Guide to Online Information.

External Notification and Trouble-ticket Service

This section describes how to configure the OVO interfaces to external notification services and trouble ticket services. The OVO administrator can define notification services, for example, e-mail, pager, beeper, etc., and can setup a schedule as to when each service is to be used. OVO will forward the message, with a total of 16 parameters, to those services scheduled for the time at which the message arrives at the management server.

For trouble ticket services, the exit from OVO is similar, except that no schedule is defined and the trouble ticket system is always called. The parameters passed to the trouble ticket system are the same as for notification. Please see the *OVO Administrator's Reference* for a list of parameters and example values.

External notification is a unidirectional communication from the OVO management server to the notification services. With trouble ticket systems, bidirectional communication with OVO allows the trouble ticket system to obtain additional information about the message, to add annotations, or to acknowledge the message it received.

Example Script

To show you how to call an external notification service or trouble ticket system, OVO provides the following example script:

```
/opt/OV/bin/OpC/extern_intf/ttns_mail.sh
```

This script sends an email to all operators responsible for the message.

Guidelines for Writing Scripts and Programs

When writing your script or program, follow these guidelines:

❑ **Default Directory**

For scripts and programs calling external interfaces, you can use the following default directory provided by OVO:

```
/opt/OV/bin/OpC/extern_intf
```

CAUTION If you place your scripts and programs in this directory, they will be erased when you de-install OVO.

❑ **Shell Scripts**

If your script is a shell script, the first line must contain a statement such as the following:

```
#!/usr/bin/sh
```

This statement ensures that the shell for which your script is designed is used during execution.

CAUTION If the first line of your shell script does not contain this statement, the execution of your script or program may fail.

❑ **Default Parameters**

OVO sends its own message parameters to the external interface. You may *not* use a command that requires additional parameters. For a list of the parameters provided by OVO, see the *OVO Administrator's Reference.*

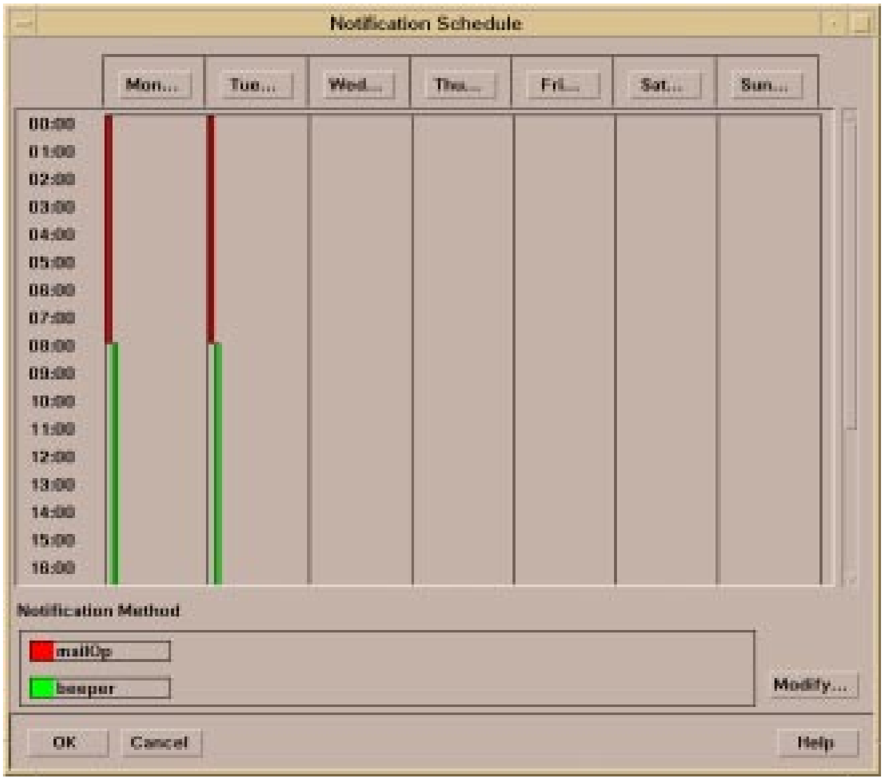
Defining Notification Services

Before specifying notification services, you must define a schedule that specifies which of the available notification services is to be used at a particular time (day, week, hour, and so on). You can define this schedule in the `Notification Schedule` window by selecting `Actions: Utilities->Notification Service...` from the `Node Bank` menu.

If required, you can even specify a different notification service for different days of the week. OVO disables the notification service for each day on which you do not define a schedule.

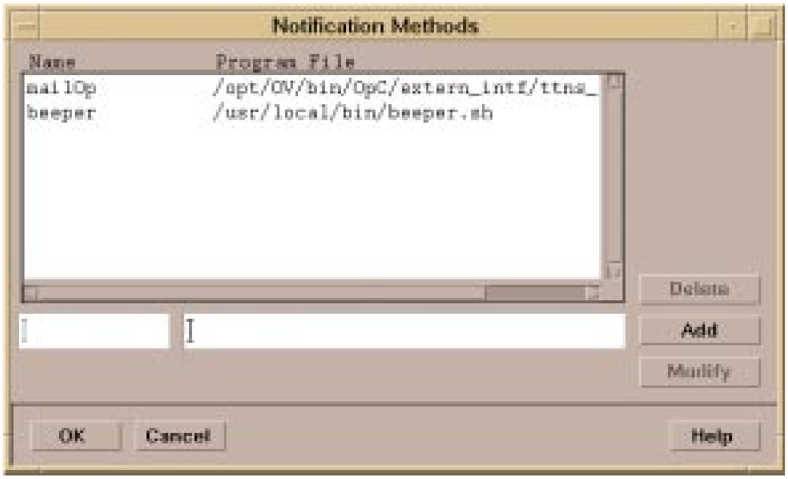
Figure 3-22 shows that two different types of notification service have been configured, and a schedule for Monday and Tuesday has been defined. If the notification schedule was left in this state, external notification would be done only on Mondays and Tuesdays, even if a message had `Notification` enabled.

Figure 3-22 Notification Schedule Window



You can set up the notification services to be scheduled in the `Notification Methods` window. You can open this window by clicking on the `[Modify]` button in the `Notification Schedule` window. To set up a notification service, you need to supply a name for the service, and the pathname of the program or shell script that is called to perform the notification. The program or shell script must be able to accept the parameters that are provided with the call, see the *OVO Administrator's Reference* for a list of parameters..

Figure 3-23 Notification Methods Window

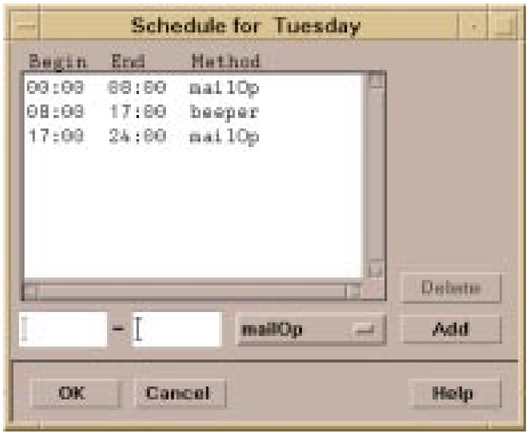


Notification Methods

Name	Program File
mailOp	/opt/OV/bin/OpC/extern_intf/ttns_
beeper	/usr/local/bin/beeper.sh

Figure 3-24 shows the window used to define the schedule for a particular day. You can access this window by clicking on the relevant day button in the upper part of the `Notification Schedule` window. For each day of the week, you can specify which notification services to use at what time.

Figure 3-24 Schedule for Day Window

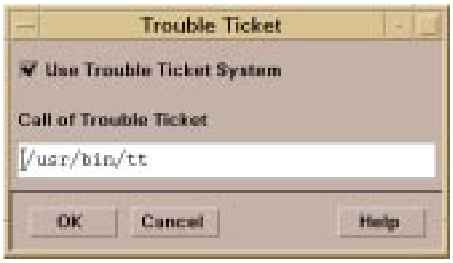


Begin	End	Method
00:00	08:00	mailOp
08:00	17:00	beeper
17:00	24:00	mailOp

Defining Trouble Ticket Services

To define trouble ticket services, select `Actions: Utilities->Trouble Ticket...` from the `Node Bank` menu. This opens the `Trouble Ticket` window shown in Figure 3-25.

Figure 3-25 Trouble Ticket Window



Trouble Ticket

☑ Use Trouble Ticket System

Call of Trouble Ticket

/usr/bin/tt

You can use this window both to enable the use of a trouble ticket system, and to specify which program forwards the message to the trouble ticket system.

Manually Forwarding to Trouble Ticket or Notification Services

In some situations, OVO operators may need to forward messages to a trouble ticket or notification service manually, if these features are not enabled for these messages. To do this, you can configure an application in the operator's Application Desktop that determines which messages are currently selected in the active Browser, and forwards these messages to the trouble ticket system. In this case, the predefined OVO interface to the trouble ticket systems is not used, instead, this mechanism exploits the capability of OVO to pass the IDs of currently selected messages to an application in the Application Desktop as parameters.

OVO operators can select the relevant messages in the Message Browser, double-click on the appropriate application icon in their desktop, and forward messages to an application.

Integrating External Applications into the OVO GUI

This section describes various ways to integrate applications into the graphical user interface of OVO. You can use this section to compare the different methods and then choose the most suitable for your application.

NOTE

For customers who have third-party applications integrated into HP OpenView Network Node Manager, and integrators who provide integration packages specifically for NNM, see “Using NNM-integrated Applications With OVO” on page 161.

GUI Integration Points and Methods

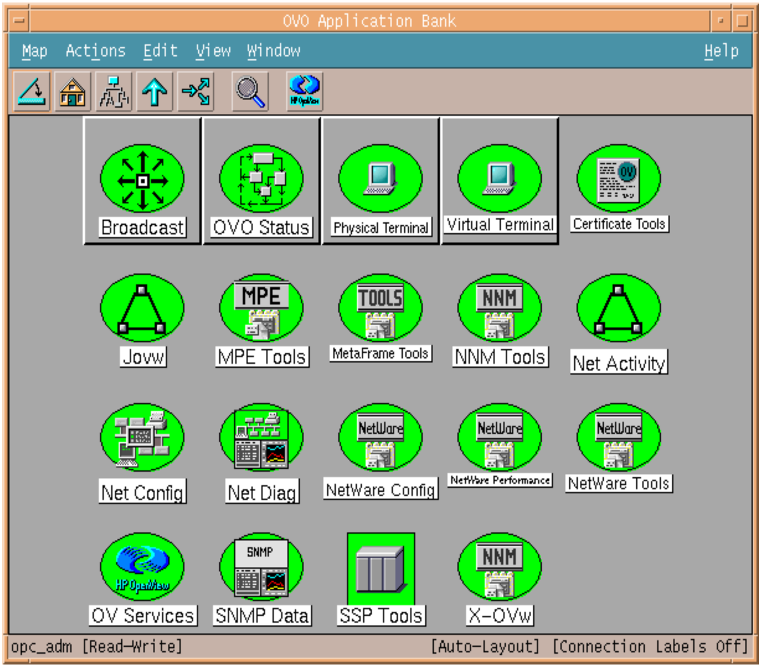
OVO provides the following points from which applications can be launched:

- ❑ The **Application Desktop** is the recommended part of a user’s environment from which to start an application.
- ❑ The **Menu Bar** can be extended to contain additional menus, menu items, or submenus.
- ❑ The **Popup Menus** are similar to the main menus but are displayed by clicking on a submap symbol.
- ❑ The **Toolbar** provides a set of icons that can be used to start frequently used applications.

The OVO Application Desktops can be customized individually for the responsibilities of different OVO users. The nodes on which an application is to be performed can either be predefined by the OVO administrator when setting up the application, or it can be determined by selecting the desired nodes in the Node Bank. The OVO operator can also modify the default application attributes using the `Customized Startup` window.

You can group similar applications together into logical units called **Application Groups**. This prevents the application desktops from becoming too cluttered.

Figure 3-26 OVO Application Bank

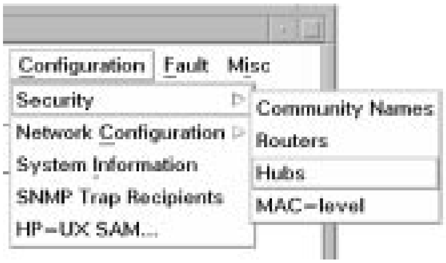


Almost any application can be integrated into the Application Desktop. Applications with an X-window user interface run in an X-window environment, others run in terminal windows.

NOTE To display the NNM GUI through the OVO Java GUI, you must have an X Window System server running on the OVO Java GUI client system.

You can also add your own entries into the OVO menu structure. The highest-level menus describe generic functionality; submenus describe specific functionality.

Figure 3-27 OVO Multi-level Menus



For each new menu item, there must be a corresponding action or sub-menu. You will need to use **application registration files** (ARFs) to associate programs with menu selections.

Menus can be enabled or disabled depending on certain selection rules which specify the type and number of nodes that must be selected in a map window before a menu item becomes active. Inactive menus are automatically “grayed out”.

Toolbars provide a quick, intuitive means of invoking actions. OVO provides a default set of toolbars for invoking actions such as panning or selecting the root map. When an application is added to an OVO environment, it can add icons into existing toolbars, or create window-specific toolbars and icons.

Menu and Toolbar items can be targeted for display on submaps based on the purpose or context of the submap. Submaps can filter which menu and toolbar items are actually visible. When a submap is created it determines its context which is a list of identifiers. For example, a particular submap intended for printer management might have a context of `wantPrinterMenus`, `NoGeneric`. This submap context would limit menu items and toolbar buttons to those that were targeted using the `wantPrinterMenus` context identifier.

There are two main methods by which an application can be integrated into OVO:

- ❑ If the application is already integrated into the HP OpenView windows, then an application registration file (ARF) exists and the application is referred to as an “OpenView windows application”.

  These applications are integrated into OVO either as **OV Applications** or as **OV Services**.

- ❑ OVO provides a powerful mechanism to integrate applications into a user’s Application Desktop. An application integrated this way is referred to as an **OVO Application**. If an application is to be integrated for the first time, this is the preferred method.

The following table shows basic characteristics of the different application integration types:

Table 3-1 **Comparison of Application Integration Types**

	OVO Application	**OpenView Application**	**OpenView Service**
Application accessible to user by:	• Icon in Application Desktop.	• Icon in Application Desktop. • Push button in Toolbar.	• Menu Bar • Push button in Toolbar • Started automatically as daemon when operator logs in.[a]
Application information provided by:	• OVO GUI	• Application Registration File (ARF) • OVO GUI	
Application runs on:	One or more preconfigured or selected node(s) including the management server.		
Runs as user:	• Predefined at configuration. • Can be overwritten by operator using “customized setup”.	• UNIX user of operator who started the OVO GUI. • Can be predefined at configuration. • Any user information can be entered before launching application.	

Table 3-1 Comparison of Application Integration Types (Continued)

	OVO Application	OpenView Application	OpenView Service
Output method:	• Terminal • Window displayed by OVO. • X-application	• Terminal • X-application	

a. The IP Map application is an example of an application integrated as an OpenView service.

Advantages Gained by Integrating OVO Applications

It is recommended to incorporate an application as an OVO Application for the following reasons:

❏ The Application Desktop is the main place for applications that belong to the operators working environment and responsibilities.

❏ Many applications can be easily maintained and accessed by way of icons and application groups in the Application Desktop.

❏ OVO applications can run on remote nodes without the need to specify or transmit passwords over the network, or maintain `.rhost` files, which is not the case for OV applications. The action agents of the target nodes are informed via RPC and switch locally to the configured user.

❏ All information about the application is contained in the central OVO database and can be downloaded using the administrator's GUI. It is not necessary to add an ARF file to an integration package containing the downloaded information, simplifying the process of building integration packages and making it less prone to errors.

❏ All information about an application can be accessed and edited using the OVO GUI. If a new OVO application is configured, the configuration information is immediately checked and the OVO application can be tested by the administrator. In contrast, when configuring OV applications, an ARF file has to be written that may contain syntax- or semantic errors and only takes effect at the next startup of the user interface.

❏ OVO applications allow an OVO user to change the parameters of the application in a controlled way using the customized startup facility.

Integrating OVO Applications

OVO applications are those that are integrated directly into OVO rather than by way of NNM. For these applications you can:

- ❏ Invoke them on the OVO management server and on all OVO managed nodes that are configured as OVO controlled nodes.
- ❏ Invoke them on several nodes in parallel.
- ❏ Change the list of target nodes, by selecting different nodes in the Managed Nodes window.
- ❏ Customize the start-up defaults. You can modify the following attributes:
    - List of target nodes
    - Command parameters
    - User ID to use for execution
    - User's password
- ❏ Allow the retrieval of the list of selected nodes using the reserved variable *$OPC_NODES*.
- ❏ Do not have or make use of OV Application Registration Files (ARFs)

You can modify the configuration of an OVO application in the `Add/Modify OVO Application` window. You can test the application immediately after configuration, and add parameters to the application that may or may not be altered by the operator.

The way an OVO application makes use of windows can be used to further classify OVO applications:

Table 3-2 Windows Used by OVO Applications

Type of OVO Window	Examples	Password Required
No Window	X applications e.g. `npuix`, `xomniback`, or when output is not of interest	No
Output Only Window	`ps(1)`, `lpstat(1)`, `df(1)`	No
Input/Output Window	`sam`, `vi`, `more`, `npui`, `ftp`, `nslookup`	Yes

For all three types of OVO application, you must define a user name. In addition, for applications that use an input/output window, the user's password is required. The user name, used to run the application, can be different for the same application for each of the operators.

The following is a description of the user/password mechanisms used in OVO:

OVO application with `No Window` option (e.g. X application):

You need to define a user name but no password. When starting the application, OVO sends an RPC request to the action agent. The action agent, always running as root, switches to the appropriate user on the managed node, sets the DISPLAY variable, and starts the application.

This is mainly used for X-applications, however, you can also use it for terminal applications if a terminal emulator is available on the managed nodes. In this case, you need to specify this in the application call, for example:

```
/usr/bin/X11/hpterm -e “ls -l /tmp”
```

OVO application with `Output only - Window` option:

You need to define a user name but no password. When starting the application OVO opens a dialog box on the display and sends an RPC request to the action agent. The action agent locally switches to the appropriate user, executes the application, sends the output, using an RPC, back to the management server and displays it in the dialog box.

This is appropriate for the integration of applications which do not require interaction, for example, operating system commands like `ps`, `bdf`, and so on.

OVO application with `Input/Output - Window` option:

You need to define both a user name and a password. When starting the application, OVO opens a terminal locally on the management server and uses an rlogin mechanism, with the defined user and password, to connect to the managed node. The user profile is executed and the application is started. When the application has finished, the connection is closed automatically after the operator has pressed Return or other signal.

You can avoid specifying a password if you define an **.rhosts** entry on the managed node with the specified user name. This method should only be used for terminal applications, and if no terminal emulator is available on the managed node. This is because:

- the need for specifying passwords or rhosts entries is difficult to maintain and, in most cases, is unlikely to meet the security policies of the customer, and
- there is an inherent lack of security in the rlogin mechanism, as it is possible to get shell access by sending a signal at the right time.

Integrating HP OpenView Windows Applications

HP OpenView windows applications are applications already integrated into the HP OpenView platform. These applications are defined through OV **application registration files** (ARFs), and may then be integrated into OVO. Depending on the integration method, these applications are referred to as either **OV Applications** or **OV Services**.

Common characteristics of these applications are as follows:

- ❑ they are already integrated into HP OpenView windows
- ❑ their functions are defined in their Application Registration File (ARF), and not in the OVO `Add/Modify OV Application` window or the OVO `Add/Modify OV Service` window
- ❑ they are the only means of integrating map applications
- ❑ they are always started on the OVO management server, and rely on standard UNIX remote commands (e.g., remsh) to be started on systems other than the server
- ❑ they are restricted to startup on Customized OV Application Call Window and to the functionality defined by the window. (In particular, only objects passed to the OpenView application can be modified.)
- ❑ they allow the retrieval of the list of selected objects by using *`$OVwSelection`*, *`$OVwSelection1`* and so on. The selection name of the object is passed to the OpenView application. Exceptions are the following objects, which may not be passed:
    - OVO Message Groups
    - OVO Applications and Application Groups
    - OVO Operators
    - OVO External Nodes

After configuration, the administrator must normally restart the OVO GUI before invoking an OpenView application. This is true whether the application is integrated as an OV Application or as an OV Service.

OVO does not provide a window for OpenView applications. The application itself is responsible for establishing its own output methods.

HP OpenView Applications

HP OpenView applications integrated as OV Applications present the related application as an icon in the Application Desktop. Double-clicking on the icon starts that action as described in the ARF.

Note that OV Applications do not present the related menu items in the OVO submaps, even if the corresponding ARF contains menu items. Typical OV Applications are MIB Browser for SNMP, or HP OpenView OmniBack II.

Each action defined in the ARF must be integrated as a separate OV Application. This allows you to tailor and customize complex OV applications to the specific requirements of a user. You can specify a different bitmap/symbol for each action.

When the operator starts an application from the Application Desktop, the call is passed to the user's ovw session, which always runs under the UNIX user ID of the operator who started the OVO GUI.

HP OpenView Services

For HP OpenView Services, the integration procedure is very similar to OV Applications but the operator does not see an icon on the Application Desktop. This means that the OV Service is either started automatically when the operator logs in to OVO, or it is started on demand from a menu or toolbar.

In the administrator's Application Desktop, OV Services are represented by icons, however, unlike OV applications, they cannot be invoked by double-clicking. If required, you can invoke and access services from the menu bar. Although services cannot be invoked by clicking on an icon, they do appear as icons in the administrator's Application Desktop, and may be distributed to operators by copying as usual.

An OV Service allows you to integrate OpenView applications which start daemons and/or should be integrated in the menu bar. This means that OVO operators have no graphical representation of their assigned services, but they might see or access additional submaps if the OV Service corresponds to an OpenView map application.

If menus are defined in the OV Application Registration File (ARF), these menu items will be integrated in the menubar of the OVO submaps for the operator.

Integrating Applications into the Application Desktop

This section describes how to define the following using the OVO GUIs:

- ❏ OVO application(s)
- ❏ OV application(s)
- ❏ OV service(s)

For each OV Application or OV Service that you want to define, OVO needs the OV ARF in addition to the specifications done in the GUI. The OV ARF should reside below one of the following two directories:

`/etc/opt/OV/share/registration/<`*`lang`*`>`

`/etc/opt/OV/share/conf/OpC/mgmt_sv/appl/registration/<`*`lang`*`>`

Where <*lang*> is the language variable for the installation; this is `C` for an English language installation.

NOTE When planning the packaging of an integration, it is important to note that GUI specifications can be downloaded but ARFs cannot.

With an English language version of OVO, all registration files in the following directory are not uploaded by `opccfgupld(1M)`:

`.../APPLICATION/OVREGFILES/japanese`

The `opccfgupld(1M)` utility saves the GUI specifications in the database and copies the ARFs to the directory:

`/etc/opt/OV/share/conf/OpC/mgmt_sv/appl/registration/<lang>`

With a Japanese language version of OVO, the registration files in both of the following directories are uploaded:

`.../APPLICATION/OVREGFILES/japanese`

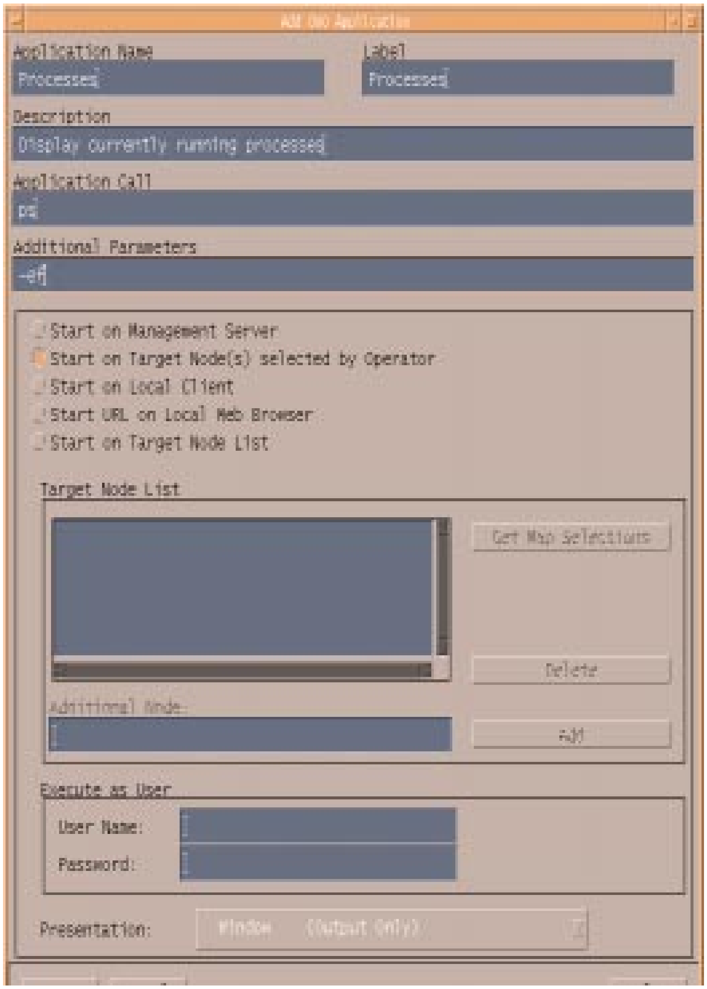
`.../APPLICATION/OVREGFILES/C`

The registration files in the `C` directory are used if the GUI is started with the *`LANG`* variable set to `C`, and conversely, the registration files in the `japanese` subdirectory are used when the GUI is started with the *`LANG`* variable set to `japanese`.

Adding OVO Applications

Use the `Add OVO Application` window shown in Figure 3-28 to add an OVO application. To access this window, select `Actions: Application->Add OVO Application...` from the menu bar of the `Application Bank` window.

Figure 3-28 Add OVO Application Window



Application Name: Processes

Label: Processes

Description: Display currently running processes

Application Call: ps

Additional Parameters: -ef

○ Start on Management Server
● Start on Target Node(s) selected by Operator
○ Start on Local Client
○ Start URL on Local Web Browser
○ Start on Target Node List

Target Node List

Additional Node:

Execute as User

User Name:

Password:

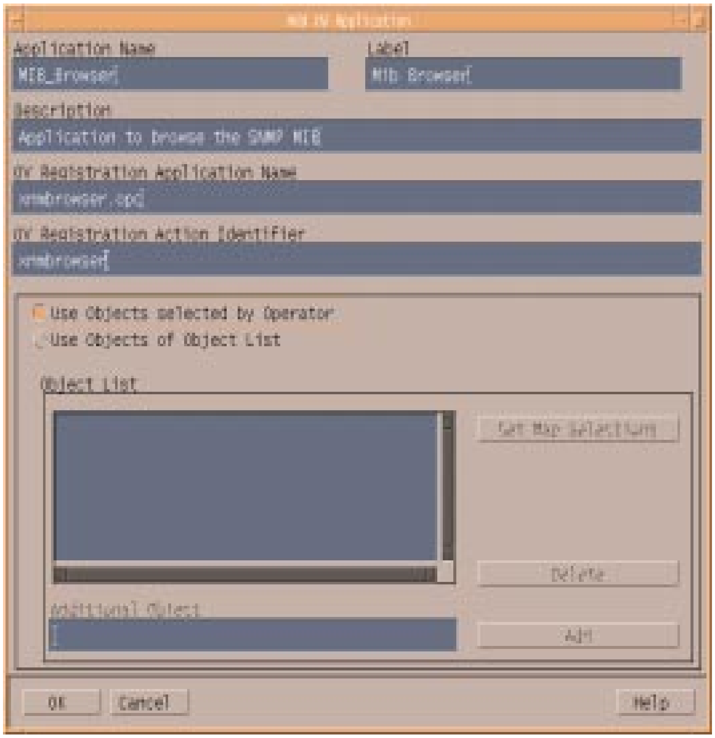
Presentation: Window (Output Only)

For information about the fields in the `Add OVO Application` window, see the OVO Administrator's Guide to Online Information.

Adding OpenView Applications

You can add an OV application in the `Add OV Application` window shown in Figure 3-29. To access this window, select `Actions: Application->Add OV Application...` from the `Application Bank` window.

Figure 3-29 **Add OV Application Window**



Add OV Application

Application Name: MIB_Browser

Label: Mib Browser

Description: Application to browse the SNMP MIB

OV Registration Application Name: xnmbrowser.opc

OV Registration Action Identifier: xnmbrowser

● Use Objects selected by Operator
○ Use Objects of Object List

Object List

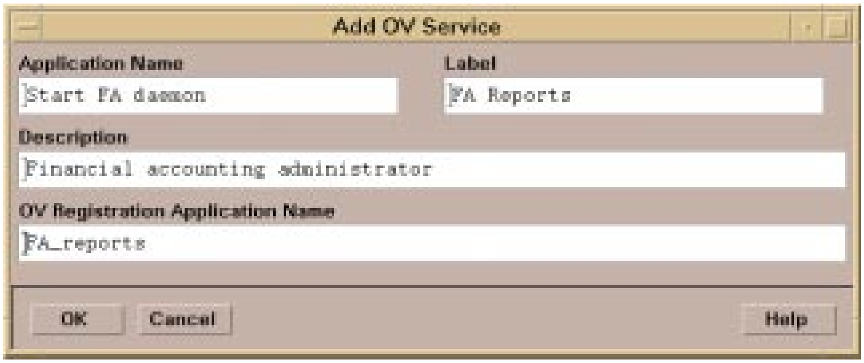
Additional Object

For information about the fields in the `Add OV Application` window, see the OVO Administrator's Guide to Online Information.

Adding an OpenView Service

You can add an OV service in the `Add OV Service` window shown in Figure 3-30. To access this window, select `Actions: Application->Add OV Service...` from the `Application Bank` window.

Figure 3-30 Add OV Service Window



Application Name: Start FA daemon
Label: FA Reports
Description: Financial accounting administrator
OV Registration Application Name: FA_reports

For information about the fields in the `Add OV Service` window, see the *OVO Administrator's Guide to Online Information*.

The application specifications entered into OVO with the GUI can be downloaded into the `APPLICATIONS` directory. You can choose either to download single applications or services, or a whole application group. The syntax for the application description files and some examples are provided in "Configuration Files for Applications" on page 323. This syntax is created automatically when downloading. The download utility does not overwrite files in the `.../APPLICATIONS` directory, but generates a file `appl0`. If this already exists, then the file names `appl1`, `appl2` and so on are used.

Remember to add the ARFs to the subtree `.../APPLICATIONS/OVREGFILES`, before you build your application integration fileset.

Using NNM-integrated Applications With OVO

This section describes how applications that are already integrated into NNM can be accessed by way of OVO, or can be integrated in different ways. Applications that are part of NNM are referred to as "**NNM Applications**" whereas applications that are manually added are referred to as "**Additional Applications**".

As opposed to NNM, which does not support user roles, applications integrated into the OVO GUI can be assigned individually by the administrator to dedicated operators. Each operator has his/her own customized working environment.

NNM Applications in OVO

All standard NNM applications that present menu items in NNM are automatically available to the OVO administrator in two ways, either:

❏ Contained in logical groups as `OV Application` icons.When these OV Applications are assigned to an operator, they are displayed as icons in the customized Application Desktop of that operator.

❏ Contained in the Application Bank under the `OV Services` application group as an **OV Service**. When these OV Services are assigned to an operator, they do not appear in the Application Desktop, but as menu items available from any of the operator's IP-Maps.

  The `OV Services` application group also contains daemon applications that do not present menu items in NNM, for example, IP-Map.

There are also three NNM-specific applications contained in the `X-OVw` application group in both the OVO administrator and operator GUIs. This default application group contains the following applications:

❏ `Start OVw`. This starts an ovw session on the OVO management server or, if available, on a remote NNM system.

❏ `Highlight Selected Node`. This application maps the selected node to an NNM system and highlights the node in an ovw session of that NNM system.

❏ `Highlight Message Node`. This maps the node related to a selected message to an NNM system, and highlights the node in an ovw session of that NNM system.

In addition to the standard user (**opc_op**), OVO provides a predefined network user (**netop**) and a predefined network administrator (**itop**):

❑ The netop environment allows the network user to perform all actions that a typical network (NNM) operator would have available, without the OVO configuration capabilities.

❑ The itop environment allows the network administrator to perform all actions that a typical network (NNM) administrator would have available with all configuration capabilities together with the remote collection station applications and some standard system management functions.

An OVO user can be given access to an NNM application when the administrator assigns an OV Application or OV Service to him/her by copying the application from the administrator's Application Bank to the operator's Application Desktop.

Always consider the following points when assigning applications to an operator:

❑ Always place applications in the operator's Application Desktop.

Whenever possible, it is recommended to assign OV Applications in preference to OV Services.

❑ In addition, you can add applications to the menus in the operator's environment by assigning OV Services.

❑ Never integrate applications into menus when they are not already integrated as icons in the operator's Application Desktop.

In general, an OVO user, excepting **netop** and **itop**, does not have the IP-Map application assigned by default and therefore does not automatically have access to the IP-Map submaps. An operator can be given access to the IP submaps when the administrator assigns the OV Service `IP-Map` to his/her Application Desktop. The operator will not see the OV Service IP-Map in the Application Desktop but will see the IP submaps themselves.

Structure of NNM Applications in OVO

In the Application Bank similar applications are grouped together into **Application Groups**. NNM applications are typically grouped into the following application groups: Net Activity, Net Config, SNMP Data, NNM Tools, OV Services, and X-OVw. Applications that use the symbol class `SW_Utils` are always started on the management server. For a list of the various application groups and applications, see the OVO Administrator's Reference.

In the OVO Java GUI, ovw applications appear in all application menus; in the Motif GUI, they appear in the Application Bank/Desktop.

Integrating Additional NNM Applications into OVO

Applications integrated into NNM, but which are not part of NNM, are not automatically available in the Application Bank of the OVO administrator or in the Application Desktop of an OVO user.

To integrate this type of application, do the following:

- ❏ If the application does any trap configuration, for example, registering for incoming traps, this configuration can be used in OVO by running a script.
- ❏ If the application has its own daemon process that creates new submaps (like IP-Map) then add it manually to the Application Bank as an OV Service OVO administrator. This OV Service can then be assigned to the desired operator.
- ❏ If the application presents menu items in NNM the OVO administrator can create them manually as OV Applications. When the application is assigned to an operator, it appears as an icon in the Application Desktop of that operator.
- ❏ If an application presents menu items in NNM the administrator can also create a new OV Service application. When the application is assigned to an operator, it appears as a menu item.

An integrator can incorporate an application as appropriate and then provide an integration package that can be easily uploaded into an existing OVO installation, using `opccfgupld(1M)` and `opccfgdwn(1M)`. The creation of an integration package is described in Chapter 8, Creating and Distributing an Integration Package,.

4 Using the OVO Application Programming Interfaces

In This Chapter

This section describes the concept and facilities of the **application programming interfaces** (APIs) provided with the OVO Developer’s Toolkit.

Overview of the OVO APIs

OVO provides several APIs to OVO functions which enable a knowledgeable user to enhance the operations and problem management of OVO. This chapter provides an overview of all available APIs—they are described in more detail in the OVO Developer's Reference. The APIs can be further subdivided into the:

❏ *OVO Operator APIs*

The OVO Operator APIs provide a set of functions that allow you to operate on OVO messages, message events, and application responses, for example to own or disown a message.

This group of APIs also includes the Interface API. The Interface API provides a set of functions that allow access to OVO by opening one of the following interfaces:

- Server Message Stream Interface
- Agent Message Stream Interface
- Legacy Link Interface
- Application Response Interface
- Message Event Interface

The OVO Interfaces use the Interface API functions to register with OVO to receive data. When the requested data is available, OVO sends it to the instance of the interface which made the request.

❏ *OVO Configuration APIs*

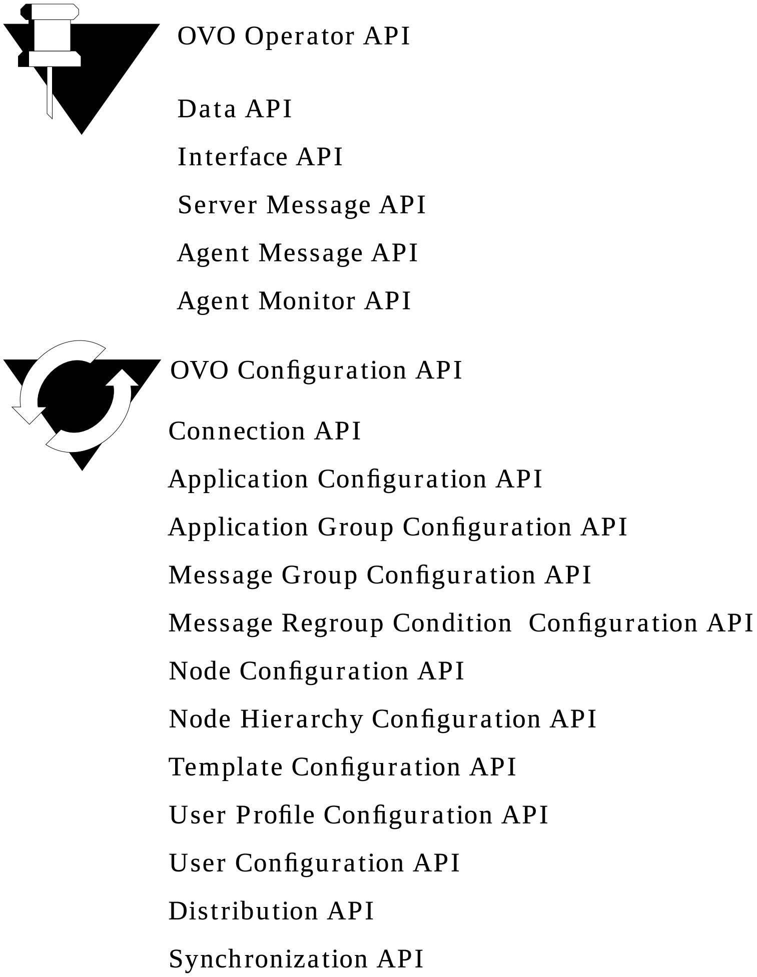
The OVO Configuration APIs provide a set of functions to configure OVO data directly in the database. The functions allow you, for example to configure new OVO templates or managed nodes, or to modify existing applications or users. In addition, functions are available to control access to OVO data, and to distribute your configuration changes to the managed nodes.

Figure 4-1 on page 169 gives an overview of the OVO User APIs, and the groups of functions included in each API.

The OVO Developer’s Toolkit also provides interfaces to the HP OpenView NNM platform functions: OpenView Windows and the OpenView SNMP System. These integration facilities are described in Chapter 7, “Integration Facilities of the HP OpenView NNM Core Platform.”.

For a list of man pages available with the OVO Developer’s Toolkit, see Appendix C, “About OVO Man Pages.”.

Figure 4-1 **Overview of the OVO User APIs**



OVO Operator API

- Data API
- Interface API
- Server Message API
- Agent Message API
- Agent Monitor API

OVO Configuration API

- Connection API
- Application Configuration API
- Application Group Configuration API
- Message Group Configuration API
- Message Regroup Condition Configuration API
- Node Configuration API
- Node Hierarchy Configuration API
- Template Configuration API
- User Profile Configuration API
- User Configuration API
- Distribution API
- Synchronization API

Figure 4-2 illustrates the information exchange between OVO and the APIs.

Figure 4-2 Overview of OVO APIs on the Management Server and Managed Nodes

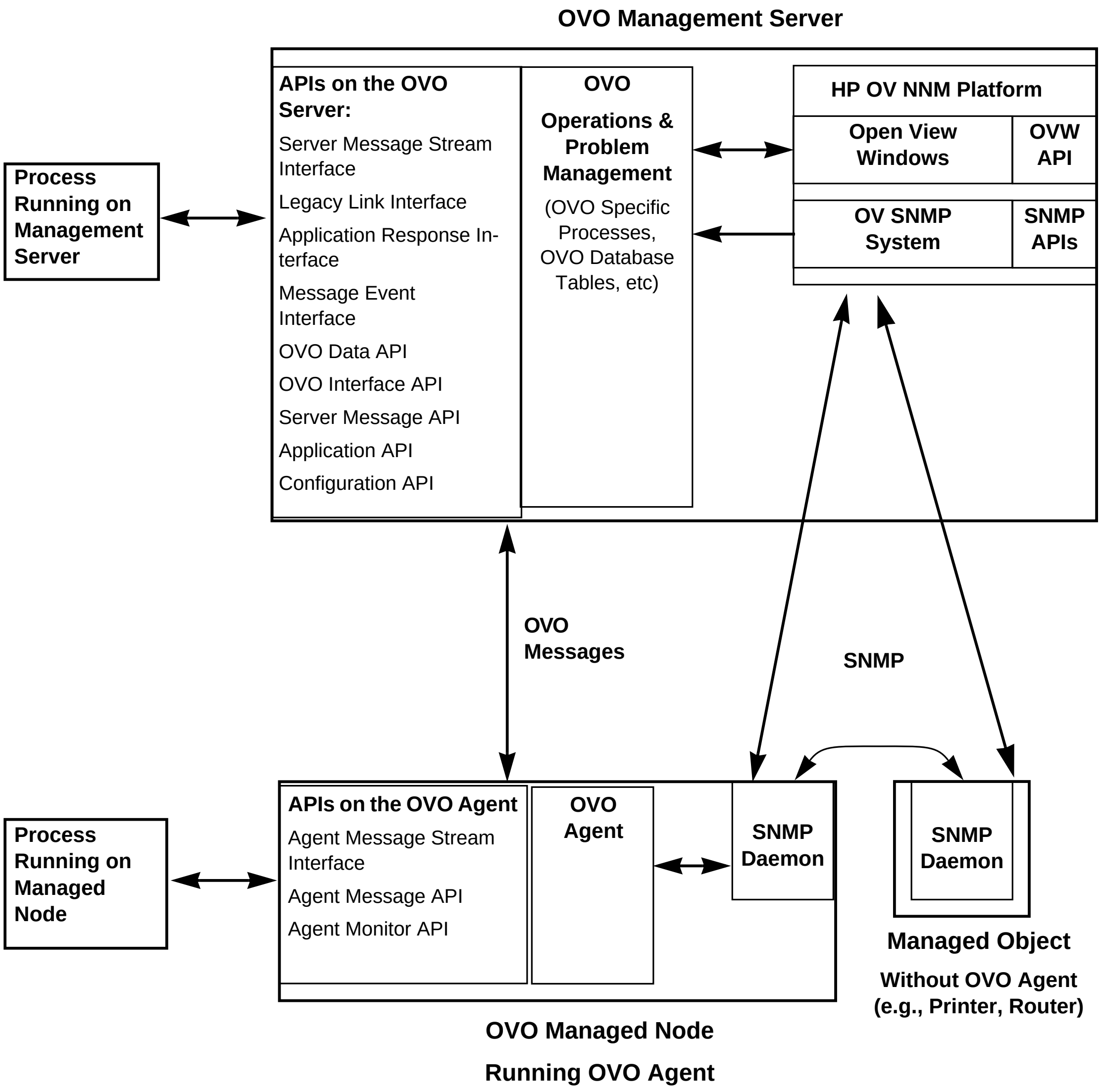


Table 4-1 on page 171 gives an overview of the location of the OVO APIs.

Table 4-1 Location of the OVO APIs

API	Location
OVO Operator APIs	
OVO Data API	Management Server and Managed Node
OVO Interface API	Management Server and Managed Node
Server Message API	Management Server
Agent Message API	Managed Node
Agent Monitor API	Managed Node
OVO Interfaces	
Server Message Stream Interface API	Management Server
Agent Message Stream Interface API	Managed Node
Legacy Link Interface API	Management Server
Application Response Interface API	Management Server
Message Event Interface API	Management Server
OVO Configuration APIs	
Connection API	Management Server
Application Configuration API	Management Server
Application Group Configuration API	Management Server
Message Group Configuration API	Management Server
Message Regroup Condition Configuration API	Management Server
Node Configuration API	Management Server
Node Hierarchy Configuration API	Management Server
Template Configuration API	Management Server
User Profile Configuration API	Management Server

Table 4-1 Location of the OVO APIs (Continued)

API	Location
User Configuration API	Management Server
Distribution API	Management Server
Synchronization API	Management Server

The OVO Interfaces

The following interfaces to OVO can be accessed by way of the Interface API:

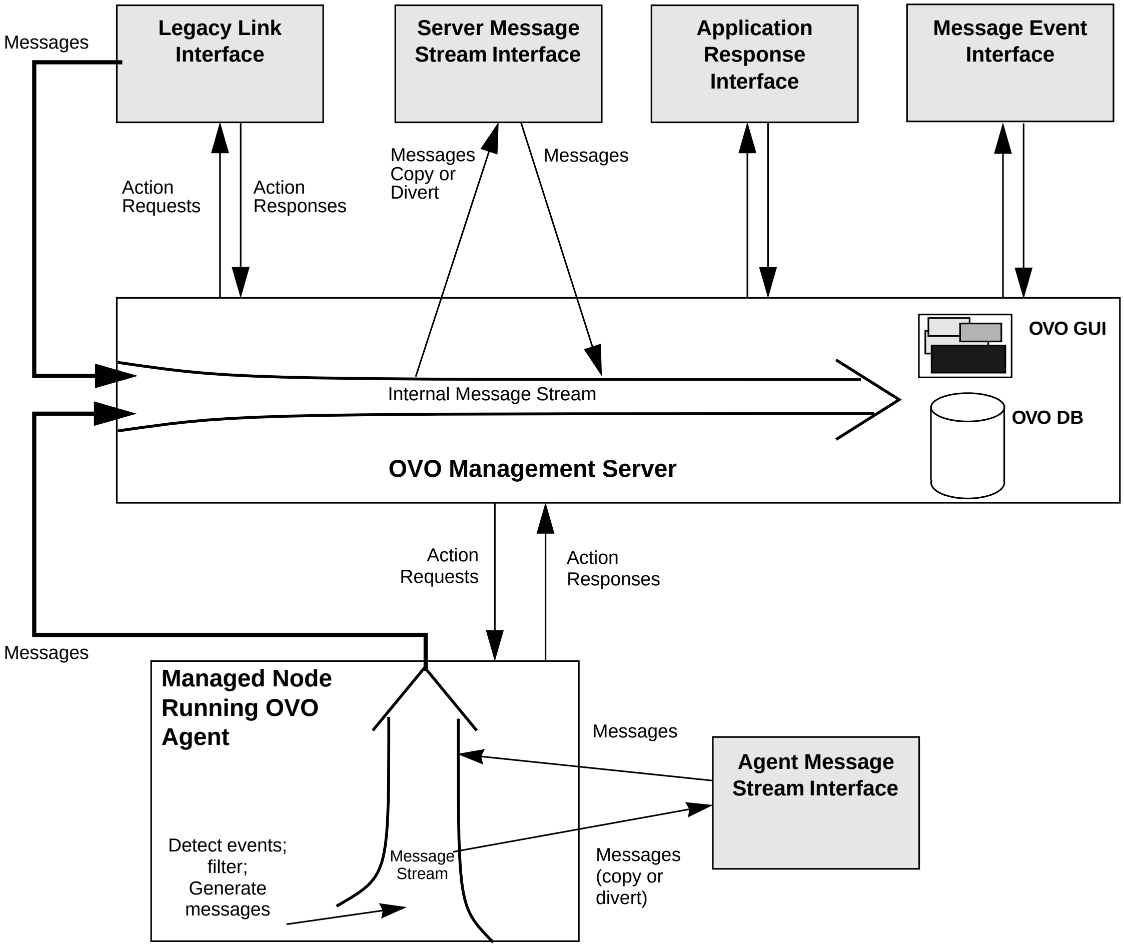
Interface	Description
Server Message Stream Interface	Enables OVO messages to be output from the server message stream to an external application.
Agent Message Stream Interface	Enables OVO messages to be output from the agent message stream to an external application.
Legacy Link Interface	Provides a link between OVO and managed nodes for which OVO agents are not currently available.
Application Response Interface	Allows external applications to receive application responses from OVO applications that have been started.
Message Event Interface	Allows external applications to receive message events to display OVO messages, for example in a GUI.

NOTE

The same API, the Interface API, is used for all interfaces. The interface type must be specified as a parameter of the API function call to distinguish between the various interfaces.

Figure 4-3 on page 174 shows how the OVO Interfaces interact with the internal message stream of the OVO management server and managed nodes. Messages are received into the internal message stream, processed, then stored in the message database and displayed in the Message Browsers of the GUI. The OVO Interfaces on the management server are ordered from left to right to show that they tap the internal message stream at different points, and that they can interact in different ways.

Figure 4-3 **Interaction of OVO Service APIs with the Server/Agent Message Flows**



The Agent Message Stream Interface, the Agent Message API, and the Agent Monitor APIs are available on the managed nodes, and all other APIs are currently available on the management server only. See Table 4-1 on page 171 for more information.

Overview of the Server Message-Stream Interface

The OVO management server's message-stream interface provides access to the internal message stream of the OVO management server. Processes may connect to the message-stream interface on OVO's management server either in parallel or in series. Multiple connections to the MSI are organized by means of order numbers. Assigning an order number to each MSI connection allows OVO's message manager to determine at which point in the serial chain those external applications which are connected to the MSI receive the messages passing through. For example, an external application connected to the MSI can choose either to receive messages in parallel with other connections or after the messages have been processed by other applications in the serial chain, such as an event-correlation (EC) engine, and passed back to the MSI.

Ideally, all messages should pass through the event-correlation engine first: not only does this reduces the overall number of messages in the MSI after the EC instance, it also means that subsequent MSI instances receive more useful messages. A Trouble Ticket service should be connected at the end of the serial chain so that it gets only those messages which are not suppressed by other MSI connections.

The routine to open an instance of the OVO interface first requires a parameter that specifies the type of interface to be accessed (server MSI, legacy-link interface, etc.) followed by a user-defined instance name and an order number. This information is stored in the configuration file `/etc/opt/OV/share/conf/OpC/mgmt_sv/msiconf`, which is described in more detail in “OVO's Serial MSI Configuration File” on page 177. Entries in the MSI configuration file use the following format:

```
<MSI instance name> <order no.>
<MSI instance name> <order no.>
```

The term “user-defined” implies a potential naming conflict if different applications specify the same instance string. This problem is similar to the possible file naming conflicts known for operating systems. As the instance name is related to the file names used for the interface message queues, and OVO supports short file name OS versions, the length of the string is restricted to 12 ASCII characters (two characters are required for internal handling). The interface ID returned by the open routine is then used in subsequent calls to the other API functions.

The routines return an error/ok code, with errors being logged in the file:

`/var/opt/OV/log/OpC/mgmt_sv/opcerror`

Access to the Server Message-stream Interface

If an application registers at the Message Stream Interface, OVO checks the setting of the `Enable Message Stream Interface` checkbox in the `Server Configuration` window. If the interface is enabled, the message manager passes the message down to the next check. If the message itself is also allowed for output, it is written to the interface queues of OPCSVIF_EXTMSGPROC_READ and OPCSVIF_EXTMSGPROC_READWRITE. If the option button `Copy Message` is selected, a copy of the message is immediately passed back to the internal flow of the management server, see Figure 4-7 on page 188.

To allow the encapsulation of the `opcif_write()` routine in a command that can be used in shell scripts, it is possible to open the write queue of the read/write interface in a separate process using another interface type.

OVO's Serial MSI Configuration File

The name of each MSI instance and the order number assigned to it are stored in the configuration file `/etc/opt/OV/share/conf/OpC/mgmt_sv/msiconf`. Both parallel and serial connections from the MSI are possible at any point in the serial chain. The start number 0 (zero) is allocated by default to the event-correlation manager, `opcecm`. Ascending or descending order numbers indicate a serial connection: order numbers of equal value indicate a parallel connection.

The message manager reads the configuration file whenever an MSI instance opens a connection to the server MSI and then distributes the messages according the values it finds in the file for the applications or instances registered. Any MSI instance that is not listed in the configuration file is assigned the order number 0, which means it receives messages in parallel with the event-correlation agent process, assuming the event-correlation agent is running. OVO allows you to assign order numbers between -127 and 127: gaps are also allowed in the sequence. You can change the order of individual instances while the message manager is running: the message manager reads the configuration file each time an instance connects (or reconnects) and uses the new value it finds. However, the message manager continues without interruption if an MSI instance disconnects itself or dies suddenly.

NOTE

The message manager ignores entries in the file `/etc/opt/OV/share/conf/OpC/mgmt_sv/msiconf` that do not conform to the format described and writes them instead to the `opcerror` log file.

Modifying Message IDs

Generally speaking, if an MSI instance modifies a message, a new message-ID is generated when the instance sends the message back to the MSI. However, there are exceptions: *no* new message-ID is generated if the message has either been *diverted* to the MSI (that is, no other copy of the message exists either inside or outside the MSI) or an MSI instance sends an unmodified message back to the MSI. If two parallel MSI instances receive the same message and both subsequently send it unchanged back to the MSI, multiple versions of the same message with the same message ID exist in the MSI. As a consequence, any MSI instance connected further down the serial chain receives multiple copies of the message with the same message ID.

NOTE It is recommended you use the serial MSI feature if you or your applications require read/write access to messages.

You can control the generation of message IDs by setting the variable, `OPC_MSI_CREATE_NEW_MSGID` in the `opc[sv]info` file to the following values:

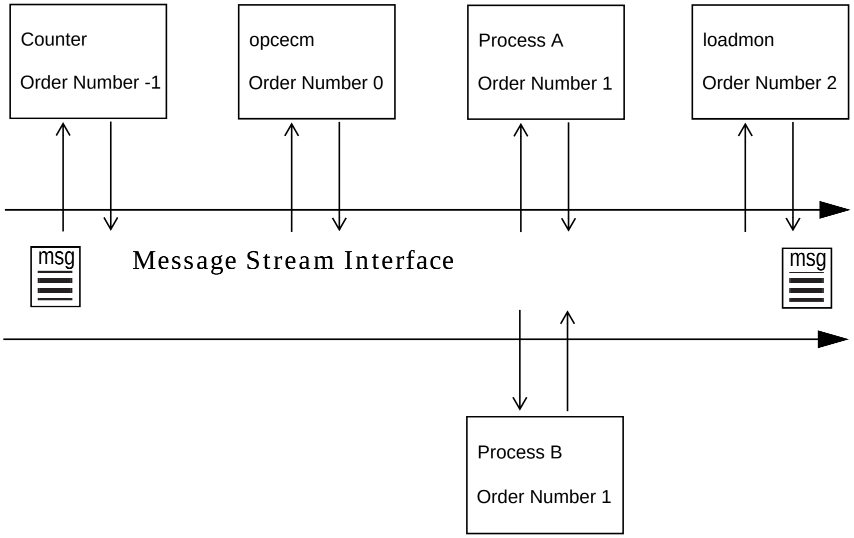
1	enable OVO A.04.00 behavior
2	enable OVO A.05.00 and higher behavior (Default)
3	custom: no automatic creation of *new* message IDs You can suppress the generation of new message IDs generally (for example, parallel messages, copied messages, and so on) using `opcdata_generate_id()` or `opcdata_set_str()`. For more information on these calls, see the *OVO Developer's Reference*.

Serial MSI Configuration: Example Scenario

Figure 4-4 on page 179 illustrates how messages are distributed in a scenario where five processes are connected to the MSI: counter, opcecm, procA, procB and loadmon (load monitor). Incoming messages go first to counter, then in series to opcecm, then in parallel to both procA and procB. Finally the message is passed in series to the instance loadmon. The MSIs that are connected need to be assigned a number for the configuration to work. The following configuration file represents the scenario described above:

```
counter     -1

opcecm       0

procA        1

procB        1

loadmon      2
```

Figure 4-4 Message Distribution in OVO's Serial MSI



Overview of the Agent Message Stream Interface

The **Agent Message Stream Interface** allows you to tap the message flow of an OVO managed node to enable additional message processing by external applications before a message is sent to the management server. This can help to reduce the amount of network traffic considerably. A typical external application might be an event correlation engine, for example ECS.

Overview of the Legacy Link Interface

The **Legacy Link Interface** enables you to manage network nodes for which OVO intelligent agent software is not available, such as legacy mainframe systems.

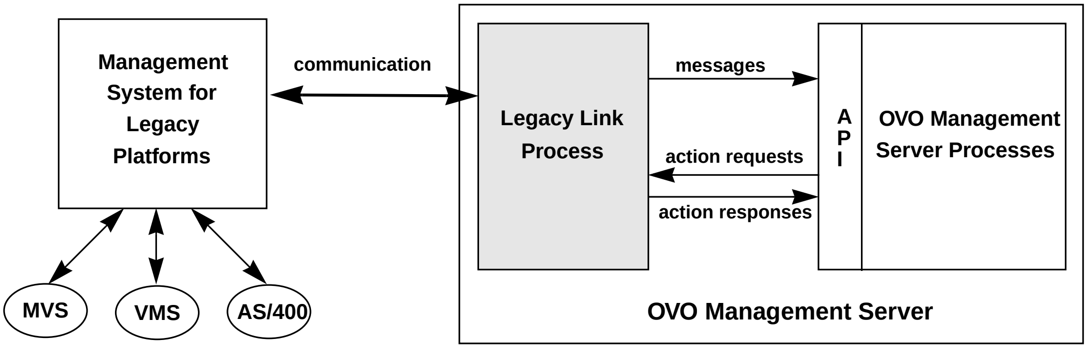
This API provides an interface to a process running on the management server node. This process can send (write) messages, receive action requests, and send action responses. It must handle all communication with the network node that lacks the intelligent agent, and it must ensure that the messages sent to the management server are correct and complete.

A Legacy Link Interface process running on the management server can use the Interface API functions to do the following:

- ❏ Pass messages from a legacy system to the internal message stream of the management server,
- ❏ Read action requests from the management server,
- ❏ Write action responses.

Figure 4-5 shows an overview of how to integrate a legacy system using the Legacy Link Interface. To do this, a **legacy link process** must be running on the management server; this process is shaded in gray in the figure. The legacy link process must manage the communication with the legacy systems, and also communicate with the management server processes using the functions of the Legacy Link Interface.

Figure 4-5 Integrating a Legacy System Using the Legacy Link Interface



Structure of the Legacy Link Process

Figure 4-6 is a flowchart to illustrate the order in which the API functions are used when a legacy system is fully integrated.

The legacy link process in this description is a process that runs on the management server. This process communicates with the OVO management server processes using three instances of the interface:

❑ One to send (write) messages to the management server,

❑ One to receive (read) action requests from the management server,

❑ One to send (write) action responses to the management server.

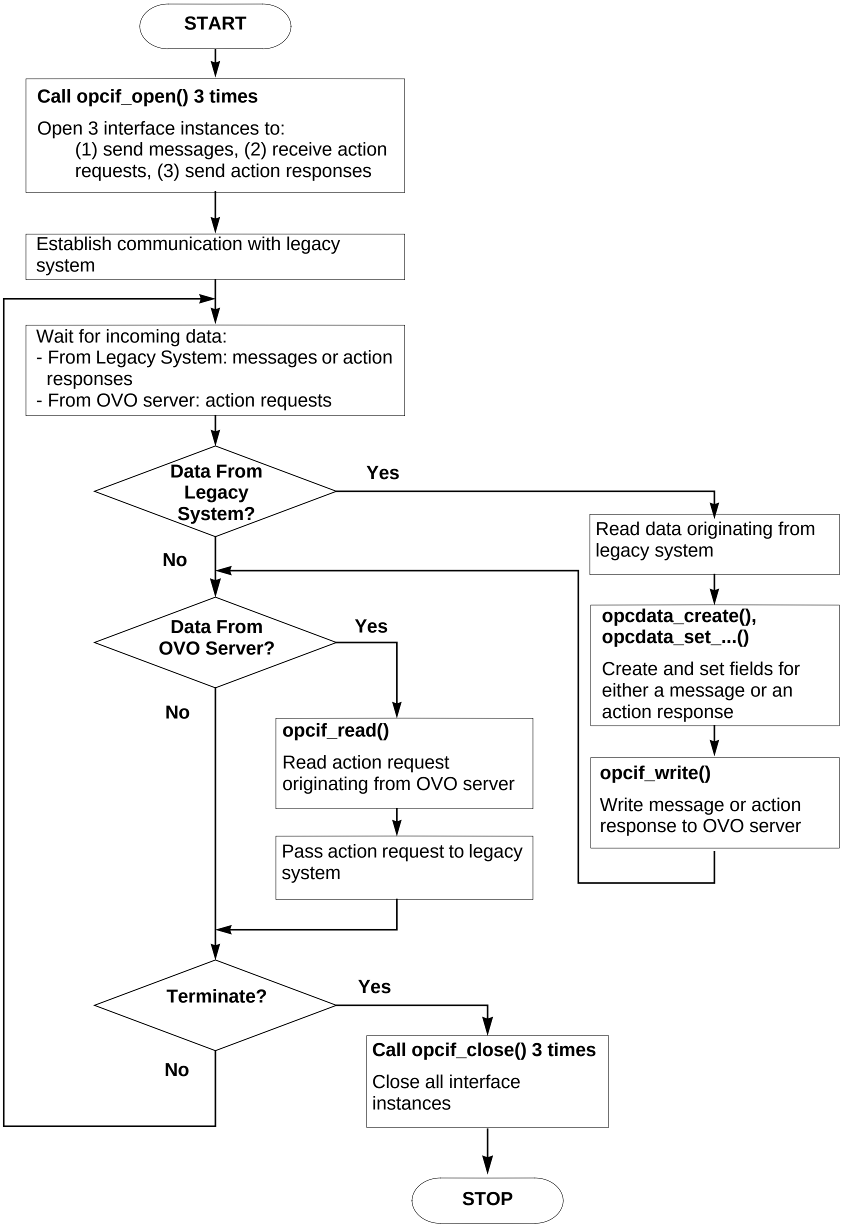
In addition, the legacy link process must establish a bidirectional communication channel with the legacy system to receive input for generating messages and action responses, and to pass on action requests received from the management server.

As soon as the communication channel between the legacy system and the OVO management server is established, the legacy link process waits for incoming data from both communication partners.

When incoming data originates from the legacy system, either an action response or a message is generated and sent to the OVO management server. If the incoming data originates from the OVO management server, it is an action request that is processed, then sent to the legacy system.

Note that action responses confirm that an action has occurred, for example, an action response might show that an action has been completed successfully. When immediate local actions are configured, an action response should be sent to show that an action has completed even though an action request was not received from the OVO management server.

Figure 4-6 Using the Legacy Link Interface to Integrate a Legacy System


START
Call opcif_open() 3 times
Open 3 interface instances to:
(1) send messages, (2) receive action requests, (3) send action responses
Establish communication with legacy system
Wait for incoming data:
- From Legacy System: messages or action responses
- From OVO server: action requests
Data From Legacy System?
Yes
No
Read data originating from legacy system
opcdata_create(), opcdata_set_...()
Create and set fields for either a message or an action response
opcif_write()
Write message or action response to OVO server
Data From OVO Server?
Yes
No
opcif_read()
Read action request originating from OVO server
Pass action request to legacy system
Terminate?
Yes
No
Call opcif_close() 3 times
Close all interface instances
STOP

Overview of the Message Event Interface

The **Message Event Interface** enables an application running on the OVO management server to register for and receive **Message Events**. A message event is generated whenever the status of an OVO message changes in any way. For example, a message event can be generated if an annotation is added to a message, if the ownership of a message changes, etc. This is a read-only interface, so that if a message event is sent from a source, for example, the OVO Message Browser, the application will be informed.

This interface might be used, for example, by a message viewing application to update the status of the messages displayed by the application.

Access to Message Events

If an application registers for message events of the OPCSVIF_MSG_EVENTS interface, the read queue for this interface instance is created by OVO and accessed by the client, see Figure 4-8 on page 189. The display manager registers the instance and writes all desired message events to this queue. To use this interface, it is not necessary to enable it in the configuration of the management server. The only restriction is that the application must run with root permissions on the management server.

You can get the following message events by way of the Message Event Interface:

Table 4-2 Message Event Flags

Description	Event Flag
All message events	OPC_MSG_EVENT_ALL
Message attributes changed	OPC_MSG_EVENT_CHANGE
Message in active browser acknowledged	OPC_MSG_EVENT_ACK
Message in history browser unacknowledged	OPC_MSG_EVENT_UNACK
Message owned	OPC_MSG_EVENT_OWN
Message disowned	OPC_MSG_EVENT_DISOWN
Annotation added	OPC_MSG_EVENT_ANNO
Annotation deleted	OPC_MSG_EVENT_NO_ANNO
Message escalated to another server	OPC_MSG_EVENT_ESCALATED
Message escalated from another server	OPC_MSG_EVENT_ESCALATED_FROM
Automatic action started	OPC_MSG_EVENT_AA_START
Automatic action finished	OPC_MSG_EVENT_AA_END
Operator-initiated action started	OPC_MSG_EVENT_OA_START
Operator-initiated action finished	OPC_MSG_EVENT_OA_END

Overview of the Application Response Interface

The **Application Response Interface** enables an external application to register for and receive **Application Responses** from OVO applications that have been started by the Application API `opcappl_start()`. OVO generates an application response whenever the status of a running application changes.

Access to Action Responses

The Application Response Interface (`OPCSVIF_APPLIC_RESPONSE`) works in a similar way to the Message Events Interface (`OPCSVIF_MSG_EVENTS`). When an application registers for application responses, a queue is opened to which the application responses are written. The function `opcif_read()` reads the application responses from this queue, see Figure 4-8 on page 189.

Read and Write Access to the OVO Message Stream

Figure 4-7 on page 188 shows the difference between the three types of process that access the OVO message interface. **Read-write applications** are message processing programs that read messages and then modify attributes or create new messages depending on the input flow. Any resulting messages are then written back to the OVO message stream. **Read-only applications** only read the message flow, without writing messages back to the interface. These applications might include programs for statistical purposes, debugging tools for support, or a separate presentation layer. **Write-only applications** are external agents which send messages to OVO in the same way as the OVO agents. These messages should go through the Server or Agent Message Stream Interface.

Figure 4-7 on page 188 shows the message flow within the message manager and the different queues used for the interface. It also shows the Message Event flow and Application Response flow within the display manager. When an external application has opened an interface instance and registered, the server generates a queue and sends the ID of the queue to the application. The application can then open and read that queue.

Figure 4-8 on page 189 shows how the display manager and the other processes communicate.

Figure 4-7 Overview of the Interface APIs

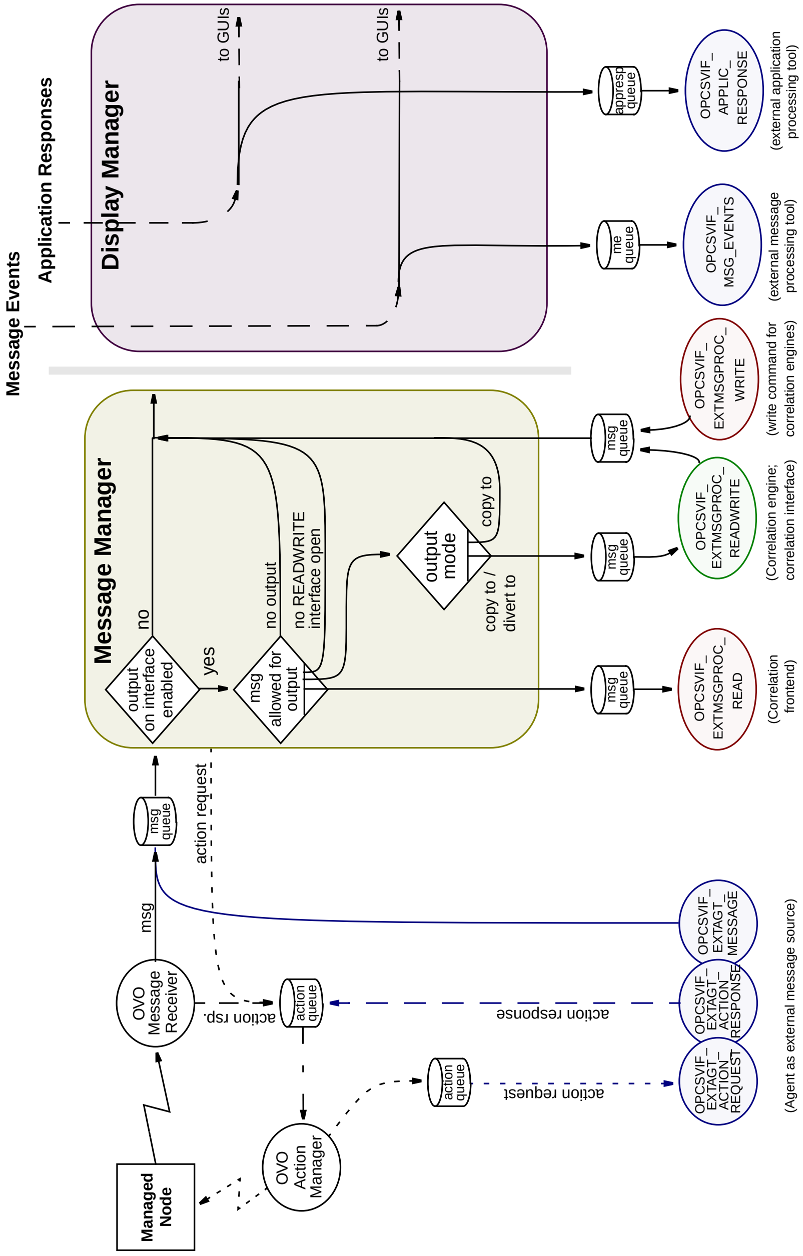
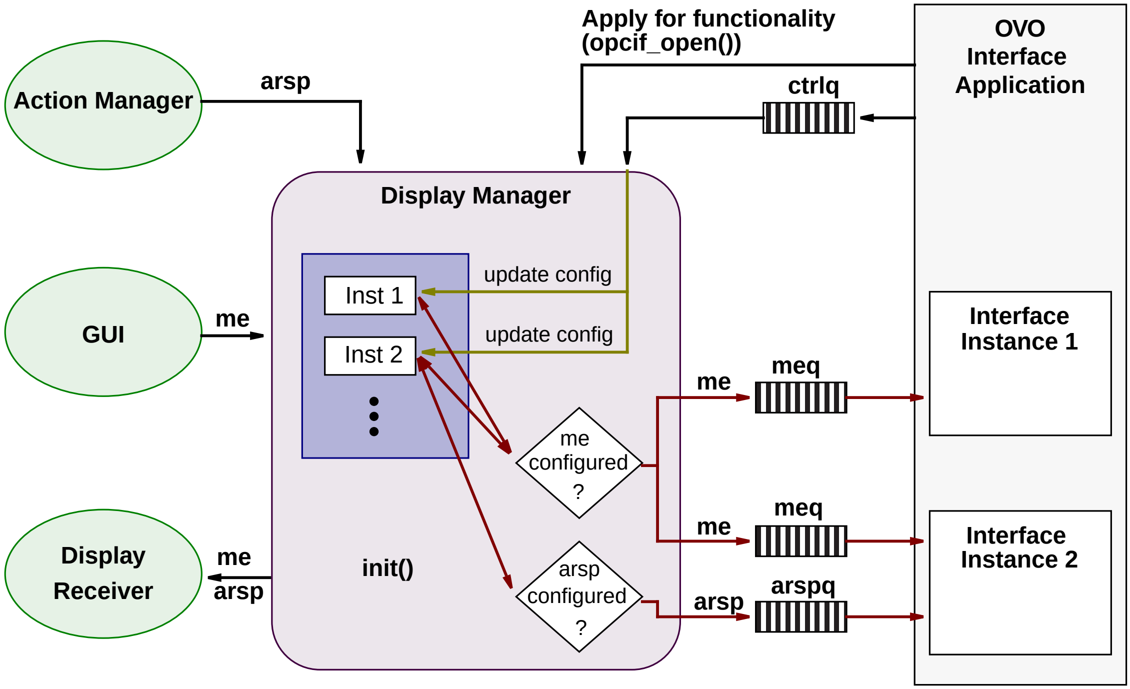

Message Events
Application Responses
Display Manager
to GUIs
to GUIs
apresp queue
me queue
OPCSVIF_ APPLIC_ RESPONSE
(external application processing tool)
OPCSVIF_ MSG_EVENTS
(external message processing tool)
Message Manager
output on interface enabled
no
yes
msg allowed for output
no output
no READWRITE interface open
output mode
copy to
copy to / divert to
msg queue
msg queue
msg queue
OPCSVIF_ EXTMSGPROC_ WRITE
(write command for correlation engines)
OPCSVIF_ EXTMSGPROC_ READWRITE
(Correlation engine; correlation interface)
OPCSVIF_ EXTMSGPROC_ READ
(Correlation frontend)
msg queue
action request
msg
OVO Message Receiver
action rsp.
action queue
action response
OPCSVIF_ EXTAGT_ MESSAGE
OPCSVIF_ EXTAGT_ ACTION_ RESPONSE
OPCSVIF_ EXTAGT_ ACTION_ REQUEST
(Agent as external message source)
action queue
action request
OVO Action Manager
Managed Node

Figure 4-8 Communication With the Display Manager



Key:

me = Message Event
meq = Message Event Queue
arsp = Application Response
arspq = Application Response

The OVO Operator APIs

This group of APIs enable actions to be directly performed on OVO data. These APIs include:

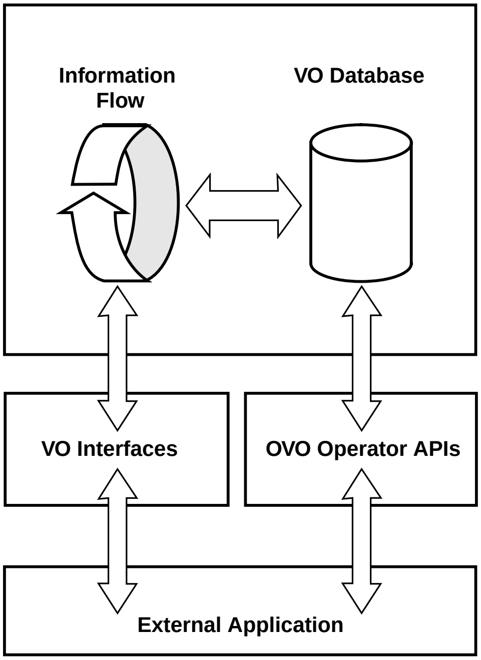
API	Description
OVO Data API	API to get/set information in OVO data structures. This API is used for: • OVO Data Structures • Containers • Iterators
OVO Interface APOVO	API to connect to the OVO Interfaces; it contains the functions: `opcif_*()` and `opcreg_*()`
Server Message API	API on the management server to perform operations on OVO messages; it contains the functions: `opcmsg_*()` and `opcanno*()`
Agent Message API	API on managed nodes to perform operations on OVO messages; it contains the functions: `opcagtmsg_*()` and `opcmsg()`
Agent Monitor API	API on managed nodes to send monitor values to the management server; it contains the functions: `opcagtmon_*` and `opcmon()`.

In addition to these APIs, the commands `opcackmsg(1M)` and `opcack(1M)` are provided to acknowledge messages from external sources. The command `opcmack(1M)` is also available on the managed node to acknowledge a message for OVO.

The OVO Interfaces and the OVO Operator API — A Comparison

Figure 4-9 shows the different methods by which the OVO Interfaces and the OVO Operator API access OVO data. The OVO Interfaces must open an interface to the information flow on the agent or server; register to receive or send information; be given permission to receive or send that information; and finally close the interface. The OVO Operator APIs, however, access the OVO database directly, without opening an interface instance, to perform the required action(s).

Figure 4-9 **Comparison of the OVO Service APIs and the OVO Operator APIs**



The OVO Configuration APIs

This group of APIs accesses OVO configuration data. These APIs include:

API	Description
Connection API	API to connect to and disconnect from the OVO database; it contains the functions `opc_connect()` and `opc_disconnect()`
Application Configuration API	API to configure OVO applications and application groups; it contains the functions `opcappl_*()`
Application Group Configuration API	API to configure OVO application groups; it contains the functions `opcapplgrp_*()`
Message Group Configuration API	API to configure OVO message groups; it contains the functions `opcmsggrp_*()`
Message Regroup Condition Configuration API	API to configure OVO message regroup condition; it contains the functions `opcmsgregrp_*()`
Node Configuration API	API to configure OVO nodes and node groups; it contains the functions `opcnode_*()` and `opcnodegrp_*()`

Node Hierarchy Configuration API

API to configure OVO node layout groups; it contains the functions `opcnodehier_*()`

Template Configuration API

API to configure message source templates and template groups; it contains the functions `opctempl*()`

User Profile Configuration API

API to configure OVO user profiles; it contains the functions `opcprofile_*()`

User Configuration API

API to configure OVO users; it contains the functions `opcuser_*()`

Distribution API

Distribution API to distribute agent configuration to managed nodes

Synchronization API

Accesses, modifies, and synchronizes OVO configuration data

Summary of OVO API Functions

Functions of the OVO Data API

Functions to Manipulate OVO Data Structures

See the man page *opcdata_api(3)* for information about these functions and *opcdata(3)* for information about the OVO data-specific structures.

Table 4-3 Overview of the Data Access and Creation Functions

Function Call	Description
`opcdata_append_element()`	Appends a copy of the given element to the container
`opcdata_clear()`	Clears all fields in the container
`opcdata_copy()`	Makes a copy of the specified opcdata structure
`opcdata_copy_info_to_actresp()`	Copies components from a message or action request to an action response
`opcdata_create()`	Creates an empty data structure of type data_type
`opcdata_delete_element()`	Deletes the element in the container and frees the memory
`opcdata_free()`	Deallocates previously allocated memory
`opcdata_generate_id()`	Generates an OVO UUID
`opcdata_get_cma()`	Gets the value of a specified custom message attribute
`opcdata_get_cmanames()`	Returns a list of all custom message attributes available for a message
`opcdata_get_double()`	Gets the double value of the specified attribute
`opcdata_get_element()`	Makes a copy of the desired element

Table 4-3 Overview of the Data Access and Creation Functions

Function Call	Description
opcdata_get_error_msg()	Returns the error description for the given error code (thread-safe)
opcdata_get_long()	Get long value of specified attribute
opcdata_get_str()	Gets string value of specified attribute
opcdata_insert_element()	Inserts a copy of the given element at given index
opcdata_ladd()	Adds an element to the specified list
opcdata_ldel()	Deletes an element from the specified list
opcdata_lget_len()	Returns the number of an element in the list
opcdata_lget_long()	Returns the value of the attribute in the list
opcdata_lget_str()	Returns a pointer to the string of the attribute in the list
opcdata_lset_long()	Replaces the value in the attribute of the list element
opcdata_lset_str()	Replaces the string in the attribute of the list element
opcdata_num_elements()	Returns the number of elements of the container
opcdata_report_error()	Returns the error description for the given error code (not thread-safe)
opcdata_set_cma()	Sets a custom message attribute (name and value) for a message
opcdata_set_double()	Sets double attribute in data to value
opcdata_set_long()	Sets attribute in data to value
opcdata_set_str()	Sets string attribute in data to value
opcdata_remove_cma()	Removes custom message attributes from a message
opcdata_type()	Returns the opcdata type

Functions of the OVO Iterator

See the man page *opciter(3)* for more information about these functions.

Table 4-4 Overview of the Iterator Functions

Function Call	Description
`opciter_begin()`	Sets the iterator to the first element and returns its pointer.
`opciter_create()`	Creates an iterator
`opciter_end()`	Returns past-end-value (== NULL)
`opciter_free()`	Frees memory of the iterator
`opciter_get_pos()`	Returns actual position of the iterator
`opciter_next()`	Returns pointer to the next container element and increments the iterator
`opciter_nth()`	Returns pointer to the element on the nth position. The iterator remains unchanged
`opciter_prev()`	Returns pointer to the previous container element and decrements the iterator
`opciter_set_pos()`	Sets iterator to specified position

The OVO Data Structures

Table 4-5 Overview of the OVO Data Structures

OVO Data Type	Description
OPCDTYPE_ACTION_REQUEST	Defines an OVO action request
OPCDTYPE_ACTION_RESPONSE	Contains the response of a previously started action
OPCDTYPE_ANNOTATION	Contains a message annotation
OPCDTYPE_APPLIC	Defines an OVO application
OPCDTYPE_APPLIC_RESPONSE	Contains the response of a previously started OVO application
OPCDTYPE_APPL_CONFIG	Contains the configuration information of an OVO application
OPCDTYPE_APPL_GROUP	Contains the configuration information of an application group
OPCDTYPE_CONTAINER	Contains a list of OVO objects
OPCDTYPE_EMPTY	Creates an empty structure
OPCDTYPE_INFORM_USER	Contains the name of the user and the message
OPCDTYPE_LAYOUT_GROUP	Defines an OVO node layout group
OPCDTYPE_MESSAGE	Contains information about the attributes of an OVO message
OPCDTYPE_MESSAGE_EVENT	Contains a message event
OPCDTYPE_MESSAGE_GROUP	Contains the name and description of an OVO message group
OPCDTYPE_MESSAGE_ID	Contains an OVO message ID
OPCDTYPE_MONITOR_MESSAGE	Contains the most necessary information for a monitor value
OPCDTYPE_NODE	Contains limited information about an OVO managed node

Table 4-5 Overview of the OVO Data Structures (Continued)

OVO Data Type	Description
OPCDTYPE_NODE_CONFIG	Contains a complete definition of an OVO managed node with all its attributes
OPCDTYPE_NODE_GROUP	Defines an OVO node group
OPCDTYPE_NODEHIER	Defines an OVO node hierarchy
OPCDTYPE_REGROUP_COND	Contains configuration information about a message regroup condition
OPCDTYPE_TEMPLATE_INFO	Contains the most necessary information to define an OVO message source template or template group
OPCDTYPE_USER_CONFIG	Contains the configuration of an OVO user
OPCDTYPE_USER_RESP_ENTRY	Contains the responsibility information of an OVO user

Functions of the OVO Service APIs

Functions to Access the OVO Interface

Table 4-6 Overview of the OVO Service API Functions

Function Call	Description
opcif_close()	Terminate a connection to the interface
opcif_get_pipe()	Returns pipefd of its interface input queue
opcif_open()	Opens an instance of the Server/Agent MSI, Legacy Link Interface, Message Event Interface, or Application Response Interface
opcif_read()	Read information from specified interface
opcif_register()	Register for certain attributes
opcif_unregister()	Unregister condition
opcif_write()	Writes a message to the interface

Functions to Access the Registration Conditions

See the man page *opcregcond*(3) for information about these functions.

Table 4-7 Overview of the Registration Condition Access and Creation Functions

Function Call	Description
opcreg_create()	Creates an empty registration condition structure
opcreg_copy()	Creates a copy of registration condition
opcreg_free()	Deallocates previously allocated memory
opcreg_get_long()	Gets value of given field
opcreg_get_str()	Gets string value of given field
opcreg_set_long()	Sets attribute in regcond to value
opcreg_set_str()	Sets string attribute of regcond to value

Functions of the Server Message API

Functions to Manipulate Messages

Table 4-8 Overview of the Server Message API Functions

Function Call	Description
opcanno_add()	Adds an annotation to an existing message
opcanno_delete()	Deletes an annotation from a message
opcanno_get_list()	Gets a list of all existing annotations for a message
opcanno_modify()	Modifies an existing message annotation
opcmsg_ack()	Acknowledges an active message
opcmsg_disown()	Disown specified message
opcmsg_escalate()	Escalates a message to the escalation server
opcmsg_get()	Gets detailed information about a message
opcmsg_get_instructions()	Gets instructions for a message
opcmsg_modify()	Allows to modify the severity and message text of a given message
opcmsg_own()	Own the specified message
opcmsg_select()	Highlight specified message in the Message Browsers
opcmsg_start_auto_action()	Starts an automatic action
opcmsg_start_op_action()	Starts an operator-initiated action
opcmsg_unack()	Unacknowledges a message

Functions of the Agent Message API

Functions to Send/Acknowledge Messages

Table 4-9 Overview of the Agent Message API Functions

Function Call	Description
`opcagtmsg_ack()`	Acknowledges the specified message
`opcagtmsg_send()`	Writes a message into the queue of the message interceptor
`opcmsg()`	Writes a message into the queue of the message interceptor

Functions of the Agent Monitor API

Functions to Send Monitor Values

Table 4-10 Overview of the Agent Monitor API Functions

Function Call	Description
opcagtmon_send()	Writes a monitor value/name pair into the queue of the monitor agent
opcmon()	Writes a monitor value/name pair into the queue of the monitor agent

Functions of the Connection API

Functions to Connect to the Management Server

Table 4-11 Overview of the Connection API Functions

Function Call	Description
`opc_connect()`	Connect to the OVO database to get access.
`opc_disconnect()`	Disconnect from the OVO database.

Functions of the Application Configuration API

Functions to Configure OVO Applications

Table 4-12 Overview of the Application API Function

Function Call	Description
opcappl_add()	Adds an OVO application
opcappl_delete()	Deletes an OVO application
opcappl_get()	Gets the full configuration of an OVO application
opcappl_get_list()	Gets a list of all configured applications
opcappl_modify()	Modifies an OVO application
opcappl_start()	Starts an OVO application

Functions of the Application Group Configuration API

Functions to Configure OVO Application Groups

Table 4-13 Overview of the Application API Function

Function Call	Description
`opcapplgrp_add`	Adds an application group
`opcapplgrp_assign_applgrps()`	Assigns application groups to a specified application group
`opcapplgrp_assign_appls()`	Assigns applications to a specified application group
`opcapplgrp_deassign_applgrps()`	Deassigns application groups from a specified application group
`opcapplgrp_deassign_appls()`	Deassigns applications from a specified application group
`opcapplgrp_delete()`	Deletes an application group
`opcapplgrp_get()`	Gets the full configuration of a specified application group
`opcapplgrp_get_applgrps()`	Gets a list of all assigned application groups
`opcapplgrp_get_appls()`	Gets a list of all assigned applications
`opcapplgrp_get_list()`	Gets a list of all application groups
`opcapplgrp_modify()`	Modifies an application group

Functions of the Message Group Configuration API

Functions to Configure OVO Message Groups

Table 4-14 Overview of the Message Group Configuration API Functions

Function Call	Description
`opcmsggrp_add()`	Adds a message group to the OVO database
`opcmsggrp_delete()`	Removes a message group from the OVO database
`opcmsggrp_get_list()`	Gets a list of all message groups
`opcmsggrp_modify()`	Modifies an existing message group

Functions of the Message Regroup Condition Configuration API

Function to Configure OVO Message Regroup Conditions

Table 4-15 Overview of the Message Regroup Condition Configuration API Functions

Function Call	Description
`opcmsgregrp_add()`	Creates a new regroup condition
`opcmsgregrp_delete()`	Deletes a given regroup condition
`opcmsgregrp_get()`	Gets the attributes of a given regroup condition
`opcmsgregrp_get_list()`	Gets a list of all known regroup conditions
`opcmsgregrp_modify()`	Modifies a given regroup condition
`opcmsgregrp_move()`	Moves a given regroup condition to a new position in the condition list

Functions of the Node Configuration API

Function to Configure OVO Managed Nodes

Table 4-16 Overview of the Node Configuration API Function

Function Call	Description
`opcconf_get_nodes()`	Gets all configured nodes from the database
`opcnode_add()`	Adds a managed node to the database and the OVO node bank hierarchy
`opcnode_assign_templates()`	Assigns templates and template groups to a node
`opcnode_deassign_templates()`	Deassigns templates and template groups from a node
`opcnode_delete()`	Deletes a node from the database, and acknowledges all messages from that node
`opcnode_get()`	Gets the full configuration of a given managed node
`opcnode_get_defaults()`	Gets the default configuration of a node from the database
`opcnode_get_list()`	Gets a list of all managed nodes
`opcconf_get_nodes()`	Connects to the OVO database and reads the managed node configuration
`opcnode_get_templates()`	Gets a list of all templates assigned to the node
`opcnode_modify()`	Modifies the attributes of an existing node

Function to Configure OVO Node Groups

Table 4-17 Overview of the Node Configuration API Function

Function Call	Description
`opcnodegrp_add()`	Adds a node group to the database
`opcnodegrp_assign_nodes()`	Assigns nodes to a node group
`opcnodegrp_assign_templates()`	Assigns templates or template groups to a node group
`opcnodegrp_deassign_nodes()`	Deassigns nodes from a node group
`opcnodegrp_deassign_templates()`	Deassigns templates or template groups from node group
`opcnodegrp_delete()`	Deletes a node group from the database
`opcnodegrp_get()`	Gets a node group configuration
`opcnodegrp_get_list()`	Gets a list of all node groups
`opcnodegrp_get_nodes()`	Gets a container of nodes assigned to the node group
`opcnodegrp_get_templates()`	Gets a list of templates or template groups assigned from a node group
`opcnodegrp_modify()`	Modifies a node group in the database

Functions of the Node Hierarchy Configuration API

Functions to Configure OVO Node Hierarchies

Table 4-18 Overview of the Node Hierarchy Configuration API Functions

Function Call	Description
`opcnodehier_add()`	Adds a new node hierarchy
`opcnodehier_add_layoutgrp()`	Adds a new layout group
`opcnodehier_copy()`	Copies a node hierarchy
`opcnodehier_delete()`	Deletes a node hierarchy
`opcnodehier_delete_layoutgrp()`	Deletes an empty layout group
`opcnodehier_get()`	Gets the attributes of a given node hierarchy
`opcnodehier_get_all_layoutgrps()`	Gets a list of all node layout groups in a node hierarchy
`opcnodehier_get_all_nodes()`	Gets a list of all nodes in a node hierarchy
`opcnodehier_get_layoutgrp()`	Gets the full configuration of a node layout group
`opcnodehier_get_layoutgrps()`	Gets a list of all layout groups assigned to a given layout group
`opcnodehier_get_list()`	Gets a list of all node hierarchies
`opcnodehier_get_nodeparent()`	Moves each node in the list to the specified node layout group
`opcnodehier_get_nodes()`	Gets a list of all nodes assigned to a layout group in the node hierarchy
`opcnodehier_modify()`	Modifies a node hierarchy
`opcnodehier_modify_layoutgrp()`	Modifies a node layout group
`opcnodehier_move_layoutgrp()`	Moves a layout group into another layout group
`opcnodehier_move_layoutgrps()`	Moves layout groups into another layout group
`opcnodehier_move_nodes()`	Moves a node from one location to another

Functions of the Template Configuration API

Function to Configure OVO Templates

Table 4-19 Overview of the Template Configuration API Functions

Function Call	Description
`opctempl_delete()`	Deletes an existing template from the OVO database
`opctempl_get_list()`	Gets a list of all templates from the OVO database
`opctemplfile_add()`	Adds templates to the OVO database
`opctemplfile_get()`	Gets details of the template and writes them into a file
`opctemplfile_modify()`	Modifies already existing OVO templates.

Functions to Configure OVO Template Groups

Table 4-20 Overview of the Template Configuration API Functions

Function Call	Description
`opctemplgrp_add()`	Adds a new template group to the database
`opctemplgrp_assign_templates()`	Assigns templates to a template group
`opctemplgrp_deassign_templates()`	Deassigns templates from a template group
`opctemplgrp_delete()`	Deletes a template group from the OVO database
`opctemplgrp_get()`	Gets the configuration of a template group from the OVO database
`opctemplgrp_get_templates()`	Gets a list of templates / template groups assigned to a template group
`opctemplgrp_modify()`	Modifies an already existing template group

Functions of the User Profile Configuration API

Functions to Configure OVO User Profiles

Table 4-21 **Overview of the User Profile API Function**

Function Call	Description
`opcprofile_add()`	Adds a user profile
`opcprofile_assign_applgrps()`	Assigns application groups to a user profile
`opcprofile_assign_appls()`	Assigns applications to a user profile
`opcprofile_assign_profiles()`	Assigns user profiles to a user profile
`opcprofile_assign_resps()`	Assigns responsibilities to a user profile
`opcprofile_deassign_applgrps()`	Deassigns application groups from a user profile
`opcprofile_deassign_appls()`	Deassigns applications from a user profile
`opcprofile_deassign_resps()`	Deassigns responsibilities from a user profile
`opcprofile_delete()`	Deletes a user profile
`opcprofile_get()`	Gets the full configuration of a user profile
`opcprofile_get_applgrps()`	Gets a list of all assigned application groups
`opcprofile_get_appls()`	Gets a list of all assigned applications
`opcprofile_get_list()`	Gets a list of all existing user profiles
`opcprofile_get_profiles()`	Gets a list of all assigned user profiles
`opcprofile_get_resps()`	Gets a list of all assigned responsibilities
`opcprofile_modify()`	Modifies a user profile

Functions of the User Configuration API

Functions to Configure OVO Users

Table 4-22 Overview of the User API Function

Function Call	Description
opcuser_add()	Adds an OVO user
opcuser_assign_applgrps()	Assigns application groups to an OVO user
opcuser_assign_appls()	Assigns applications to an OVO user
opcuser_assign_nodehier()	Assigns a node hierarchy to an OVO user
opcuser_assign_profiles()	Assigns profiles to an OVO user
opcuser_assign_resps()	Assigns responsibilities to an OVO user
opcuser_deassign_applgrps()	Deassigns application groups from an OVO user
opcuser_deassign_appls()	Deassigns applications from an OVO user
opcuser_deassign_profiles()	Deassigns profiles from an OVO user
opcuser_deassign_resps()	Deassigns responsibilities from an OVO user
opcuser_delete()	Deletes an OVO user
opcuser_get()	Gets the full configuration of an OVO user
opcuser_get_applgrps()	Gets a list of all assigned application groups
opcuser_get_appls()	Gets a list of all assigned applications
opcuser_get_list()	Gets a list of all existing OVO users
opcuser_get_nodehier()	Gets the assigned node hierarchy
opcuser_get_profiles()	Gets a list of all assigned profiles
opcuser_get_resps()	Gets a list of all assigned responsibilities
opcuser_modify()	Modifies an OVO user

Functions of the Distribution API

Functions to Distribute Configuration to Managed Nodes

Table 4-23 Overview of the Distribution API Function

Function Call	Description
`opc_distrib()`	Distributes the OVO agent configuration to managed nodes.

Functions of the Server Synchronization API

Functions to Modify and Update Configuration Data

Table 4-24 Overview of the Server Synchronization API Functions

Function Call	Description
`opc_inform_user()`	Informs all affected user interfaces
`opcsync_inform_server()`	Synchronizes the server processes after configuration changes
`opcsync_inform_user()`	Informs all affected users of any changes to data configuration
`opctransaction_commit()`	Finishes the current transaction and commits all transactions to the database
`opctransaction_rollback()`	Rolls back changes to the current user transaction
`opctransaction_start ()`	Starts a user transaction

Using APIs in Internationalized Environments

All OVO API functions are internationalized. This means that they will initialize the language setting, check the codeset for compatibility, and convert codesets if necessary, provided your API programs support Native Language Support (NLS) environments.

When writing API programs for internationalized environments, you must ensure that your programs do select the appropriate locale. In C programs, you do this by calling the function `setlocale()` at the beginning of your program.

It is recommended to use `setlocale(LC_ALL,"")`. The category `LC_ALL` names the program's entire locale. `""` adopts the setting of the current shell.

See the man page *setlocale(3C)* for more information about the `setlocale()` function.

5 Integrating with Java GUI

In This Chapter

This chapter describes the concept, integration details and usage of the Java GUI Remote APIs.

Overview of the Java GUI Remote APIs

Java GUI Remote APIs enable you to control certain features in the Java GUI remotely from other Java applications. You can resolve problems without searching for problem causing elements in the Java GUI. Java GUI Remote APIs are useful if you have mapped OVO services and nodes in other applications.

For the list of available Remote APIs that you use to control these features, see “Summary of Java GUI Remote APIs Methods” on page 233.

The Java GUI acts as a server, which listens for API calls from clients. In this case, clients are Java applications that execute Java GUI Remote APIs in the Java GUI. For more details on how clients connect to the Java GUI, see “Connecting to Java GUI” on page 230.

A client can be one of the following:

❑ **Independent application running on a localhost**

You can run a Java applet or application separately from the Java GUI. The only prerequisite is that the client is using the Java Interface from the Java GUI Remote APIs. You can execute actions, such as `Open Root Cause Graph` or `Open Filtered Message Browser`, through Remote APIs without physically using the Java GUI.

❑ **Java applet running inside the Java GUI**

A Java applet can run as an integrated add-on component in the workspace of the Java GUI.

NOTE

The Java GUI can act as an application or an applet running in Internet Explorer or Netscape Navigator. For the security reasons, all applets running in the workspace of the Java GUI must be signed.

To ensure the connection between your client and the Java GUI is established, you *must* enable Java GUI Remote APIs on the management server. See “Enabling the Java GUI Remote APIs” on page 223 for information on how to enable the Remote APIs.

You must create your own client that will establish a connection with the Java GUI. When the connection is established, you will be able to execute the Remote APIs in the Java GUI. For information on how to create your own client, see “Creating the Client” on page 223.

It is possible to connect to Java GUI with more than one client at a time. A synchronization mechanism ensures that the first called API finishes before the next one is called.

NOTE Java GUI Remote APIs allow you to execute *only* URL applications. Execution of OVO applications using the Java GUI Remote APIs is disabled for security reasons.

Calling the Java GUI Remote APIs

The `ito_op_api_cli` is a command line wrapper script for the Java Remote APIs client. Client `com.hp.ov.it.ui.api.examples.japiwrap` is located in `ito_op_API.jar` and `ito_op.jar` files. The `ito_op_api_cli` script enables calling the `japiwrap` program, which implements a simple JavaRemote client, from a command line.

Using `ito_op_api_cli`, all parameters needed to successfully start the program are passed to the `japiwrap` program. The JavaRemote client executes several basic RemoteAPIs, such as: starting a new JavaGUI, opening a new message browser, opening new service graphs and so on.

This script can be found in the following locations:

UNIX: `/opt/OV/www/htdocs/ito_op/`

Windows: `C:\Program Files\Hewlett-Packard\HP OVO Java Console`

For more information see the `ito_op_api_cli` man page.

Configuring the Java GUI Remote APIs

This section describes how to enable Java GUI Remote APIs, and how to create and run the client required to establish a connection to the Java GUI.

Enabling the Java GUI Remote APIs

Enable the Java GUI Remote APIs by setting a `JGUI_API_ENABLED` variable in a configuration file on the OVO management server. Enter the following:

```
/opt/OV/bin/ovconfchg -ovrg server -ns opc -set\
JGUI_API_ENABLED TRUE
```

IMPORTANT You *must* restart Service Navigator for the changes to take effect.

After Java GUI Remote APIs are enabled, an additional icon is displayed in the Java GUI. Refer to the *OVO Java GUI Operator's Guide* for more information.

NOTE If you want to disable the Java GUI Remote APIs, set the `JGUI_API_ENABLED` parameter to `FALSE`, or remove the entry from the configuration file.

Creating the Client

IMPORTANT For creating and using the client, you *must* have the Java2SDK1.4.1 installed on your system.

To create your own client, you have to implement the following:

❑ Your preferred type of user interface (command line or graphical).

- ❏ An instance of OV_JGUI_JavaBridge class, which holds the OV_JGUI_RemoteProxy class with all Remote APIs defined.

See “Example of the Basic Client Implementation” on page 225 for more information.

NOTE You have the possibility to create a more advanced client that, for example, automatically starts the Java GUI on a localhost when no Java GUIs are running. See “Example of Creating the Client with Automatic Java GUI Startup on a Localhost” on page 226 to learn more about creating the client that provides automatic Java GUI startup on a localhost.

For creating the client, you can use one of the following provided files:

- ❏ ito_op_API.jar
- ❏ ito_op_addon.jar and ito_op.jar

These files are located in JavaGUI installation directory. They contain the following predefined classes:

- ❏ Predefined classes:
  - com.hp.ov.it.api.client.OV_JGUI_RemoteProxy
  - com.hp.ov.it.api.client.OV_JGUI_JavaBridge
  - com.hp.ov.it.api.common.OV_JGUI_RemoteFilterData

You can access the detailed Java GUI Remote APIs Specification, through the following URL:

http://*<management_server>*:3443/ITO_DOC/C/manuals/APIdoc

In this instance, *<management_server>* is the fully qualified hostname of your management server.

For information on how to compile and run your client, see “To Compile the Client” on page 228 and “To Run the Client” on page 229.

Example of the Basic Client Implementation

The following example shows how to create a user interface and the corresponding instance of OV_JGUI_JavaBridge class.

```
package com.hp.ov.it.api.samples;

 import com.hp.ov.it.api.client.*;
 import com.hp.ov.it.api.common.OV_JGUI_RemoteFilterData;
 import java.util.Vector;

 public class RemoteClient
 {
   public static void main(String args[])
   {
     if (args.length < 2)
     {
       System.out.println("\nUsage: RemoteClient < mgmtsv >
       < username >");
       return;
     }

     try
     {
        String mgmtsv  = args[0];
        String user    = args[1];

        //create JavaBridge instance that will connect to Java
         //GUI which is connect to the specified managment
        //server and specified user
        OV_JGUI_JavaBridge m_bridge =
        OV_JGUI_JavaBridge.getNewInstance(user, mgmtsv);

        //get remote proxy
        OV_JGUI_RemoteProxy  m_proxy =
        m_bridge.getRemoteProxy();

        //create a vector of services for filtering
        Vector s = new Vector();
        s.add("Customer Services");

        //create a vector of nodes for filtering
        Vector n = new Vector();
        n.add(mgmtsv);

        //create empty filter
        OV_JGUI_RemoteFilterData filter =
        new OV_JGUI_RemoteFilterData();

        //set nodes and services
        filter.setNodes(n);
        filter.setServices(s);

        //Show Filtered Active Message browser in browser pane
         m_proxy.openMessageBrowser(true, m_proxy.TYPE_ACTIVE,
        filter);
      }catch (OV_JGUI_CommunicationException e)
      {
        System.err.println("Error: "+e);
      }
    }
  }
```

Example of Creating the Client with Automatic Java GUI Startup on a Localhost

Java GUI Remote APIs provide you with the possibility to start Java GUI automatically with the user-defined parameters on a localhost.

By default, Java GUI is installed in the following directory:

❑ **On UNIX systems**

  `/opt/OV/www/htdocs/ito_op/`

❑ **On Windows systems**

  `C:\Program Files\Hewlett-Packard\HP OVO Java Console`

However, you can define a custom installation path using the `setInstallDir` method. For the list of available Remote APIs, see “Summary of Java GUI Remote APIs Methods” on page 233.

The following example shows how to configure Java GUI automatic startup on a localhost:

```
package com.hp.ov.it.api.samples;

 import com.hp.ov.it.api.client.*;
 import com.hp.ov.it.api.common.OV_JGUI_RemoteFilterData;
 import java.util.Vector;
 import java.net.URL;

 public class AutoStartup
 {
```

```
public static void main(String args[])
{
  if (args.length < 34)
  {
    System.out.println("\nUsage: AutoStartup < mgmtsv >\
    < username > < password > < apisid > [-trace]");
    return;
  }

  try
  {
    String mgmtsv  = args[0];
    String user    = args[1];
    String passwd  = args[2];
    String apisid  = args[3];
    boolean mode = OV_JGUI_JavaBridge.MODE_APPLICATION;

    for (int i=0; i < args.length; i++)
    {
      if (args[i].equals("-trace"))
      OV_JGUI_Logger.setTrace(true);
    }
    OV_JGUI_JavaBridge m_bridge =
    OV_JGUI_JavaBridge.getNewInstance(user, mgmtsv,apisid);

    //Checking if JavaGUI is up
    if (!m_bridge.isUp())
    {
       //Setting a custom install path where JavaGUI is
      installed
      m_bridge.setInstallPath("D:\\ProgramFiles\\/
      Hewlett-Packard\\HP OVO Java Console");

      //Launch JavaGUI
      try
      {
        int port = m_bridge.launch(passwd, mode);
      } catch (OV_JGUI_TimeOutException e)
      {
        System.out.println(e);
        return;
      }
    }

    URL  url = new URL("http://www.hp.com");
```

```
            //User should obtain OV_JGUI_RemoteProxy instance after
             communication with JavaGUI has been established
            OV_JGUI_RemoteProxy m_proxy = m_bridge.getRemoteProxy();

            //Call Remote API
            m_proxy.startURL(url);

          }catch (Exception e)
          {
            System.err.println("Error: "+e);
            e.printStackTrace();
          }
        }
      }
```

To Compile the Client

Before you compile your client, the following files have to be located in the current directory:

- ❏ `RemoteClient.java` source file
- ❏ One of the following:
  - `ito_op_API.jar` file
  - `ito_op.jar` and `ito_op_addon.jar` files

To compile the client, perform the following:

1. Make sure that `javac.exe` is included in your PATH variable.
2. Compile the client.
   - *For UNIX systems*

     Enter one of the following:

     — **`java -classpath ito_op_API.jar com.hp.ov.it.api.samples.RemoteClient`**

     — **`javac -d . -classpath ito_op_addon.jar:ito_op.jar RemoteClient.java`**

- *For Windows systems*

  Enter one of the following:

  — **`javac -d . -classpath ito_op_API.jar RemoteClient.java`**

  — **`javac -d . -classpath ito_op_addon.jar;ito_op.jar RemoteClient.java`**

To Run the Client

Before you run your client, at least one Java GUI has to be running on your system.

To run the client, perform the following:

1. Make sure that `java.exe` is included in your PATH variable.
2. Start the client.

   - *For UNIX systems*

     Enter one of the following:

     — **`java -cp .:ito_op_API.jar com.hp.ov.it.api.samples.RemoteClient`**

     — **`java -cp .:ito_op.jar:ito_op_addon.jar com.hp.ov.it.api.samples.RemoteClient`**

   - *For Windows systems*

     Enter one of the following:

     — **`java -cp .;ito_op_API.jar com.hp.ov.it.api.samples.RemoteClient`** ***`<management_server> <username>`***

     — **`java -cp .;ito_op.jar;ito_op_addon.jar com.hp.ov.it.api.samples.RemoteClient`** ***`<management_server> <username>`***

   Where *`<management_server>`* is the fully qualified hostname of your management server, and *`<username>`* is name of the user that is logged into the Java GUI.

Connecting to Java GUI

This section describes the process of establishing a connection between the client you have created and the Java GUI. The connection is achieved through the port repository file.

The Port Repository File

The port repository file, named `OV_JGUI_portRepository`, is automatically created in the user's home directory upon configuring the Java GUI Remote APIs. It registers information related to a particular Java GUI, such as username, management server, port number, and session ID. For more information about the session ID, see "Assigning a Session ID to Java GUI" on page 231.

NOTE

When the Java GUI Remote APIs are enabled on the management server, an available port number is automatically assigned to the Java GUI during its startup.

The client-server communication is established when the client connects to the Java GUI using the port number and the Java GUI session ID obtained from the port repository file.

For each running Java GUI, a new line is appended to the port repository file. The syntax is as follows:

`<username> <management_server> <port_number> <session_ID>`

The following naming scheme explains the above mentioned terms:

`<username>`	Username logged on to the Java GUI.
`<management_server>`	Management server to which the Java GUI is connected.
`<port_number>`	Port number that the client uses to connect to the Java GUI.
`<session_ID>`	Session ID of the Java GUI.

The following is an example of the port repository file, with two Java GUIs registered:

```
opc_op integra 3719
opc_op integra 3827 OV_JGUI_API
```

NOTE When Java GUI closes, the client disconnects from it and the session is removed from the port repository file.

It is recommended to create a client that is capable of re-establishing the connection to the Java GUI automatically. See “Example of Creating the Client with Automatic Java GUI Startup on a Localhost” on page 226 for more details on how to create a client with automatic Java GUI startup on a localhost.

Assigning a Session ID to Java GUI

A session ID is a string used to identify specific sessions of the Java GUI when more than one Java GUI is running on the same management server using the same username.

At the Java GUI startup, the session ID attribute value (*<session_ID>*) is automatically appended to the port repository file together with the *<username>*, *<management_server>* and *<port_number>* attribute values. For more information on the port repository file syntax, see “The Port Repository File” on page 230.

The default *<session_ID>* attribute value is `OV_JGUI_API`. However, you can specify the session ID manually. See “Specifying the Session ID Manually” on page 232 for more information.

The Session ID is unique for Java GUIs started on the different management servers, or using different usernames. Nevertheless, each additional Java GUI session started using the same username on the same management server acquires the session ID from the previously started Java GUI. The *<session_ID>* attribute value of the previously started Java GUI will be blank.

Specifying the Session ID Manually

You can specify the session ID manually by using one of the following methods:

❏ `getNewInstance` method in `OV_JGUI_JavaBridge` class

See "Summary of Java GUI Remote APIs Methods" on page 233 for a list of available Remote APIs.

❏ command line parameter `-apisid`

Set the `<session_ID>` attribute value at the Java GUI startup using the command line parameter `-apisid`. For example, on the Windows NT client, start the Java GUI by entering the following:

```
ito_op <management_server> -apisid=<user-defined_session_ID>
```

NOTE

If you try to set a session ID that is already registered in the port repository file using the same username on the same management server, the newly started Java GUI will acquire this particular session ID. The `<session_ID>` attribute value of previously started Java GUI will be blank.

Summary of Java GUI Remote APIs Methods

This section summarizes the Java GUI Remote APIs methods within the following classes:

- ❏ `OV_JGUI_RemoteProxy` class
- ❏ `OV_JGUI_JavaBridge` class

For more details about the available Java GUI Remote APIs, refer to the Java GUI Remote APIs Specification, which can be accessed through the following URL:

`http://`*`<management_server>`*`:3443/ITO_DOC/C/manuals/APIdoc`

In this instance, *`<management_server>`* is the fully qualified hostname of your management server.

OV_JGUI_RemoteProxy Class Methods

Table 5-1 Overview of OV_JGUI_RemoteProxy Class Methods

Methods	Description
`createWorkspace()`	Creates a new workspace.
`exists()`	Returns true if service is found in the Java GUI, otherwise false.
`getDefaultWorkspace()`	Returns a default workspace from the Java GUI.
`getRootServiceFromGraph()`	Returns a root service name of the selected service.
`getSelectedServices()`	Returns all selected services from the Java GUI in a vector.
`getSelectedViewType()`	Returns a selected service view’s type.
`getTargetURI()`	Returns the target where the proxy is sending requests.
`getUser()`	Returns a name of the user, logged into the Java GUI.
`highlightMessages()`	Highlights given messages in the Java GUI.
`moveToCenter()`	Positions service that matches a parameter name to the center of the service view if possible.

Table 5-1 Overview of OV_JGUI_RemoteProxy Class Methods (Continued)

`openMessageBrowser()`	Opens a new filtered message browser in a current workspace or in a browser pane.
`openServiceView()`	Opens a service view of a specified type on a given services.
`ping()`	Pings the server to check if it is alive.
`selectServiceInTree()`	Selects one or more services that are specified in the services parameter in the Object Pane of the Java GUI.
`selectServiceInView()`	Selects service in the currently selected view.
`selectServicesInView()`	Selects one or more services in the currently selected view.
`selectWorkspace()`	Selects a workspace in the Java GUI.
`setDefaultWorkspace()`	Specifies a default workspace in the Java GUI, where all service views are opened. Default workspace is the `Services` workspace.
`setVerbose()`	Determines if XML data should be printed on `stdout` during communication with the server.
`startURL()`	In a current workspace starts the given URL.
`toString()`	Returns a text representation of this proxy.

OV_JGUI_JavaBridge Class Methods

Table 5-2 Overview of OV_JGUI_JavaBridge Class Methods

Methods	Description
`destroy()`	Destroys the instance, and closes the connection to the server.
`findServerPort()`	Queries `OV_JGUI_portRepository` for a Java GUI running on the local machine that matches specified parameters.
`findServerPorts()`	Queries `OV_JGUI_portRepository` for Java GUIs running on the local machine that match the specified parameters.
`getHostname()`	Returns a hostname of the JavaBridge instance.
`getInstallPath()`	Returns the current installation path of the Java GUI.
`getLaunchTimeOut()`	Returns the current timeout for the method launch.
`getNewInstance()`	Creates and returns a new `OV_JGUI_JavaBridge` instance that will connect to the Java GUI running on a local machine.
`getPort()`	Returns a port number of the JavaBridge instance.
`getRemoteProxy()`	Returns the remote proxy element on which remote methods should be called.
`isUp()`	Checks if the Java GUI specified within the current bridge is up and running.
`launch()`	Launches the Java GUI that connects to the specified management server using the specified username, or using the username specified in the current JavaBridge instance.
`reinit()`	Calls the `destroy()` method and creates a new client instance and a new proxy element.
`setInstallPath()`	Sets the installation path where the Java GUI is installed.
`setLaunchTimeOut()`	Sets the timeout interval for a method launch.

Table 5-2 Overview of OV_JGUI_JavaBridge Class Methods (Continued)

Methods	Description
`toString()`	Prints out the current `OV_JGUI_JavaBridge` instance.

6 Integrating with Service Navigator

In This Chapter

HP OpenView Service Navigator is an add-on component of HP OpenView HP OpenView Operations. It enables you to manage your IT environment while focusing on the IT services you provide. See the *Service Navigator Concepts and Configuration Guide* for more information.

When using the standard Service Navigator product, service configuration is done with the command line tool `opcservice`. Service operation is performed with the Service Navigator GUI which displays the current status of the monitored services.

With the OVO Developer's Toolkit, it is possible to integrate directly with Service Navigator. The following integration facilities are provided:

❏ **XML Data Interface**

The XML Data Interface allows you to write or get service configuration directly into or from the service engine via a filesystem socket.

- Allows you to *write* the service configuration directly into the service engine. The configuration syntax follows the XML rules defined in the document type definition (DTD) `operations.dtd`.
- Allows you to *get* the current service configuration and service status directly from the service engine. The output syntax follows the XML rules defined in the DTD `results.dtd`.

The XML Data Interface is of special interest to integrators who, for example, want to provide service discovery scripts to automatically discover the services to be monitored by the Service Navigator integration.

❑ **C++ APIs of the service engine**

- The Service Operations Interface
- The Registration Interface for Service Status Changes

These APIs are C++ interfaces and come complete with:

- `opcsvcapi.h` header file
- `libopcsvcapi.sl` shared library

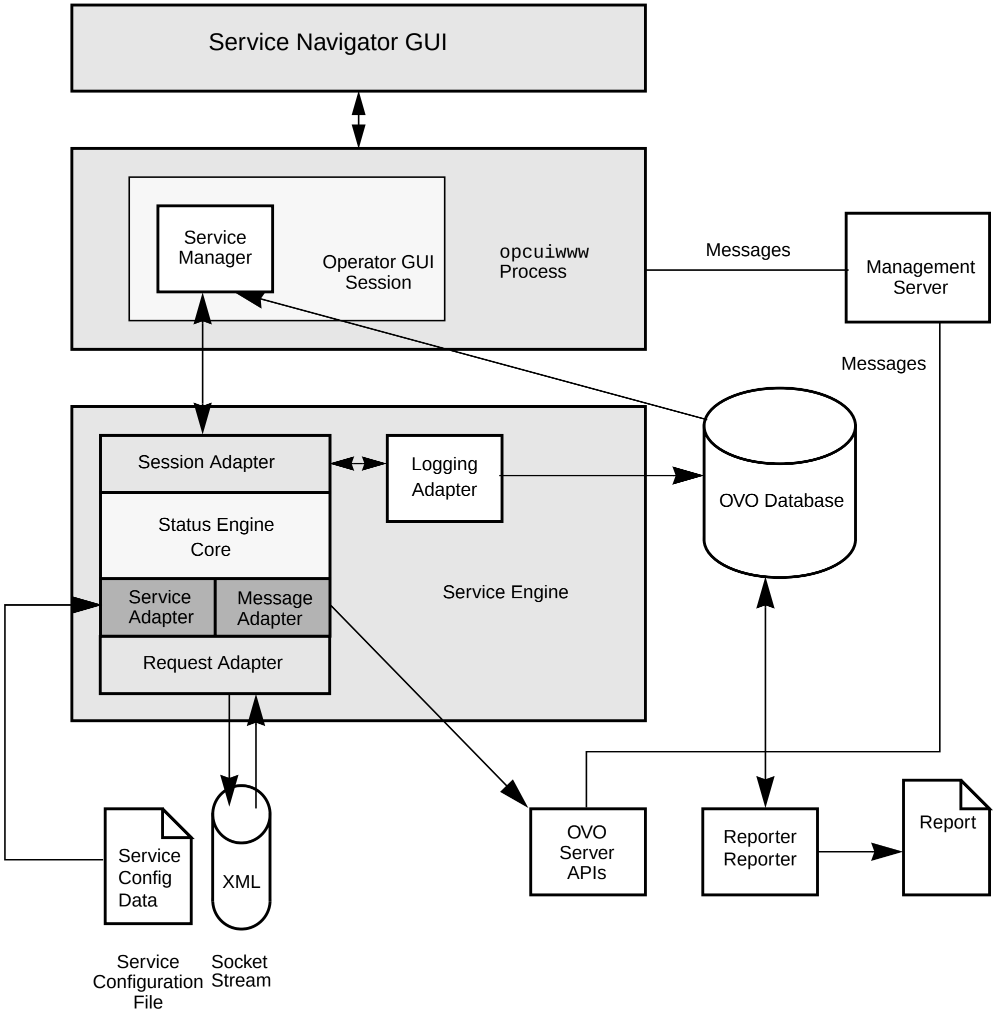
Use an ANSI C++ (aCC) compiler. See also the man page *opcsvc_api(3)* for more information.

The C++ APIs of the service engine are of special interest to integrators who, for example, want to integrate with trouble ticket systems.

The Service Navigator Architecture

The following figure gives an overview over the general architecture of the service engine of Service Navigator.

Figure 6-1 **The Service Navigator Global Architecture**



The service engine has the following components:

❏ The service adapter manipulates the service data (configuration tasks).

❏ The message adapter gets messages and message change events from the OVO management server.

❏ The session adapter performs operational tasks.

❏ The status engine core calculates the status and maintains the data structures.

❏ The logging adapter is responsible for persistent service logging.

❏ The request adapter handles operations from clients by contacting the service, session, and logging adapter.

The XML Data Interface

The XML Data Interface uses filesystem sockets as communication mechanism. The request adapter of the service engine binds to the socket and listens for requests. Each request is handled by a request handler in parallel. If a new request comes in, it opens a filesystem socket over which it communicates with the new client. The client writes the request into the socket after a successful connection.

Depending on the type of request, the client also provides information as XML text. Incoming requests comply to the `operations.dtd`, outgoing XML to the `results.dtd`. Depending on the request, the request adapter contacts the session, service, or logging adapter.

The following namespaces are used by the Service Navigator DTDs:

❏ XML namespace of the `service.dtd`:

  `http://www.hp.com/OV/opcsvc`

❏ XML namespace for the `operations.dtd`:

  `http://www.hp.com/OV/opcsvcoperations`

❏ XML namespace for the `results.dtd`:

  `http://www.hp.com/OV/opcsvcresults`

Name spaces are specified within the toplevel XML tag and are used to uniquely identify the XML tags. For example, a file `operations.xml` should start like this:

```
<?xml version=’1.0’ ?>
<Operations xmlns="http://www.hp.com/OV/opcsvcoperations"
version="1.0">
```

The DTDs are available in:

`/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/services.dtd`

`/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/operations.dtd`

`/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/results.dtd`

You can test your XML commands interactively using the `opcsvcterm` program. This is an interface to the service engine that inputs XML into stdin and outputs XML to stdout. See also the man page *opcsvcterm(1M)*.

See the *OVO Developer's Reference* for more information about the XML syntax.

The C++ APIs

The OVO Developer's Toolkit provides the following C++ interfaces for Service Navigator:

- ❑ **Service operations interface**

  This interface allows you to set or remove the service attributes, and to request the status and some basic properties of the service.

- ❑ **Registration interface for service status changes**

  This interface allows you to register for service status changes. The information that is returned includes the service name, the previous severity, and the new severity.

  This is of interest to integrators who want to react to service changes with appropriate actions, for example, forward the information to a trouble ticket system or notification service, or to execute other commands.

The following sections describe the concept of these interfaces in more detail. See the *OVO Developer's Reference* for more information about the C++ classes and for detailed examples.

The Service Operations Interface

This interface allows you to set or remove service attributes, and request the status and some basic properties of a service. For example, the label, description, icon, or background.

The Registration Interface for Service Status Changes

A client sends a registration request to the interface in the service engine which describes which events the client wishes to receive. The request is in XML format. The registration manager parses the XML registration structure and passes it to the core engine. The XML structure for the registration interface is defined in the `operations.dtd` (which also includes the `service.dtd`).

The service name identifies the client registrations in the service engine. The registration manager maintains the registration information for each client during runtime but it is the client's responsibility to react to shutdowns of the service engine, for example, if the socket is closed on the server side. The client has to re-register when the service engine is restarted.

When a client updates its registration information, the new registration information must be passed to the interface. The handler stores it in memory. It is the responsibility of the client to handle any additions or removals to/from the registration information. The registration manager initiates an update in the service engine core which propagates the service graph data structures according all the registrations.

The registration information has the form of a `<Registration>` XML structure. Clients pass this structure by way of an API function to the service engine.

The Registration Conditions

The registration conditions can register for services status changes if certain criteria are met. The following conditions are available:

❏ Register depending on service structure:

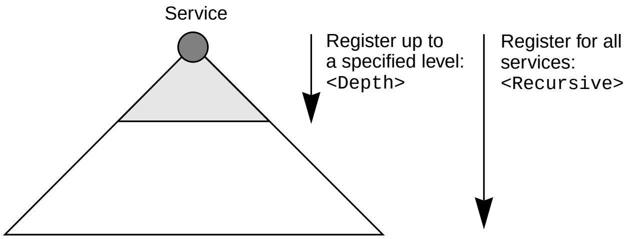
- Recursive: `<Recursive>`
- Depth of graph: `<Depth>`

❏ Register for all status change events:

- All events: `<All>`

See Figure 6-2 on page 246 for an illustration and examples.

Figure 6-2 Registration Condition for Service Structure


Service
Register up to a specified level: <Depth>
Register for all services: <Recursive>

For example:

```
<Registration>
   <RegCondition>
      <ServiceRef>svc_1</ServiceRef>
      <Recursive/>
   </RegCondition>
   <RegCondition>
      <ServiceRef>svc_2</ServiceRef>
      <Depth>3</Depth>
   </RegCondition>
</Registration>
```

For example:

```
<Registration>
   <All />
</Registration>
```

7 Integration Facilities of the HP OpenView NNM Core Platform

In This Chapter

This chapter describes the integration facilities of HP OpenView Network Node Manager, the core platform of the OVO Developer's Toolkit.

Overview

When you are working on an application that is to be integrated into the Network Node Manager Platform, it is useful to consider the following different data domains:

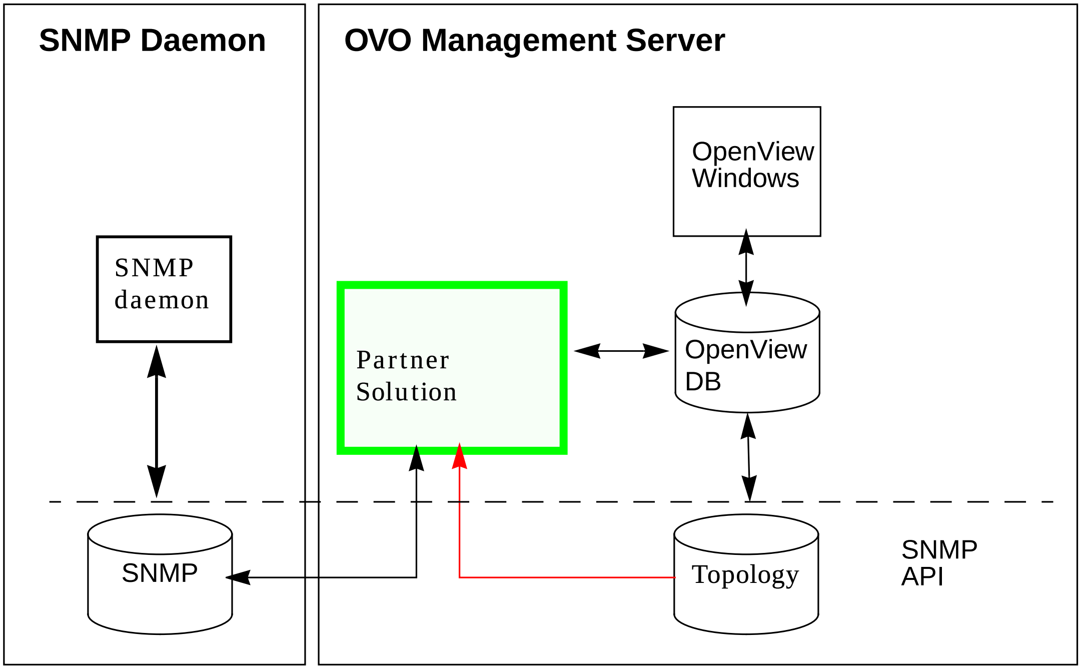
❑ SNMP MIB data

❑ Topology data

❑ OpenView object database

These domains are shown in Figure 7-1 and are described in the following sections.

Figure 7-1 Data Domains of the HP OpenView NNM Platform



The OpenView Windows API

This API enables an OVO partner to create and manipulate maps and submaps, and also to generate dialog for user interaction. It is intended for the development of applications that take advantage of these features.

The following two components enable developers can construct OpenView Windows applications:

❏ Application Registration Files (ARFs)

  Application registration files (ARFs) are configuration files that are read when OVW is started. Registration files can be used to perform many of the functions that would otherwise require programming. ARF files are described in the section “GUI Application Integration”.

❏ OpenView Windows Programming Libraries (OVw API)

  More complex applications require that you write programs using the OVw API. The OVw API gives you more control over the behavior of OpenView Windows applications. The OVw API consists of a large number of routines that let the programmer perform tasks related to the following functional areas:

  - Application Integration with OVW

    Several routines are available to integrate your application into OpenView Windows. Every application that uses the OVW API needs to use at least some portion of these calls. These routines allow your application to connect to OpenView Windows, to determine when users select particular menu items, and to handle errors.

  - Object Database Access

    There are over 40 OVw API routines that operate on objects and fields of the OV object database. These routines are used to create objects, to create the fields that comprise objects, to get or set field values in objects, and to relate fields to objects and vice versa. Fields can also be created using Field Registration Files.

    The object database and its access is discussed in more detail in “HP OpenView Data - Objects and Fields”.

- Map and Submap Routines

  The OVw API contains many routines that are used to operate on maps and submaps. These routines let you create and modify submaps, as well as retrieve information about maps and submaps. A few OVw API routines let you change the background images of submaps.

- Symbol Routines

  These routines create symbols, alter symbol behavior and appearance, and get information about an object as it exists on a map. Symbols can also be created using Symbol Registration Files.

  Maps, Submaps and Symbols are described in the section "Maps, Submaps and Symbols".

- User Verification

  Several OVw API routines allow a program to verify changes to maps and objects made by the user by way of the user interface.

- Dynamic registration

  There are over 60 OVw API routines that dynamically configure the OVW menu structure. These routines are rarely used by most application developers.

- Callback Routines

  OpenView Windows uses callback routines to communicate with applications when various events occur. OpenView Windows provides many definitions for callback routines to be provided by the developer

As this API is fully documented in a separate manual set and as it is not usually the primary integration capability used for OVO integration, it is not discussed here at the C-function level.

For more information about this API, see the following manuals:

❑ *OVW Developer's Guide*

❑ *OVW Application Style Guide*

Maps, Submaps, and Symbols

A map is a collection of OVW objects and their relationships. Users don't view maps directly - they view windows called submaps that display a subset of map information.

Users, not applications, create maps. Users can create several maps, and they can also view maps control which applications operate on the various maps. Whereas users create maps and define their scope, applications dynamically update maps to reflect the state of the management environment.

A submap is a collection of related symbols that are displayed in a single graphical window. Submaps essentially provide a view into the map object space. Each submap displays a different perspective of the information in the map, with the submaps typically organized in a hierarchical fashion. OpenView submaps provide several layout algorithms for symbols, including bus, star, ring, row/column, and point-to-point. Applications can also set or change the color images that are used as map backgrounds.

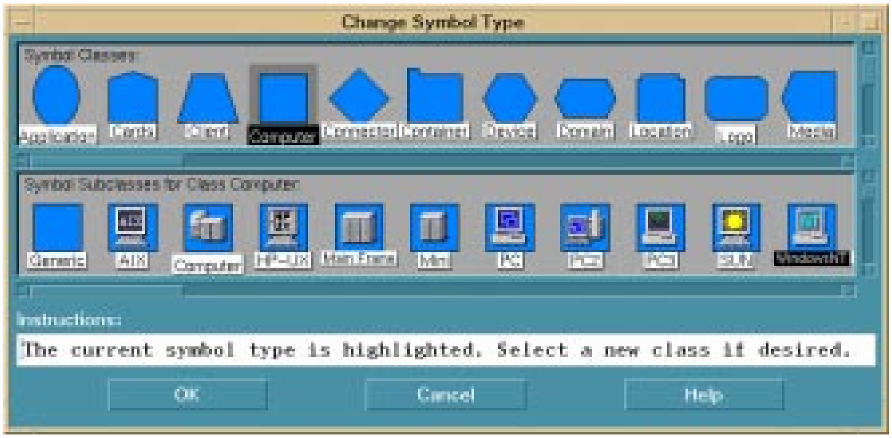
Symbols can be placed in OpenView submaps to represent objects in the OpenView database, such as nodes, connections, or agents. Symbols can also represent collection of objects, such as a subnets or connections. Each symbol is identified by its symbol type. The symbol type is defined by a symbol class/subclass pair. The symbol class defines the symbol category, while the symbol subclass defines a particular element within that class.

Symbols can be made executable. If an operator selects an executable symbol, a predefined action is started. Symbols can also change color to indicate status, e.g. green for normal, or red for critical. New symbols can be added by providing icon files and ASCII registration files.

OVO uses maps, submaps and symbols to represent IP networks. The hierarchy can be explored from the “root” (internet) level, down to the subnets, devices, and finally to agents.

OVO provides you with a rich set of predefined symbol classes and subclasses. Figure 7-2 on page 253 shows some of the symbol classes and subclasses provided by OVO.

Figure 7-2 **OVO Symbol Palette**



The current symbol type is highlighted. Select a new class if desired.

The HP OpenView programmer’s interface lets your application create symbols and submaps that show object hierarchies and connections (e.g., WAN connections). This is described in detail in the *OVW Developer’s Guide*.

Symbol Type Registration ASCII Files can also be used to define symbol classes and subclasses in OpenView Windows. The predefined OVW symbol classes and subclasses are themselves defined using various symbol type registration files. You are encouraged to use these predefined symbol classes and subclasses in your applications.

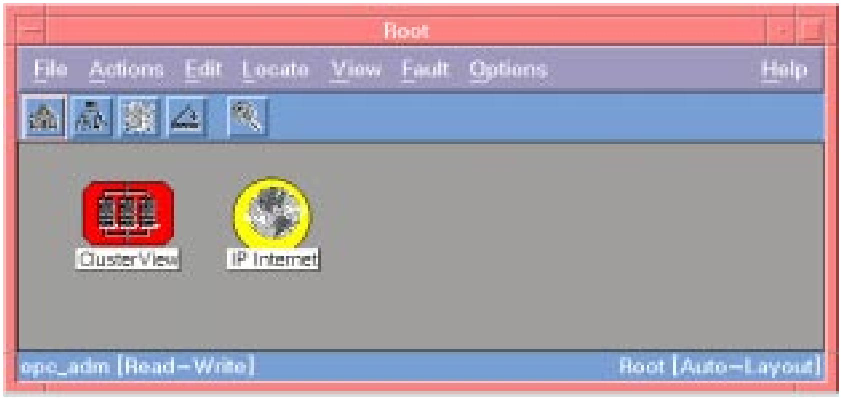
You may find, however, that existing classes and subclasses are not adequate for your needs. You can add new symbol subclasses to existing symbol classes or you can define your own symbol classes. Symbol Type Registration Files are described in detail in the OVW Developer’s Guide.

ClusterView: An Example of an Integrated Map Application

The ClusterView application is integrated into OpenView Network Node Manager and allows to display and monitor MC/ServiceGuard clusters from the HP OpenView environment in a distributed commercial environment. An MC/ServiceGuard cluster is a loosely defined collection of nodes that assure the availability of a common set of High Availability Services by using replicated hardware resources (CPUs, LAN cards, shared disks) and MC/ServiceGuard specific processes that monitor registered services.

The ClusterView Map-application clusmap uses the OpenView Windows API to create its own submaps and symbols that represent MC/ServiceGuard clusters. In the root-submap, it places an explodable "Clusters" Symbol next to the "IP Internet" symbol to allow the user to access the cluster hierarchy. Like the ipmap Map-application that sets the color of the symbols under the "IP Internet" symbol, the clusmap application sets the color of the symbols under the "Clusters" symbol according to the status of the MC/ServiceGuard elements.

Figure 7-3 **Root Submap Showing ClusterView Symbol**



The user can double click on the `Clusters` icon to reach the Clusters submap which contains a symbol for each discovered MC/ServiceGuard cluster. The cluster icons in this submap are automatically placed in this submap for all clusters auto-discovered by the ClusterView process clusmon. Status changes will be reflected by the color of the cluster icons. This screen can be used to drill down to get information on each cluster: a double click on the MC/ServiceGuard Cluster icons will bring the user to the "MC/ServiceGuard Cluster" submap.

The “MC/ServiceGuard Cluster” submap displays the nodes and packages configured in the cluster. In MC/ServiceGuard terminology, a package is a collection of network resources and high availability services managed together such that a single package can be moved between nodes within a cluster in order to make it highly available.

HP OpenView Data: Objects and Fields

The OpenView database is used by applications to store a wide range of information about network objects. An object is an internal representation of a logical or physical entity or resource that exists somewhere in a computer network. It consists of a unique object identifier and a set of fields that specify all the characteristics of the object.

OVO automatically makes topology information available in the OpenView database; it also adds new fields such as “IP Status”. This information is then available to management applications. OVO uses OpenView database information to generate network maps and hierarchies.

Other applications are able to read, write, and create OpenView database objects and fields. New fields can include information such as node serial numbers, trouble reports, or non-IP status. Access to this database is provided through the HP OpenView programmer’s interface.

The HP OpenView programmer’s interface provides almost 50 functions for interaction with the OpenView Windows object database. The API is not as complex as it might first appear, though. Among other things, it provides access to perform the following tasks:

- ❏ Field operations
  - Creating Fields

    Fields can be created either using the appropriate API routine or the Field Registration File (FRF). The FRF is the normal and preferred way to create fields.
  - Getting Field information for a single or all fields
  - Getting/Setting object field values
- ❏ Object operations
  - Creating an object
  - Deleting an object
- ❏ Convenience Routines

Fields are object attributes stored in the OVW database that can be seen as the building blocks from which objects are constructed. There are three components to a field definition:

❏ the field name

❏ the data type of the field

❏ flags that indicate how the field is used

The following table lists the available data types for field definitions:

Table 7-1 Data Types and Field Definitions

Data Type	Description	Example Field
Integer32	A 32-bit signed integer	IPMap Version, TopM Node Count
Boolean	This type can have the values True or False	isSNMPSupported, isLocation
String	A standard character string, limited to 256 characters in length	SNMP sysDescr, SNMP sysLocation
Enumeration	Declares the field to be an enumerated type. All possible values for the enumerated type are declared in a separate Enumeration statement.	SNMP ifType, vendor

While the field type specifies the data type of the field, the field flags specify how the field is treated by OpenView Windows. OVW provides the flexibility to treat fields in different ways. There are five types of flags (behavior) that can be applied to fields. The examples given behind a field flag item are taken from the Field Registration File:

❑ List

This field is a List of the specified type. Currently, the only support types for lists are strings and integers. For example, you might define a field for the names of administrators of your computer network. The type would be string, and the flag field would be set to list to allow you to store multiple names:

```
Field “administrators” {
  Type String;
  Flags list;
}
```

❑ Name

Name fields uniquely identify objects, i.e. there is only one object with a specific value for this field. Because the hostname of a computer would be unique for all nodes on the network, the name flag can be set for the field:

```
Field “IP Hostname” {
  Type String;
  Flags name;
}
```

❑ Locate

A locate operation may be done on the Locate field. This field will appear in the `Locate: By Attribute` dialog box. To search for, or locate all nodes with a particular status, you might use enumerated types to classify the status of nodes on a network and setting the locate flag:

```
Field “IP Status” {
  Type    Enumeration;
  Flags   locate;
  Enumeration           “Unset”,
                        “Unknown”,
                        “Normal”,
                        “Marginal”,
                        “Critical”,
                        “Unmanaged”,
                        “Warning”,
                        “Major”,
                        “Restricted”,
                        “Disabled”;
}
```

❑ General

This field will appear in the general attributes section of the Add and Describe boxes on an object.

❑ Capability

This field is a capability and is used to classify an object. Only booleans and enumerated types are supported as capability fields. To determine whether an object has computer capabilities the following field could be used:

```
Field “isComputer” {
  Type boolean;
  Flags capability;
}
```

Field Registration Files and the HP OpenView programmer’s interface are described in more detail in the *OVW Developer’s Guide*.

The OpenView SNMP API

The OpenView SNMP API provides the following functions:

❏ SNMP Communications API and Related Commands

A command interface and a programmer's interface to SNMP communication functions.

❏ SNMP Configuration API

An API for SNMP configuration purposes.

❏ Topology Data

It stores information about the network topology in a topology database.

Each component is described in a separate section.

The data on the SNMP agent is accessed by both the agent software and by management applications.

The layout of agent MIB data is specified by a MIB definition, very much as you would describe a programming data structure. Let's look at an example agent data structure, and components that are used when integrating with OVO.

Table 7-2 Example of SNMP Agent Data

MIB Variable	Value
system.sysObjectID.0	...hp.nm.system.hpux.hp9000s700
system.sysUpTime.0	111579040
system.sysContact.0	Jane Smith

The Object ID field is extremely important to OVO; it identifies the type of SNMP agent that is running on a node. OVO registration files can use this ID to specify the capabilities of this agent, and which icon will be used to represent it in HP OpenView maps.

The remaining fields contain agent status, configuration, and metrics. The read-write fields may be written from your integrated application (SNMP “sets”), or from OVO applications. All fields can be read from your application (SNMP “gets”), from the SNMP MIB Browser, or plotted over time by the xnmgraph application. Data may also be included in trap messages sent by the agent.

SNMP Communications API and Related Commands

There are two methods available to read or write agent MIB data:

- ❏ use the Simple Network Management Protocol Application Programming C-Interface (C functions of the SNMP-API)
- ❏ use the commands `snmpget`, `snmpset`, and `snmptrap`.

Available C-API Functions

This API allows to establish SNMP communication sessions for exchanging SNMP messages. The available functions and their functionality are listed in Table 1. The table should suffice to create an impression of what can be achieved using the functions of the API.

Formally a SNMP message is called a PDU (protocol data unit). Basically, there are functions for opening and closing SNMP sessions, for creating, and manipulating PDUs, for sending and receiving SNMP PDUs, and for manually retransmitting PDUs.

Most of the API functions are available in two flavors. One flavor is to be used in event driven X applications. Functions of this flavor have the letter “X” in their name. The other flavor is for use in conventional programs. Functions of those two flavors may not be mixed in one program.

For more information on this API, seethe corresponding man pages and the *SNMP Developer’s Guide* and Reference. This manual provides a good introduction to the underlying concepts, a description of the corresponding data structures, and scenarios for proper use of the functions.

Table 7-3 SNMP Communications API Functions

Category	Function Name for Event-Driven X Applications	Function Name for Traditional Applications	Description
Session Management	OVsnmpXOpen	OVsnmpOpen	Establishes an active SNMP session.
	OVsnmpXClose	OVsnmpClose	Terminates an active SNMP session, and frees associated resources.
Event Framework Session Functions	OVsnmpCreatePdu	OVsnmpEventOpen	Successor to OVsnmp[X]TrapOpen(). Establishes a logical session with the OVsnmpAPI for the purpose of receiving SNMP traps via the OpenView Event Framework. It allows filters to be applied to events to reduce the number of events that are received.
	OVsnmpXTrapOpen	OVsnmpTrapOpen	Predecessor to the OVsnmp[X]EventOpen() function. This function is provided for backward compatibility only. Use the OVsnmp[X]EventOpen() routine in the future.

Table 7-3 SNMP Communications API Functions (Continued)

Category	Function Name for Event-Driven X Applications	Function Name for Traditional Applications	Description
Message Setup and Manipulation	OVsnmpCreatePdu		Allocates an SNMP Protocol Data Unit (PDU) data structure. A PDU contains an SNMP message.
	OVsnmpAddNullVarBind, OVsnmpAddTypedVarBind		Used in conjunction with SNMP Set, Get, GetNext and SNMPv2 Inform Requests. Allocates, initializes and appends an OVsnmpVarBind structure for the specified SNMP MIB object to the SNMP Request PDU.
	OVsnmpFixPdu		If, in a list of variables, one or more variables cause a request to fail (for any reason), OVsnmpFixPdu(3) can be used to strip the offending variable(s) from the list. The result is a list that can be used to retry the request.
	OVsnmpCopyPdu		Creates a copy of the specified OVsnmpPdu data structure and its contained elements
	OVsnmpFreePdu		Deallocates a PDU structure, and recovers associated resources.

Table 7-3 SNMP Communications API Functions (Continued)

Category	Function Name for Event-Driven X Applications	Function Name for Traditional Applications	Description
Communications	OVsnmpXSend	OVsnmpSend	Sends an SNMP message in non-blocking mode. Resources associated with the PDU are deallocated with this call, unless an error occurs. Using OVsnmpXSend(3) in an event-driven X-based application achieves automatic non-blocking transmission.
	OVsnmpBlockingSend		Sends an SNMP message in blocking mode. Resources associated with the PDU are deallocated with this call, unless an error occurs.
	OVsnmpRecv		Sends an SNMP message in blocking mode. Resources associated with the PDU are deallocated with this call, unless an error occurs.
	OVsnmpRead		Receives messages on all active SNMP sessions. Returns information via the callback function.
Manual Retransmission	OVsnmpGetRetryInfo		Gets retransmission information on pending SNMP requests.
	OVsnmpDoRetry		Retransmits a pending SNMP request.

Related Commands

The following commands related to SNMP communication are available:

- ❏ `snmpwalk(1)` - query a node repeatedly using SNMP GetNext requests
- ❏ `snmpnext(1)` - query a node using SNMP GetNext request
- ❏ `snmpset(1)` - issue an SNMP Set request
- ❏ `snmpget(1)` - query a node using SNMP Get request
- ❏ `snmptrap(1)` - issue an SNMP Trap

For more details, see the corresponding man pages.

SNMP Configuration API

The SNMP Configuration API provides programmer access to the SNMP Configuration Database. This database is the repository on the management station for configuration information that controls the behavior of a SNMP Session. There are three classes of configuration parameters that are stored in the database:

- ❏ Configuration parameters for specific managed nodes
- ❏ Configuration parameters for wildcard IP addresses
- ❏ Configuration parameters for a global default

Whenever the configuration parameters for a managed node are requested, they are obtained from these three classes of stored information. The resulting parameters, and the associated IP address of the destination can be cached so that subsequent lookups will be performed more quickly.

The available functions and their functionality are listed in Table 2. The table should suffice to create an impression of what can be achieved using the functions of the API.

For more information on this API, please refer to the corresponding man pages and to the *SNMP Developer's Guide*. This manual provides a good introduction to the underlying concepts, a description of the corresponding data structures, and scenarios for proper use of the functions.

Table 7-4 SNMP Communications API Functions

Category	Function Name	Description
Database Access	OVsnmpConfOpen	Open the SNMP configuration database for subsequent access.
	OVsnmpConfClose	Close the SNMP configuration database.
	OVsnmpConfDbName	Obtain the pathname of the SNMP configuration database.
	OVsnmpConfFileName	Obtain the pathname of the backwards compatibility (shadow) file that is associated with the SNMP configuration database.
Resolution	OVsnmpConfResolveDest	Obtain the effective configuration parameters for a specified destination agent.
	OVsnmpConfFreeDest	Deallocate all memory associated with the OVsnmpConfDest structure returned by OVsnmpConfResolveDest(3).
Editing	OVsnmpConfReadNextEntry	Sequentially read specific node records from the SNMP configuration database.
	OVsnmpConfReadEntry	Read a specific node record from the SNMP configuration database.
	OVsnmpConfCreateEntry	Create a new specific node record in the SNMP configuration database. This operation will fail if a record already exists for the specified node.
	OVsnmpConfStoreEntry	Create or replace, as necessary, a specific node record in the SNMP configuration database.
	OVsnmpConfDeleteEntry	Delete a specific node record from the SNMP configuration database.
	OVsnmpConfAllocEntry	Allocate an OVsnmpConfEntry structure.
	OVsnmpConfCopyEntry	Allocate and initialize an OVsnmpConfEntry structure with the values from the specified OVsnmpConfEntry structure.

Table 7-4 SNMP Communications API Functions (Continued)

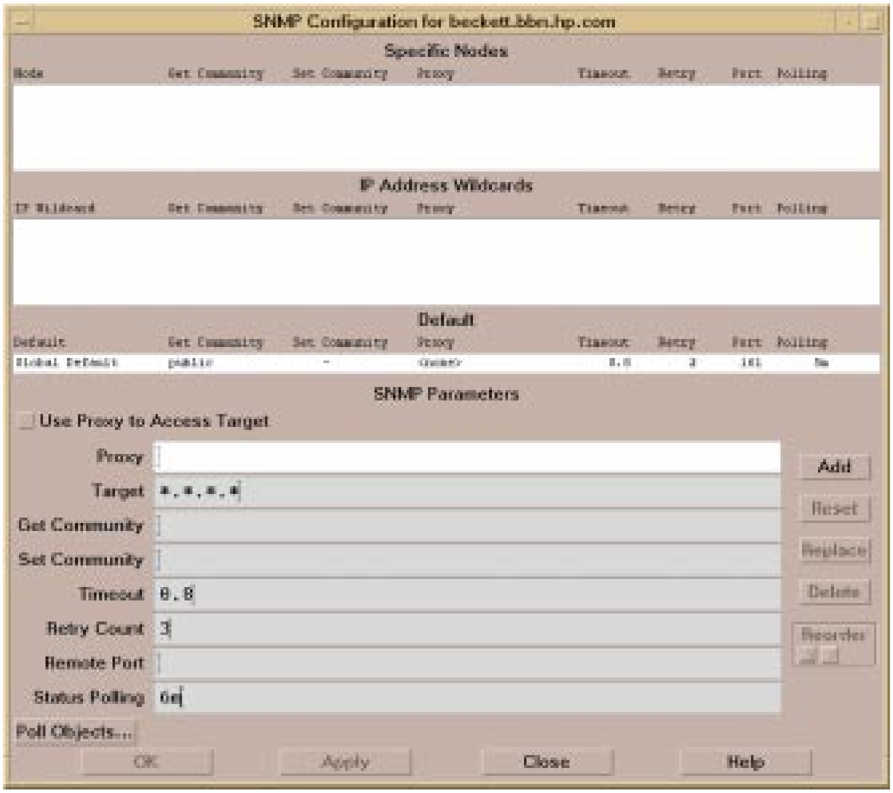
Category	Function Name	Description
Editing (cont.)	OVsnmpConfParseEntry	Allocate and initialize an OVsnmpConfEntry structure with the values extracted from a configuration string in the format of an ovsnmp.conf(4) configuration file entry.
	OVsnmpConfFreeEntry	Deallocate all memory associated with the specified OVsnmpConfEntry structure.
	OVsnmpConfReadWcList	Read the list of IP address wildcard records from the SNMP configuration database. Since the semantics of wildcard records are partly dependent upon the order of the records in the database, these records can only be updated as a single list.
	OVsnmpConfStoreWcList	Create or replace, as necessary, the list of IP address wildcard records in the SNMP configuration database.
	OVsnmpConfAllocWcList	Allocate memory for an OVsnmpConfWcList structure. This structure may then be inserted into the wildcard list obtained using OVsnmpConfReadWcList(3).
	OVsnmpConfFreeWcList	Deallocate all memory associated with the specified list of OVsnmpConfWcList structures.
	OVsnmpConfReadDefault	Read the global default record from the SNMP configuration database.
	OVsnmpConfStoreDefault	Create or replace, as necessary, the global default record in the SNMP configuration database.
	OVsnmpConfImportFile	Replace the contents of the SNMP configuration database with records extracted from configuration strings in the specified ovsnmp.conf(4) configuration file.
	OvsnmpConfExportFile	Dump the contents of the SNMP configuration database to the specified ovsnmp.conf(4) configuration file.

Table 7-4 SNMP Communications API Functions (Continued)

Category	Function Name	Description
Admini-stration	OVsnmpConfReadCntl	Read the current SNMP configuration database administration record. This record contains options which control run-time caching and use of the backward compatibility configuration file.
	OVsnmpConfStoreCntl	Update the current SNMP configuration database administration record. This enables you to modify the options which control run-time caching and use of the backward compatibility file.
Misc.	OVsnmpConfDeleteCache	Remove all entries from the SNMP configuration run-time cache. New entries accumulate in the cache as applications call OVsnmpOpen(3) and OVsnmpConfResolveDest(3).
	OVsnmpConfReadNextDest	Sequentially read entries from the SNMP configuration run-time cache. This function is provided for troubleshooting only (i.e., to dump the contents of the cache for inspection.)
	OVsnmpConfReadNextDest	Print the contents of an OVsnmpConfDest structure to stdout.
	OVsnmpConfPrintEntry	Print the contents of an OVsnmpConfEntry structure to stdout.
	OVsnmpConfPrintCntl	Print the contents of an OVsnmpConfCntl structure to stdout.

The SNMP configuration can also be done interactive with the tool xnmsnmpconf.

Figure 7-4 **SNMP Configuration Using xnmsnmpconf**



Node	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling

IP Wildcard	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling

Default	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling
Global Default	public	-	<none>	0.8	3	161	6m

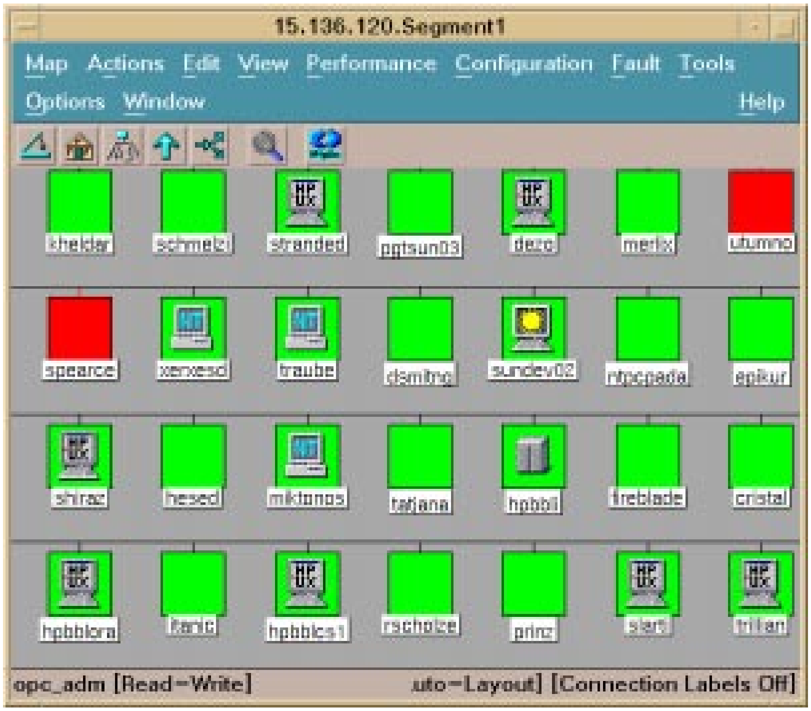
□ Use Proxy to Access Target

Proxy:
Target: *.*.*.*
Get Community:
Set Community:
Timeout: 0.8
Retry Count: 3
Remote Port:
Status Polling: 6m

Topology Data

OVO provides a topology manager that automatically discovers and maintains a database of TCP/IP nodes and connections. This database includes fields that describe each node's functionality, address, and status. If a device has an SNMP agent, then that agent's identification is used to determine its capabilities and how it should be represented on OVO maps.

Figure 7-5 TCP/IP Submap



The topology manager passes information to the OpenView database. Other applications may provide discovery mechanisms for non-TCP/IP protocols; for example, they may discover connections using MAC, WAN, or vendor-specific protocols. As with the OVO topology manager, these applications can store topology information in the OpenView database.

OVO can be configured to use the Oracle relational database to manage topology information. Some customers use the query and report-generating capabilities of this database to analyze and document their network.

8 Creating and Distributing an Integration Package

In This Chapter

The final stage of the integration process is to create an integration package for distribution to customers. You will find that the configuration download/upload utility of OVO will play an important role in the creation of the integration package.

This chapter describes the structure of OVO configuration files, explains the use of application registration files (ARFs), and describes how to download and upload configuration information, and how to add programs/scripts for distribution by OVO.

To create an integration package you will need to do the following tasks:

1. Define the required configuration in the environment of the integration package.

   - You will do most of this task in the OVO administrator's GUI. For example, define new templates, or add new conditions to existing templates; define message groups, node groups, default operator(s); set up applications in the Application Desktop, etc.

   - In addition, it may sometimes be necessary to place files at predefined locations in the file trees maintained by OVO, for example, if monitor scripts, or programs for performing certain actions are to be distributed by OVO.

2. Download the configuration information into a file tree.

   The OVO administrator's GUI offers a powerful facility for downloading configuration information. This facility enables you to choose the full OVO configuration or selected parts that are relevant to the integration package.

   You can also use the command `opccfgdwn(1M)` to download predefined configuration sets from the command line.

3. Package the file tree, add any additional software if required and then ship the integration bundle.

   Additional software might be executables for processes running on the OVO management server, for example, if APIs on the management server are used, installation scripts, man pages, etc.

The OVO configuration upload command `opccfgupld(1M)` can be used at the customer's site to upload the configuration information into the local OVO installation.

Structure of OVO Configuration Files

The behavior and capabilities of OVO are determined by configuration information stored in a relational database. This information can be downloaded into a tree structure of flat files. The structure of this file tree is shown in Figure 8-1.

Figure 8-1 Structure of an OVO Configuration File Tree

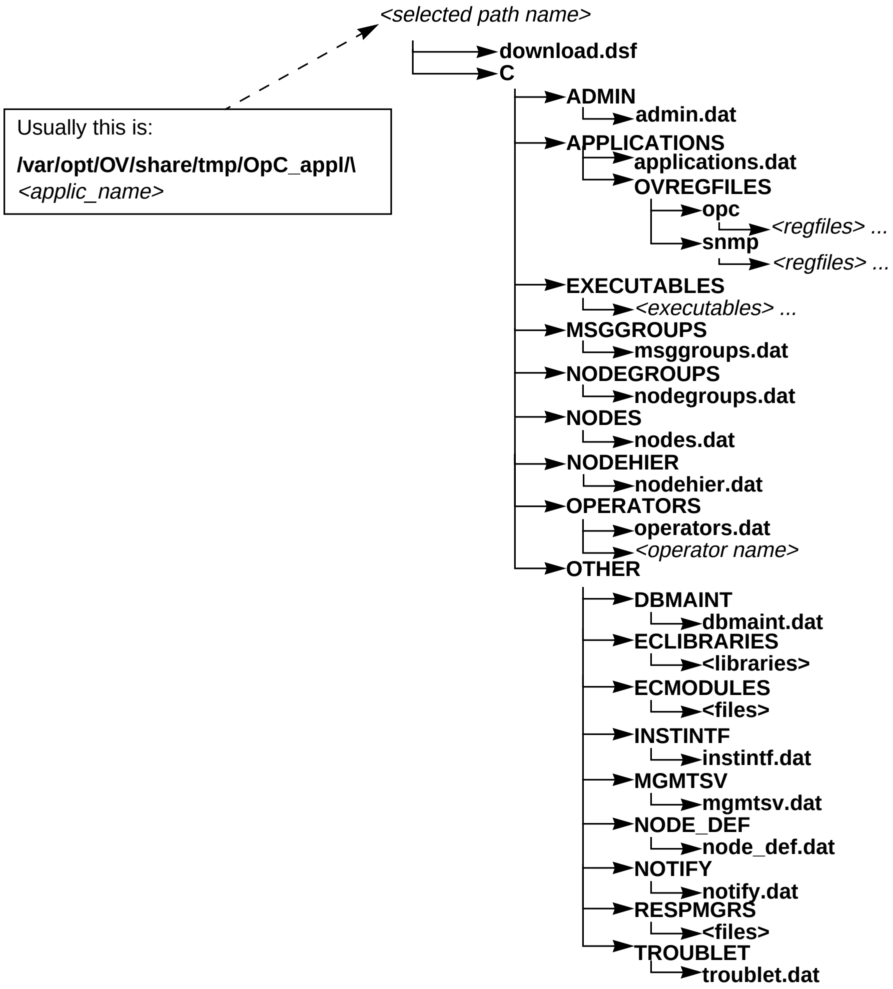
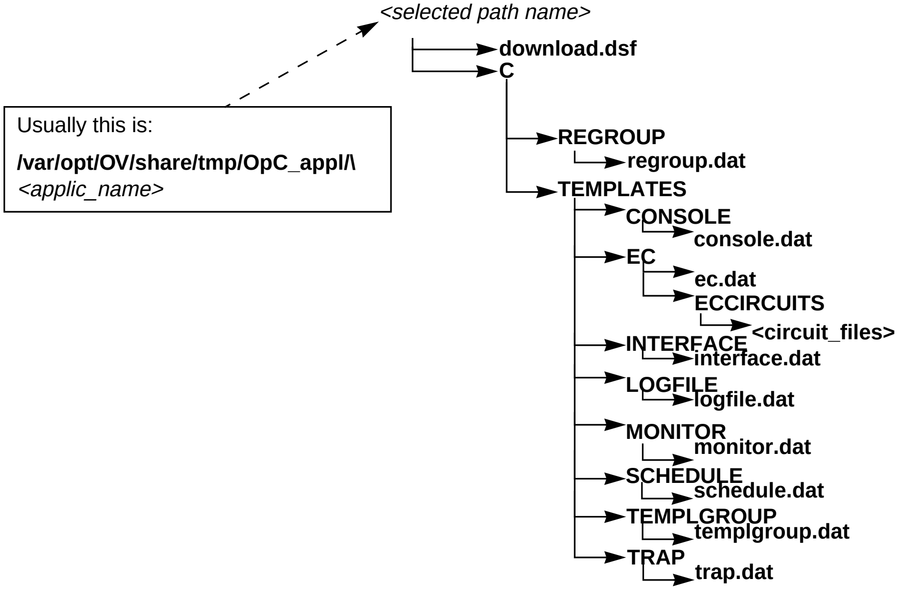


Figure 8-2 Structure of an OVO Configuration File Tree (cont.)



Downloading Configuration Information

You can download configuration information either by using the OVO administrator’s GUI, or from the command line using the `opccfgdwn(1M)` command.

Both methods enable you to select the parts of the configuration that you want to download. For example, instead of downloading the entire configuration, you may choose to download only the templates. The different parts of the configuration to be downloaded are specified in the following file:

```
/var/opt/OV/share/tmp/OpC_appl/cfgdwn/download.dsf
```

The following is an example extract from a `download.dsf` file:

```
APPLICATION_BANK;
NODE_GROUP “net_devices”;
OPERATOR “itop”
CONFIGURATION *;
OPERATOR “netop”
CONFIGURATION *;
LOGFILE_TEMPLATE “dflt_ApplEvLog (NT)” ;
LOGFILE_TEMPLATE “dflt_SecEvLog (NT)” ;
LOGFILE_TEMPLATE “dflt_SysEvLog (NT)” ;
```

This specification file is required as a parameter by the `opccfgdwn(1M)` command.

To download configuration data:

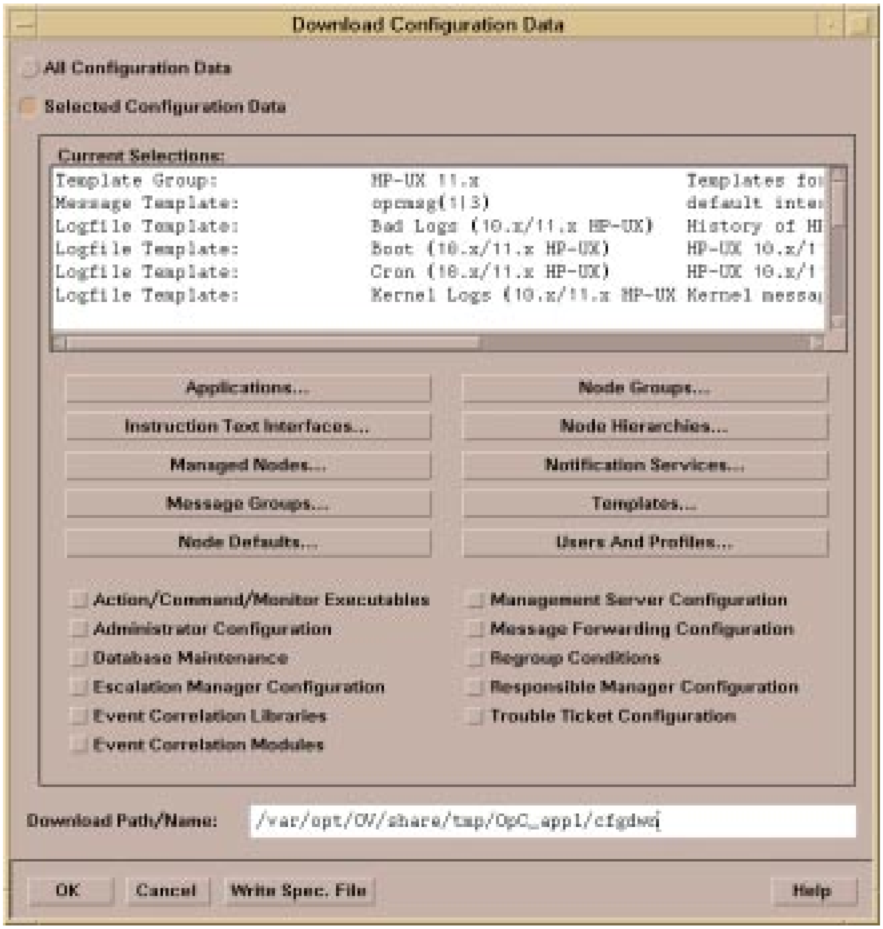
1. Open the `Download Configuration Data` window in the OVO administrator's GUI.

   Select `Actions: Server->Download Configuration`

2. Select the parts of the OVO configuration that you want to download.

   Figure 8-3 shows the `Download Configuration Data` window.

Figure 8-3 Download Configuration Data Window



Download Configuration Data

○ All Configuration Data

● Selected Configuration Data

Current Selections:

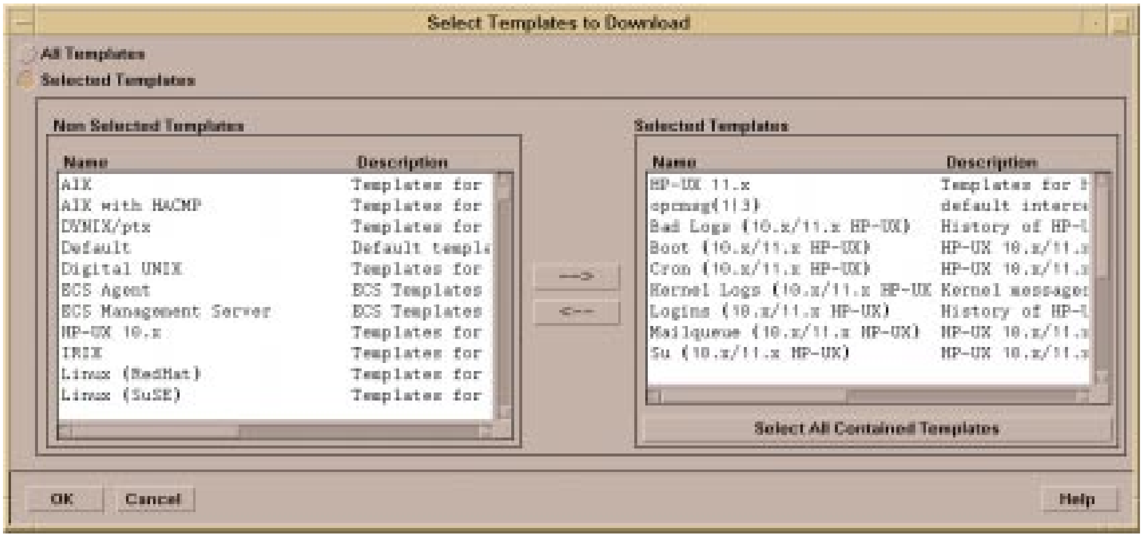
```
Template Group:            HP-UX 11.x                  Templates fo
Message Template:          opcmsg(1|3)                 default inte
Logfile Template:          Bad Logs (10.x/11.x HP-UX)  History of HI
Logfile Template:          Boot (10.x/11.x HP-UX)      HP-UX 10.x/1
Logfile Template:          Cron (10.x/11.x HP-UX)      HP-UX 10.x/1
Logfile Template:          Kernel Logs (10.x/11.x HP-UX Kernel messa
```

Applications...
Node Groups...
Instruction Text Interfaces...
Node Hierarchies...
Managed Nodes...
Notification Services...
Message Groups...
Templates...
Node Defaults...
Users And Profiles...
□ Action/Command/Monitor Executables
□ Administrator Configuration
□ Database Maintenance
□ Escalation Manager Configuration
□ Event Correlation Libraries
□ Event Correlation Modules
□ Management Server Configuration
□ Message Forwarding Configuration
□ Regroup Conditions
□ Responsible Manager Configuration
□ Trouble Ticket Configuration

Download Path/Name: /var/opt/OV/share/tmp/OpC_appl/cfgdwn

3. If you click on any of the buttons between `[Applications…]` and `[Users and Profiles…]` a second window opens that lets you choose the parts of the configuration to be downloaded more precisely.

Figure 8-4 Select Templates to Download Window



○ All Templates
● Selected Templates

Non Selected Templates

Name	Description
AIX	Templates for
AIX with HACMP	Templates for
DYNIX/ptx	Templates for
Default	Default templa
Digital UNIX	Templates for
ECS Agent	ECS Templates
ECS Management Server	ECS Templates
HP-UX 10.x	Templates for
IRIX	Templates for
Linux (RedHat)	Templates for
Linux (SuSE)	Templates for

Selected Templates

Name	Description
HP-UX 11.x	Templates for H
opcmsg(1|3)	default interce
Bad Logs (10.x/11.x HP-UX)	History of HP-U
Boot (10.x/11.x HP-UX)	HP-UX 10.x/11.x
Cron (10.x/11.x HP-UX)	HP-UX 10.x/11.x
Kernel Logs (10.x/11.x HP-UX	Kernel message:
Logins (10.x/11.x HP-UX)	History of HP-U
Mailqueue (10.x/11.x HP-UX)	HP-UX 10.x/11.x
Su (10.x/11.x HP-UX)	HP-UX 10.x/11.x

4. Clicking on the button `[Write Spec. File]` lets you write a specification file that you can use for downloading the configuration data at a later point in time. The `opccfgdwn(1M)` command requires the name of a specification file and a directory name under which to write the file tree.

5. Click `[OK]` to start the downloading process.

For example, assume that you want to download a configuration file tree into the directory:

```
/var/opt/OV/share/tmp/OpC_appl/newConf
```

The name of the specification file is `download.dsf`, and the download command expects that the directory contains a subdirectory `C`, for the English version of OVO, and that the specification file resides there. The full pathname of the configuration file must be:

```
/var/opt/OV/share/tmp/OpC_appl/newConf/C/download.dsf
```

To keep the output of the example short, a specification file was used that downloads two templates only.

Enter:

```
/opt/OV/bin/OpC/opccfgdwn download.dsf \
/var/opt/OV/share/tmp/OpC_appl/newConf
```

You will see the following messages displayed:

```
  verifying target filetree
  parsing download specification file
  “/var/opt/OV/share/tmp/OpC_appl/newConf/C/\
  download.dsf”
  starting download of selected configuration data
  reading logfile templates from DB
  downloading template “Bad Logs (HP-UX standalone)”
  reading interface templates from DB
  downloading template “opcmsg(1|3)”
  opccfgdwn finished

Warning:
         Since not “All Configuration Data” was selected
         for the download, the upload may result in
         unexpected database contents. Try “man 1m
         opccfgupld” for details.
```

The administrator can now upload configuration information into the OVO internal database at any time, and independent of the GUI, using the command `opccfgupld(1M)`.

Preparing to Download: Adding Executables

If the integration package contains threshold monitors, automatic actions, or operator-initiated actions to be performed on the managed nodes, you must make sure that the OVO agents can find the required executables (programs or scripts).

❑ Check whether the program/script already exists on the managed node, for example, if it is part of the OS or part of a monitored application.

❑ Use OVO to distribute the program/script to the managed node(s)

OVO will distribute the program/script to directories on the managed node which are maintained by OVO and which are automatically in the command search path of the OVO agents.

This section explains how to use OVO to distribute programs/scripts.

NOTE Distribution by OVO may only be used for small executables, preferably small wrapper scripts for the reasons stated below.

Prepare an executable for distribution by OVO as follows:

1. Place the executable into a source directory from which OVO will fetch it during the configuration downloading. On the OVO management server, use the directory:

   ```
   /var/opt/OV/share/databases/OpC/mgd_node/customer
   ```

   A 3-level directory hierarchy below this directory is predefined according to vendor, platform, and OS. Each directory below this triplet contains subdirectories for monitors, commands, actions, and package types. The package type is the communication type used by remote procedure calls (RPCs) of a particular agent platform.

   If your executables depend on the communication type, that is on NCS_RPC, on RPC_DCE_TCP, or on RPC_DCE_UDP, place them in the directories below the package type level. If they are independent of the communication type, place them in the appropriate subdirectory below the triplet.

   For example, a monitoring executable suitable for 10.x agents on s700 and s800 machines and using the RPC_DCE_TCP communication type, needs to be placed into the following subdirectories:

   ```
   /var/opt/OV/share/databases/OpC/mgd_node/customer

        ./hp/s700/hp-ux10/RPC_DCE_TCP/monitor

        ./hp/s800/hp-ux10/RPC_DCE_TCP/monitor
   ```

2. When choosing the configuration information to be downloaded, make sure that your executables are included.

   This has the effect that when you download the configuration information, OVO fetches the executables from the directory where you put them and includes them in the file tree with the configuration information.

3. Create the integration package including the configuration information and ship to the customer.

4. Upload the configuration information at the customer's site.

   This causes OVO to copy the executables into an OVO-maintained directory on the customer's management server.

5. Advise the OVO administrator, for example, in the documentation of the integration package, to distribute the new configuration information to the required managed nodes.

   In the `Install/Update OVO Software and Configuration` window, the administrator must distribute `Actions`, `Monitors` or `Commands`, depending on the requirements of the integration.

Warnings

It is important to consider the following issues when distributing executables using OVO:

❏ Names of executable files must not exceed a length of 12 characters.

❏ Names of executables must be unique; the names must not conflict with executables in other integration packages.

To achieve this, it is recommended to use the naming conventions that are part of the certification requirements for OVO integration packages. These naming conventions require integrators to prefix object names with a unique string to identify their solution.

❏ Only small executable files may be distributed by OVO

Distribution to managed nodes depends on the vendor, platform, and OS characteristics of the managed node and not on the templates or monitors distributed to the managed node.

For example, you might provide a monitoring script for a business application running on HP-UX 10.x machines. If you put the script into the corresponding directory, download it, and then upload it onto the customer's management server, this script will be distributed to any s800 machine running HP-UX 10.x, provided the administrator distributes “Monitors” to these nodes. This will happen even if the managed node doesn't run any part of the business application meaning that the monitor will never be used.

Uploading Configuration Information

The `opccfgupld(1M)` command can only be issued by user root, and you must specify at least the name of the directory under which the configuration information is stored.

You can use the following command line options depending on whether you want to overwrite existing information in the OVO database or not:

-replace Overwrite any information that already exists in the OVO internal database.

-add Insert additional information that is different from information that already exists in the OVO internal database.

Other options are available so that you can upload information at a subentity level, see below for details.

To upload configuration information into the OVO internal database:

1. Stop the OVO management server processes using the administrator's GUI. Select `Actions: Server->Stop Services` from the Node Bank menu.
2. Exit the administrator's GUI.
3. Upload the configuration information, enter:

   ```
   /opt/OV/bin/OpC/opccfgupld <upload_options> \
   <config_directory>
   ```

   This command is discussed in more detail in the examples below.
4. After successfully uploading the configuration information, restart the OVO management server processes:

   ```
   /opt/OV/bin/OpC/opcsv -start
   ```
5. Restart the OVO administrator's GUI.
6. Install the new configuration information on the managed nodes. Select `Actions: Agents->Install/Update SW & Config`.

The following examples demonstrate some of the functionality of the `opccfgupld(1M)` command. For more information, see the man page *opccfgupld(1M)*.

These examples assume that a file tree containing configuration information has already been created, using either the OVO administrator's GUI or the `opccfgdwn(1M)` command. The configuration file tree is assumed to be in the directory:

```
/var/opt/OV/share/tmp/OpC_appl/newConf
```

Example 1: Uploading in Add Mode (Default)

This is the most straightforward way to use `opccfgupld(1M)`. The entire configuration information under the directory `/var/opt/OV/share/tmp/OpC_appl/newConf` is uploaded into the OVO internal database. As the Add mode is the default setting, you do not need to specify the `-add` option. In this mode, only the entities that are not already stored in the OVO database are uploaded from the configuration file tree.

To keep the example output short, the file tree only contains configuration information for two templates. To demonstrate the effect of "Add" mode, the example shows what happens when one of the templates already exists in the OVO internal database. You will see that a warning message is generated for the existing template, and the new template is added.

To upload a configuration file in "Add" mode, enter:

```
/opt/OV/bin/OpC/opccfgupld /var/opt/OV/share/tmp/\
OpC_appl/newConf
```

You will see the following messages displayed:

```
  parsing index file
  "/var/opt/OV/share/tmp/OpC_appl/newConf/C/newConf.idx"
  starting upload
    uploading logfile templates
  Template Bad Logs (HP-UX standalone) already exists
(OpC50-79)
  Warning: not all requested objects were processed.
(OpC50-24)
    uploading interface templates
  opccfgupld terminated with 2 warning(s)/error(s)
```

Example 2: Uploading in Replace Mode

This example is similar to Example 1, except that Replace mode is used. You will see that, even though one of the templates already exists in the OVO database, it is overwritten by the upload command and no warning is issued.

To upload a configuration in Replace mode, enter:

```
/opt/OV/bin/OpC/opccfgupld -replace \
/var/opt/OV/share/tmp/OpC_appl/newConf
```

You will see the following messages displayed:

```
parsing index file
"/var/opt/OV/share/tmp/OpC_appl/newConf/C/newConf.idx"
starting upload
uploading logfile templates
uploading interface templates
opccfgupld finished
```

Example 3: Uploading and Replacing Information at a Subentity Level

The previous examples showed how to upload complete entities of configuration information, with these entities being complete templates. You may, however, prefer to work on subentities instead of on complete entities. For example, you may want to add conditions to a template.

You can merge configuration information using the options: -add -subentity. If you can identify a match condition in the default SNMP trap template, before or after which the new trap conditions should be inserted, the existing condition can be used as an anchor to control where opccfgupld(1M) inserts the new conditions.

Assume that the default template for OVO message interception contains the following conditions:

```
condition 1
condition 2
condition 3
```

and that the partner solution needs the following additional conditions:

```
new condition a
new condition b
```

If the configuration file tree contains a template for OVO message interception containing the two new conditions above, enter:

```
/opt/OV/bin/OpC/opccfgupld -add -subentity \
/var/opt/OV/share/tmp/OpC_appl/newConf
```

This command merges the two templates, so that the new conditions are appended to the end of existing template. The resulting template contains the following conditions:

```
condition 1
condition 2
condition 3
new condition a
new condition b
```

However, the sequence of conditions within a template is important, because the first condition that matches is applied. OVO, therefore, lets you insert new conditions at specific positions within the list of conditions.

Assume that, after analyzing the default template for intercepting OVO messages, you decide to insert the two new conditions after “condition 2” of the existing template, as follows:

```
condition 1
condition 2
new condition a
new condition b
condition 3
```

You can obtain this sequence of conditions as follows:

1. Copy the OVO default template for message interception, containing the following conditions:

   ```
   condition 1
   condition 2
   condition 3
   ```

2. Remove all conditions except “condition 2”.

3. Add the new conditions needed to integrate the partner solution. You will now have the following sequence of conditions:

   ```
   condition 2
   new condition a
   new condition b
   ```

4. Download the required configuration information, including at least the new template that you have just created.

5. Stop the OVO server processes and exit the GUI

6. Upload the configuration information, enter:

   ```
   /opt/OV/bin/OpC/opccfgupld -add -subentity \
   /var/opt/OV/share/tmp/OpC_appl/newConf
   ```

   When you specify the `-subentity` option, and OVO adds information to an existing template, it compares the conditions to be added with the existing conditions. If OVO finds a condition in the uploaded template that is contained in the existing template, this condition is used as an anchor to determine where to add the new information.

7. Restart the OVO server processes.

8. Restart the OVO administrator’s GUI.

9. Open the template for `opcmsg(1|3)` interception. You will see that the conditions are listed in the following order:

   ```
   condition 1
   condition 2
   new condition a
   new condition b
   condition 3
   ```

A Syntax Used in OVO Configuration Files

In This Chapter

This appendix provides a detailed description of the syntax used by the configuration download command `opccfgdwn(1M)` to store OVO configuration information in flat files which in turn are required by the configuration upload command `opccfgupld(1M)`.

Notation Used

In this appendix, the syntax is described using a BNF grammar. Keywords are written in boldface, non-terminal symbols are written in italics.

The symbol “ε” represents an empty string

The symbol “|” separates alternatives of which one is to be selected.

Square brackets “[” and “]” are used for grouping parts of a rule.

General OVO Syntax Rules

The following syntax rules apply to all OVO configuration files:

#	In the first column indicates a comment; all text until the new line is treated as comment.
separators	Blanks, tabs, new lines
keywords	Individual keywords are used, all in uppercase.
strings	All strings must be enclosed in quotes (“string”). Quotes within a string must be preceded by a backslash (\). Empty strings are represented by two quotes “”. To specify a backslash in a string use \\.

Configuration Files for Templates

This section describes the syntax used in configuration files that describe templates for the following message sources: logfiles, SNMP traps, MPE/iX console messages, and messages passed to OVO by the message interface `opcmsg(1|3)`.

In the following grammar examples, you will find that the configuration file can have the keyword `SYNTAX_VERSION` followed by a number at the beginning.

NOTE It is recommended to put the keyword `SYNTAX_VERSION` followed by the number of the current version at the beginning of a configuration file.

For reasons of backward compatibility, the `SYNTAX_VERSION` is dynamically determined at template distribution time:

Table A-1 **Template Syntax Versions**

Syntax Version	OVO Version	Keyword	Description
8	A.07.00	`CUSTOM`	Keyword for custom message attributes.
7	A.06.00	`SUPP_DUPL_IDENT_OUTPUT_MSG`	Keyword for suppress identical output messages option.
6	Used by HP HP OpenView Operations for Windows.		
5	A.05.00	`CONDITION_ID`	Keyword for condition ID.
		`MSGKEY` `MSGKEYRELATION`	Keywords for message correlation.
		`SERVICE_NAME`	Keyword for service name mapping for the HP OpenView Service Navigator.

Table A-1 Template Syntax Versions (Continued)

Syntax Version	OVO Version	Keyword	Description
4	A.04.00	ECS CIRCUIT_FILE	Keywords for event correlation.
		SCHEDULE	Keywords for scheduled actions.
3	A.03.00	Major Minor	Keyword for new severity states.
		MPI_AGT_DIVERT_MSG MPI_AGT_COPY_MSG	Keywords for the agent message stream API.
		NT_UNICODE NT_ANSI_L1 NT_ANSI_JP NT_OEM_L1 NT_OEM_US NT_OEM_JP	Character sets used by Windows NT/2000 managed nodes
2	A.02.00	The syntax version is set to 2 (also used in OpC 2.x) if none of the keywords mentioned above is used.	

Since the syntax descriptions for the configuration files of all these templates share most rules, they are presented here as one grammar. The following restrictions apply:

- ❏ a configuration file for a logfile template must fit the syntax definition <logsources>
- ❏ a configuration file for an SNMP trap template must fit the syntax definition <snmpsources>
- ❏ a configuration file for a MPE/iX console message template must fit the syntax definition <consources>
- ❏ a configuration file for an OVO message interface template must fit the syntax definition <opcsources>
- ❏ a configuration file for an event correlation template must fit the syntax definition <ecsources>
- ❏ a configuration file for a scheduled action template must fit the syntax definition <schedsources>

The grammar is as follows:

```
file ::= SYNTAX_VERSION <num> [ <logsources> | <snmpsources
> |
         <consources> | <opcsources> | <ecsource> |
         <schedsources> e ]

  logsources ::= <logsource > <logsources> | e
  snmpsources ::= <snmpsource> <snmpsources> | e
  consources ::= <consource> <consources> | e
  opcsources ::= <opcsource> <opcsources> | e
  ecsources ::= <ecsource> <ecsources> | e
  schedsources ::= <schedsource> <schedsources> | e

    logsource ::= LOGFILE <string> DESCRIPTION <string>
                  <logdefopts> <conditions>
    snmpsource ::= SNMP <string> DESCRIPTION <string>
                   <snmpdefopts> <snmpconditions>
    consource ::= CONSOLE <string> DESCRIPTION <string>
                  <condefopts> <conditions>
    opcsource ::= OPCMSG <string> DESCRIPTION <string>
                  <opcdefopts> <conditions>
    ecsource ::= ECS <string> DESCRIPTION <string>
                  <ecdefopts> <ecverification>
```

```
              CIRCUIT_FILE <string> <circuit>
schedsource ::= SCHEDULE <string> DESCRIPTION <string>
                <scheddefopts>
  logdefopts ::= <logdefopts> [ <stddefault> |
                 <logoption> | <stdoption> ] | e
  logoption ::= LOGPATH <string> | EXEFILE <string> |
                READFILE <string> | INTERVAL <string> |
                CHSET <chset> | FROM_LAST_POS |
                ALWAYS_FROM_BEGIN | FIRST_FROM_BEGIN |
                NO_LOGFILE_MSG  | CLOSE_AFTER_READ
  snmpdefopts ::= <snmpdefopts> [<stddefault> |
                  <stdoption>] | e
  condefopts ::= <condefopts> [<stddefault> |
                 <stdoption>] | e
  opcdefopts ::= <opcdefopts> [<stddefault> |
                 <stdoption>] | e

  ecdefopts ::= <ecdefopts> [ECS_LOG_INPUT |
                ECS_LOG_OUTPUT] | e
  ecverification ::= VERIFIED | UNVERIFIED
  circuit ::= <circuit> [<string>] | e
  scheddefopts ::= <scheddefopts> [SCHEDPROG
                   <string>] |
                   USER <string> | MONTH <string> |
                   MONTHDAY <string> | WEEKDAY
                   <string> |
                   HOUR <string> | MINUTE <string> |
                   TIMEZONE_VALUE <string> |
                   YEAR <num> | LOGLOCAL | SEND_OUPUT |
                   TIMEZONE_TYPE <tz_type> |
                   BEFORE SET <sets> |
                   FAILURE SET <sets> |
                   SUCCESS SET <sets> | e
    tz_type ::= [MGR_LOCAL | AGT_LOCAL | FIX]
    stddefault ::= SEVERITY <severity> | APPLICATION
                  <string| MSGGRP <string> |
                  OBJECT <string> |
                  MAP_COLORING <string> |
                  SERVICE_NAME <string> |
                  MSG_KEY <string> | MSGKEY <string> |
                  HELPTEXT <string> | HELP <string> |
                  INSTRUCTION_TEXT_INTERFACE <string> |
                  INSTRUCTION_PARAMETERS <string>
    stdoption ::= LOGMATCHEDMSGCOND |
                  LOGMATCHEDSUPPRESS |
                  LOGUNMATCHED | FORWARDUNMATCHED |
                  UNMATCHEDLOGONLY | MPI_SV_COPY_MSG |
                  MPI_SV_DIVERT_MSG | MPI_SV_NO_OUTPUT|

                  MPI_AGT_WPY_MSG |
                  MPI_AGT_DIVERT_MSG |
                  MPI_AGT_NO_OUTPUT |
                  MPI_IMMEDIATE_LOCAL_ACTIONS |
                  SUPP_DUPL_COND <supp_dupl> |
                  SUPP_DUPL_IDENT <supp_dupl> |
                  SUPP_DUPL_IDENT_OUTPUT_MSG
                  <supp_dupl> |
                  SEPARATORS <string> | ICASE |
                  DISABLED

  conditions ::= <conditions> [MSGCONDITIONS <msgconds|

                 SUPPRESSCONDITIONS <suppressconds>
                 SUPP_UNM_CONDITIONS
                 <supp_unm_conds>] | e

    msgconds ::= <msgconds> DESCRIPTION <string>
                 <cond_supp_dupl> <condition_id>
                 CONDITION <conds> SET
                 <sets> | e

    suppressconds ::= <suppressconds> DESCRIPTION
                      <string>
                      <condition_id> CONDITION
                      <conds> | e
    supp_unm_conds ::= <supp_unm_conds> DESCRIPTION
                       <string>
                      <condition_id> CONDITION
                      <conds> | e
    conds ::= <conds> [ SEVERITY <severities> | NODE
              <nodelist>| APPLICATION <string> | MSGGRP

             <string> | OBJECT<string> | TEXT
```

```
      <pattern> ] | e
snmpconditions ::= <snmpconditions> [ MSGCONDITIONS
                   <snmpmsgconds> |
                   SUPPRESSCONDITIONS
                   <snmpsuppressconds>
                   SUPP_UNM_CONDS
                   <snmpsupp_unm_cond> ] | e
snmpmsgconds ::= <snmpmsgconds> DESCRIPTION<string>
                 <cond_supp_dupl> <condition_id>
                 CONDITION
                 <snmpconds> SET <sets> | e
snmpsuppressconds ::= <snmpsuppressconds>
                      DESCRIPTION
                      <string> <condition_id>
                      CONDITION <snmpconds> | e
snmpsupp_unm_conds ::= <snmpsupp_unm_cond>
                      DESCRIPTION <string>
                      <condition_id>
                      CONDITION <snmpconds> | e
  snmpconds ::= <snmpconds> [ $e <string> |
                $G <num>|
                $S <num> | $<num> <pattern> | NODE
                <nodelist> ] | e

  sets ::= <sets> [ SEVERITY <severity> | NODE
           <node>|
           APPLICATION <string> | MSGGRP <string> |

           OBJECT <string> | MSGTYPE <string>| TEXT

           <string> | MAP_COLORING <string> |
           SERVICE_NAME <string> |
           MSG_KEY <string> | MSGKEY <string> |
           MSG_REL_BLANK ACK <pattern> |
           MSGKEYRELATION ACK <pattern> |
           SERVERLOGONLY | AUTOACTION <action> |
           OPACTION <action> | TROUBLETICKET
          [ACK | e] |
           NOTIFICATION | MPI_SV_COPY_MSG |
           MPI_SV_DIVERT_MSG | MPI_SV_NO_OUTPUT |
           MPI_AGT_COPY_MSG | MPI_AGT_DIVERT_MSG |
           MPI_AGT_NO_OUTPUT |
           MPI_IMMEDIATE_LOCAL_ACTIONS | HELPTEXT
           <string> | HELP <string> |
           INSTRUCTION_TEXT_INTERFACE <string> |
           INSTRUCTION_PARAMETERS <string> |
           SERVICE_NAME <string> | <cma> ] | e

cond_supp_dupl ::= SUPP_DUPL_COND <supp_dupl> |
                   SUPP_DUPL_IDENT <supp_dupl> |
                   SUPP_DUPL_IDENT_OUTPUT_MSG
                   <supp_dupl> | e

  supp_dupl ::= <string> |<string> RESEND <string>|

                <string> COUNTER_THRESHOLD <num> |
                <string> COUNTER_THRESHOLD <num>
                RESET_COUNTER_INTERVAL <string> |
                <string>RESEND <string>
                COUNTER_THRESHOLD
                <num> | <string> RESEND <string>
                COUNTER_THRESHOLD <num >
                RESET_COUNTER_INTERVAL <string> |
                COUNTER_THRESHOLD <num> |
                COUNTER_THRESHOLD <num>
                RESET_COUNTER_INTERVAL <string>

  action ::= <string> [ ACTIONNODE <node> | e ]
             [ ANNOTATE | e ] [ ACK | e ]
             [SEND_MSG_AFTER_LOC_AA <msgsendok>
             <msgsendfailed>] | e
  condition_id ::= CONDITION_ID <string> | e
  msgsendok ::= SEND_OK_MSG |
                SEND_OK_MSG LOGONLY| e
  msgsendfailed ::= SEND_FAILED_MSG | e
  pattern ::= <string> [ SEPARATORS <string> | e ]
             [ ICASE | e ]
  chset ::=  ASCII | ACP1252 | ACP932 | OEMCP850 |
             OEMCP437 | ROMAN8 | ISO88591 | ISO8859|
             ISO88595 | ISO88596 | ISO88597 |
             ISO88598 |
             ISO88599 | TIS620 | EBCDIC | SJIS |
             EUCJP |
             EUCKR | EUCTW | GB2312 | BIG5 | CCDC |
            UTF8 | UCS2
  severities ::= <severities><severity> | e
  severity ::= [ Unknown | Normal | Warning |
               Minor |
               Major| Critical ]
  nodelist ::= <node> | <nodelist><node>
  node :: = IP <ipaddress> [ <string> | e] |
            DEC <string> |
            SNA <string> |
            NOVELL <string> |
            OTHER <string>
  ipaddress::= <digits>.<digits>.<digits>.<digits>
  cma::= <cma> | [ CUSTOM <string> <string> ] | e
  string ::= “any alphanumeric string”
             (quotation marks
             and the backslash occurring within a
             string must be masked by a
             backslash ‘\’)
  num ::= [ + | - | e ] <digits>
  digits::= < digits> [ 0 - 9 ] | [ 0 - 9 ]
```

Template Examples

Example of an OVO Logfile Template

The following example is a configuration file for a logfile template:

```
LOGFILE “Su (10.x/11.x HP-UX)”

DESCRIPTION “HP-UX 10.x/11.x switch user events in logfile
            /var/adm/sulog”
LOGPATH     “/var/adm/sulog”
INTERVAL    “20s”
CHSET       ISO8859
SEVERITY    Normal
APPLICATION “/usr/bin/su(1) Switch User”
MSGGRP      “Security”

SUPPRESSCONDITIONS
DESCRIPTION “suppress messages caused by mondbfile monitor
            (SU root-oracle)”
            CONDITION
            TEXT “SU <*> + <@.tty> root-oracle”

MSGCONDITIONS
DESCRIPTION “Bad su”
            CONDITION
            TEXT “SU <*> - <@.tty> <*.from>-<*.to>”

SET
            MPI_AGT_DIVERT_MSG
            MSGTYPE “bad_su”
            SEVERITY Warning
            OBJECT “<from>”
            TEXT “Bad switch user to <to> by <from>”

DESCRIPTION “Succeeded su”
            CONDITION
            TEXT “SU <*> + <@.tty> <*.from>-<*.to>”
```

```
                SET
                            MPI_AGT_DIVERT_MSG
                            MSGTYPE “succeeded_su”
                            OBJECT “<from>”
                            TEXT “Succeeded switch user to <to> by <from>”
```

Example of an OVO Message Source Specification

The following example is a configuration file for the OVO message interface used to intercept messages sent by opcmsg().

As the file is part of a downloaded configuration, it contains UUIDs (the numbers in the lines starting with the keyword “HELP”) that are used internally by OVO to refer to the respective piece of instruction text:

```
OPCMSG “opcmsg(1|3)”

DESCRIPTION “default interception of messages submitted by
            opcmsg(1) and opcmsg(3)”
FORWARDUNMATCHED

MSGCONDITIONS
           DESCRIPTION “PerfView alarms ( REPEAT/END
conditions )”
           CONDITION
           MSGGRP “Performance”
           TEXT   “^\”<*.text>\” START: <*.time>
                   <[REPEAT|END].cond>: <*.end>
                  (<*.option>)”

SET
           TEXT “<text>  ( START: <time> ; <cond>: <end> )”
           OPACTION “pv.sh  <$MSG_NODE_NAME> ‘<$MSG_OBJECT>’
                    <option> “
           HELPTEXT “An alarm was sent by the MeasureWare
                    Agent application.
                    The available operator initiated action
                    allows to start PerfView to review the
                     related metrics values. “
           HELP “5cfdbe1e-90ec-11d1-baa6-0060b0205c3e”
```

Example of an SNMP Trap Template File

```
SNMP “SNMP 6.0 Traps”

           DESCRIPTION “Message Conditions for SNMP Trap
                       Interception”
           SEVERITY Normal
           APPLICATION “SNMPTraps”
           MSGGRP “SNMP”
           FORWARDUNMATCHED
           MSGCONDITIONS

# from EVENT RMON_Rise_Alarm .1.3.6.1.2.1.16.0.1 “Threshold
  Alarms” Warning

           DESCRIPTION “RMON_Rise_Alarm”
                   CONDITION
                         $e “.1.3.6.1.2.1.16”
                         $G 6
                         $S 1
                   SET
                          MPI_AGT_DIVERT_MSG
                          MPI_SV_DIVERT_MSG
                          SEVERITY Warning
                          OBJECT “<$2>”
                          TEXT “RMON Rising Alarm: <$2>
                                exceeded threshold <$5>;
                                value = <$4>. (Sample type =
                                <$3>; alarm index = <$1>)”
                          HELPTEXT “This event is sent when
                                    an RMON device exceeds a
                                    preconfigured
                                    threshold.“

# from EVENT RMON_Falling_Alarm .1.3.6.1.2.1.16.0.2
“Threshold
  Alarms” Warning
           DESCRIPTION “RMON_Falling_Alarm”
                   CONDITION
                         $e “.1.3.6.1.2.1.16”
                         $G 6
                         $S 2
                   SET
                         MPI_AGT_DIVERT_MSG
                         MPI_SV_DIVERT_MSG
                         SEVERITY Warning
                         OBJECT “<$2>”
                         TEXT “RMON Falling Alarm: <$2> fell
below
                               threshold <$5>; value = <$4>.
                               (Sample type = <$3>; alarm
index
                               = <$1>)”
                         HELPTEXT “This event is sent when
                                   an RMON device falls
                                   below a preconfigured
                                   threshold.“
```

Configuration Files for Monitors

The following describes the syntax that is used in configuration files for threshold monitors. These files must fit the syntax rules related to <*monsource*>.

It is recommended to put the keyword SYNTAX_VERSION followed by the number of the current version at the beginning of a configuration file.

NOTE Note that the name of the monitor (the string between the keywords MONITOR and DESCRIPTION) must not contain space characters (blanks).

```
file ::= SYNTAX_VERSION <num> <monsources>

<monsources> ::= <monsource> <monsources> | e

monsource ::= MONITOR <string> DESCRIPTION <string>
              <mondefopts>
              <monconditions>

mondefopts ::= <mondefopts> [ <stddefault> | NODE <nodes> |

               <monoption> | <stdoption> |
               <oldmondefopts> ]  | e

<stddefault> ::= SEVERITY <severity> | APPLICATION <string>
                 | MSGGRP <string> | OBJECT <string> |
                 MAP_COLORING <string> | SERVICE_NAME
                 <string> |
                 MSG_KEY <string> | MSGKEY <string> |
                 HELPTEXT <string> | HELP <string> |
                 INSTRUCTION_TEXT_INTERFACE <string> |
                 INSTRUCTION_PARAMETERS <string>

<monoption> ::= INTERVAL <string> | MONPROG <string> |
                MIB <string> | MIB <string> NODE <node> |
                EXTERNAL | MINTHRESHOLD |
                MAXTHRESHOLD | GEN_BELOW_THRESHOLD |
                    GEN_BELOW_RESET | GEN_ALWAYS

stdoption ::= LOGMATCHEDMSGCOND | LOGMATCHEDSUPPRESS |
              LOGUNMATCHED | FORWARDUNMATCHED |
              UNMATCHEDLOGONLY | MPI_SV_COPY_MSG |
              MPI_SV_DIVERT_MSG | MPI_SV_NO_OUTPUT |
              MPI_AGT_WPY_MSG |
              MPI_AGT_DIVERT_MSG | MPI_AGT_NO_OUTPUT |
              MPI_IMMEDIATE_LOCAL_ACTIONS |
              SUPP_DUPL_COND <supp_dupl> |
              SUPP_DUPL_IDENT <supp_dupl> |
              SEPARATORS <string> | ICASE |
              DISABLED

supp_dupl ::= <string> | <string> RESEND <string>|
              <string> COUNTER_THRESHOLD <num> |
              <string> COUNTER_THRESHOLD <num>
              RESET_COUNTER_INTERVAL <string> |
              <string>RESEND <string> COUNTER_THRESHOLD
              <num> | <string> RESEND <string>
              COUNTER_THRESHOLD <num >
              RESET_COUNTER_INTERVAL <string> |
              COUNTER_THRESHOLD <num> |
              COUNTER_THRESHOLD <num>
              RESET_COUNTER_INTERVAL <string>

oldmondefopt ::= THRESHOLD [ <num> | <floatnum> ] |
                 THRESHOLD [ <num> | <floatnum> ]
                 FOR <string> |
                 RESET [ <num> | <floatnum> ] |
                 MSGTYPE <string> |
                 TEXT <string> |
                 SERVERLOGONLY |
                 LOGLOCAL |
                 AUTOACTION <action> |
                 OPACTION <action> |
                 TROUBLETICKET [ ACK | e ]
                 NOTIFICATION

monconditions ::= <monconditions> [MSGCONDITIONS
                  <monmsgconds> |
                  SUPPRESSCONDITIONS <monsuppressconds> |
                  SUPP_UNM_CONDITIONS
                  <monsupp_unm_conds> ] | e

monmsgconds ::= <monmsgconds> DESCRIPTION <string>
                <condition_id> CONDITION <monconds>
                SET <sets> | e

monsuppressconds ::= <monsuppressconds> DESCRIPTION
                     <string>
                     <condition_id> CONDITION
                     <monconds> | e

monsupp_unm_conds ::= <monsupp_unm_conds> DESCRIPTION
                      <string>
                      <condition_id> CONDITION
                      <monconds> | e

monconds ::= <monconds> | e

moncond ::= THRESHOLD [ <num> | <floatnum> ]
            <duration> | RESET [ <num> | <floatnum> ]
            OBJECT <pattern>

duration ::= FOR <string> | e

pattern ::= <string> [ SEPARATORS <string> | e ]
            [ ICASE | e ]

action ::= <string> [ ACTIONNODE <node> | e ]
           [ ANNOTATE | e ] [ ACK | e ]
           [SEND_MSG_AFTER_LOC_AA <msgsendok>
           <msgsendfailed>] | e

condition_id ::= CONDITION_ID <string> | e

severity ::= [ Unknown | Normal | Warning | Minor | Major |
             Critical ]

node ::= IP <ipaddress> [ <string> | e] |
         DEC <string> |
         SNA <string> |
         NOVELL <string> |
         OTHER <string>

     ipaddress::= <digits>.<digits>.<digits>.<digits>

     string ::= “any alphanumeric string” (quotation marks
                and the backslash occurring within a string
                must be masked by a backslash ‘\’)

     floatnum ::= [ + | - | e ] <digits>.<digits> | <num>

     num ::= [ + | - | e ] <digits>

     digits::= < digits> [ 0 - 9 ] | [ 0 - 9]
```

Example of an OVO Monitor Template

The following text shows the definition of a monitor for which the OVO intelligent agent calls a program every 10 minutes to determine disk usage.

```
#
-----------------------------------------------------------
----
# Template: disk_util
#
-----------------------------------------------------------
----

SYNTAX_VERSION 2
MONITOR “disk_util”

DESCRIPTION “Monitor disk space utilization on root disk”
INTERVAL    “10m”
MONPROG     “disk_mon.sh disk_util”
THRESHOLD   90.000000
RESET       85.000000
MAXTHRESHOLD
GEN_BELOW_RESET
SEVERITY    Warning
APPLICATION “OVO”
MSGGRP      “OS”
OBJECT      “root_disk”
TEXT       “Utilization of root disk (<$VALUE>%) is greater
           than <$THRESHOLD>%.”
AUTOACTION  “ana_disk.sh” ANNOTATE
HELPTEXT “Available space on the device holding the / (root)
filesystem is less than the configured threshold. This may
lead to problems for applications running on the affected
system requesting large amounts of storage space. Also,
performance penalties might be encountered.”

# end of disk_util
```

The program or script `disk_mon.sh` must contain a command or function call such as:

```
opcmon “disk_util={$RETRIEVED_VALUE}”
```

or

```
opcmon (object, value)
```

to pass on the current value of the monitored objects to the OVO Monitor Agent (opcmona).

Syntax for Message Pattern Matching

Table A-2 shows the components of the OVO pattern-matching language that can be used to write expressions to match incoming messages. You can combine individual components to form complex patterns.

Table A-2 OVO Pattern Matching Language

Component	**Description**
Ordinary Characters	Ordinary characters are expressions that represent themselves. Any character of the supported character set can be used. However, if any of the following special characters are used: [ ] < > | ˆ $ they must be prefaced with a backslash (\ ) to mask their usual function. If ˆ and $ are not used as Anchoring Characters, they are considered as ordinary characters.
The Mask Character	Use the backslash (\ ) to mask the special meaning of the characters: [ ] < > | ˆ $ A special character preceded by \ results in an expression that matches the special character itself. Note that because ˆ and $ only have special meaning when placed at the beginning and end of a pattern respectively, you need not mask them when they are used within a pattern (in other words, not at beginning or end). The only exception is the tab character, that is specified by entering **/t** in the pattern string.
Expression Anchoring	If the caret (ˆ) is used as the first character of the pattern, only expressions discovered at the beginning of lines are matched. For example, “ˆab” matches the string “ab” in the line “abcde”, but not in the line “xabcde”. If the dollar sign is used as the last character of a pattern, only expressions at the end of lines are matched. For example, **de$** matches **de** in the line **abcde**, but not in the line **abcdex**.

Table A-2 OVO Pattern Matching Language (Continued)

Component	Description
Bracket Expressions:	The brackets “ [” and “] ” are used as delimiters to group expressions. To increase performance, avoid brackets wherever they are superfluous. In the pattern **ab[cd[ef]gh]** all brackets are unnecessary **- abcdefgh** is equivalent. Expressions with brackets are used frequently with the **Alternative Operator** or the **NOT Operator**. Brackets are also often useful when assigning values to variables.
Expressions Matching Multiple Characters:	Patterns used to match strings consisting of an arbitrary number of characters are represented by one of the following expressions. Depending on the symbol used, the characters matched by the expression are restricted to a certain set. <*> Matches any string of zero or more arbitrary characters (including separators). <n*> Matches a string of n arbitrary characters (including separators). <#> Matches a sequence of one or more digits. <n#> Matches a sequence of n digits. <_> Matches a sequence of one or more separator characters. <n_> Matches a string of n separators. <@> Matches any string that contains no separator characters, for example, a sequence of one or more non-separators; this can be used to match words.

Table A-2 OVO Pattern Matching Language (Continued)

Component	Description
Variables	The matched string can be assigned to a variable that you can use to recompose messages or as a parameter for action calls. To define a parameter, add the string: **.parametername** before the closing bracket. For example, the pattern: ^errno: <#**.number**> **-** <***.error_text**> will match a message of the format: **errno: 125 - device does not exist** and assigns **125** to the parameter **number** and the message **device does not exist** to the parameter **error_text**. Variable names may only contain alphanumeric characters as well as underscores (_) and hyphens (-). The following syntax rules apply: `(Letter | '_'){ Letter | Digit | '_' | '-' }` In the syntax above, `Letter` allows letters and ideographic characters from all alphabets, and `Digit` allows digit characters from all alphabets.

Table A-2 OVO Pattern Matching Language (Continued)

Component	**Description**
Assign-to-Variable Operator	In addition to being able to use a single expression, such as **<*>** or **<#>** to assign a string to a variable, it is also possible to use the assign-to-variable operator to build up a complex sub-pattern composed of a number of operators. The assign-to-variable operator uses the same square brackets as in other bracket expressions. The basic pattern is: **<[sub-pattern].var>** Example 1: **<[<@>file.tmp].fname>**In this example, the dot between **file** and **tmp** matches a dot character, while the dot between **]** and **fname** is necessary syntax. This pattern would match a string such as **Logfile.tmp** and assign the complete string to **fname**. Example 2: **<[Warning|Error].var>**This pattern matches any instance of the string **Warning** or the string **Error** found in the message text. The message text: **Warning and Error: Shutdown** would therefore cause the string **Warning** to be assigned to **var**. Note that the first instance of a matched string is assigned. Example 3: **<[Error[<#.n><*.msg>]].complete>**In this case, any line containing the word **Error** followed by a number, has the number assigned to the variable **n** and any further text assigned to **msg**. Finally, both number and text are assigned to **complete**.

Table A-2 OVO Pattern Matching Language (Continued)

Component	**Description**
NOT Operator	The not operator (**!**) must be used with delimiting square brackets, for example: **<![WARNING]>** The pattern above matches all text that does not contain the string **WARNING**. The NOT operator may also be used with complex sub-patterns, for example: **SU <*> + <@.tty> <![root|[user[1|2]]].from>-<*.to>** This pattern makes it possible to generate a **switch user** message for anyone who is not root, user1, or user2. For example, the following would be matched: **SU 03/25 08:14 + ttyp2 user11-root** However, the following would not be matched, because it contains an entry concerning **user2**: **SU 03/25 08:14 + ttyp2 user2-root** Note that if the sub-pattern including the not operator does not find a match, the not operator behaves like a **<*>**: it matches zero or more arbitrary characters. For this reason, the **UN*X [!123]** expression cannot be duplicated: OVO's **<![1|2|3]>** matches any character or any number of characters, except 1, 2 or 3; the UN*X operator matches any one character, except 1, 2 or 3.
Alternative Operator	Two expressions separated by the pipe character (|) match a string that is matched by either expression. For example, the pattern **[ab|c]d** matches both the string **abd** and the string **cd**.

Table A-2 OVO Pattern Matching Language (Continued)

Component	**Description**
Numeric Range Operators	The pattern for constructing complex expressions with these operators, is: **<number operator [sub-pattern] operator number>** The square brackets are part of the syntax and must be provided as literals in the pattern. The sub-pattern can be a simple numeric operator, for instance **<#>** or **<2#>**. These simple operators do not require delimiting brackets. Alternatively, it may be a complex sub-pattern, using delimiting brackets, for example: **<120 -gt [<#>1] -gt 20>** It is also possible to construct a pattern using only one operator: **Error <<#> -gt 1004>** The following **Numeric Range Operators** are available: -le Less than, or equal to -lt Less than -ge Greater than, or equal to -gt Greater than -eq Equal to -ne Not equal to

Pattern Matching

It is important to understand how OVO pattern matching works, especially in conjunction with assignment to parameters.

When matching the pattern:

```
<*.var1><*.var2>
```

with the string **abcdef**, it is not immediately clear which substring of the input string will be assigned to each variable. For example, it is possible to assign an empty string to **var1** and the whole input string to **var2**, or to assign "**a**" to **var1** and "**bcdef**" to **var2**, and so on.

The OVO pattern-matching algorithm always scans both the input line and the pattern definition, including alternative expressions, from left to right.

<*> expressions are assigned as few characters as possible.

<#>, **<@>** and **<_>** expressions are assigned as many characters as possible.

Consequently, **var1** is assigned an empty string in the above example.

To match an input string such as:

```
this is error 100: big bug
```

use a pattern such as: **error<#.errnumber>:<*.errtext>**

```
error<#.errnumber>:<*.errtext>
```

The result of this match is that **100** is assigned to **errnumber**, and **big bug** is assigned to **errtext**.

For performance and pattern readability purposes, you can specify a delimiting substring between two expressions. In the above example, " **:** " is used to delimit **<#>** and **<*>**.

Matching **<@.word><#.num>** with "**abc123**" assigns "**abc12**" to **word** and "**3**" to **num**, because digits are permitted for both **<#>** and **<@>**, and the left expression takes as many characters as possible.

Patterns without expression anchoring can match any substring within the input line. Therefore, patterns such as:

```
this is number<#.num>
```

are treated in the same way as:

```
<*>this is number<#.num><*>
```

Separator Characters

The separator characters used in **<_>** and **<@>** can be specified for each pattern. The user enters the separators in the pattern definition mask; default characters are the blank and tab characters.

Case Insensitive Mode

OVO allows to specify case sensitive or insensitive mode for each pattern. The user sets the mode in the pattern definition mask.

Pattern Matching Examples

Table A-3 Pattern Matching Examples

Format	**Recognized Messages**
Error	Will recognize any message containing the keyword “Error” at any place in the message.
panic	Matches all messages containing “panic”, “Panic”, “PANIC”, etc., at any place, if case insensitive mode is used.
logon|logoff	Recognizes any message containing the keyword “logon” or “logoff”.
^getty: <*.msg> errno<*><#.errnum>$	Matches messages of format: “getty: cannot open tty’xx’ errno : 6” or “getty: can’t open ttyop3; errno 16” Note: the anchoring symbol **$** is used to assign the number at the end of the input line to **errnum**.
^errno[|=]<#.errnum> <*.errtext>	Matches messages of format: “errno 6 - no such device or address” as well as “errno=12 Not enough core” The space between **<#.errnum>** and **<*.errtext>** is used as delimiter for correct assignment of the number to **errno**.
^hugo:<*>:<*.uid>:	Matches any entry in /etc/passwd for user ‘hugo’ and returns the user ID in parameter **uid**. Note the “**:**” at the end of the pattern to delimit **uid** from the succeeding group ID in the input pattern.
^Warning: <*.text> on node <@.node>$	Matches a message of format “Warning: too many users on node hpbbx” and assigns “too many users” to **text** and “hpbbx” to **node**. Note: the anchoring symbol **$** is used to assign the word at the end of the input line to node.
SU<*>+<@.tty> <![root|admin].from> -<*.to>	Matches all SU logfile entries of all users except for “root” and “admin”.

Configuration Files for Applications

This section describes the syntax used in configuration files for applications. These files must fit the following syntax rules:

```
file ::= DOWNLOAD_DATA APPLICATION <syntax_version>
         <application_groups> | e

syntax_version ::= SYNTAX_VERSION <digits> | e

application_groups ::= <application_groups>
                       <application_group> ;

application_group ::= APPLICATION_GROUP
                      <application_group_spec>
                      SUBENTITIES APPLICATION
                      { <application_in_group> } |
                      APPLICATION_GROUP
                      <application_group_spec>
                      SUBENTITIES APPLICATION_GROUP
                      { <applgrp_in_group> } SUBENTITIES
                      APPLICATION
                      { <application_in_group> } |
                      MEMBER_APPLICATION_GROUP
                      <application_group_spec> SUBENTITIES
                      APPLICATION_GROUP
                      { <applgrp_in_group> }
                      SUBENTITIES APPLICATION {
                      <application_in_group> }

application_group_spec ::= <string> SYMBOL
                          <string> | <string> SYMBOL
                          <string> LABEL <string>
                          DESCRIPTION
                          <string> | PSEUDO_GROUP

applgrp_in_group ::= <applgrp_in_group>
                     MEMBER_APPLICATION_GROUP <string> | e

application_in_group ::= <application_in_group>
                         <application_data> | e
```

```
application_data ::= APPLICATION_REF <string> |
                     APPLICATION <string> SYMBOL <string>
                     [ TARGET <num> | START_ON_SEL_FLAG
                     <bool> ]
                     LABEL <string>
                     DESCRIPTION <string> APPL_CALL
                     <string>
                     INTERN_APPL_ACTION <num>
                     NODE { <application_node_list> }
                     APPL_LOGIN { [ UUID <uuid> | e ]
                     <application_login_list> }
                     APPLICATION_TYPE
                     <application_type_data>

application_node_list ::= <application_node_list>
                          [ <node_ident>
                          | PATTERN_OTHER <string>] | e

node_ident ::= _OPC_MGMTSV_ |
               IP <ipaddress> | IP <ipaddress>
               <string> | SNA <string> | DEC <string> |
               NOVELL
               <string> | OTHER <string>

application_login_list ::= <application_login_list>
                           [PLTFRM_FAMILY_NAME <string>
                           USER_NAME
                           <string> PASSWORD <string> ]

application_type_data ::= CSM_PLATFORM_INTERNAL |
                          CSM_PLATFORM_INTEGRATED UUID
                          <uuid>
                          START_IN_TERM_FLAG <num>
                          PARAMETERS <string>
                          USER_NAME <string>
                          PASSWORD <string> |
                          OV_PLATFORM REGISTERED_NAME
                          <string>
                          ACTION_IDENTIFIER <string>

ipaddress ::= <digits>.<digits>.<digits>.<digits>
```

```
string ::= “any alphanumeric string” (quotation
           marks and the backslash occuring within a
           string must be masked by a backslash ‘\’)

num ::= [ + | - | e ] <digits>

digits ::= <digits> [ 0 - 9 ] | [ 0 - 9 ]

bool ::= 0 | 1

uuid ::= “any UUID (Universal Unique Identifier)”
         (A UUID string consists of eight hexadecimal
         digits followed by a dash, followed by three
         groups of four hexadecimal digits separated by
         dashes, followed by a dash and twelve hexadecimal
         digits.)
```

Example of an OVO Application Configuration File

The following is an example of an Application Configuration File for an OVO application. This file is part of the OVO default configuration.

```
APPLICATION “OVO Status”
   SYMBOL “Software:Process”
   START_ON_SEL_FLAG FALSE
   LABEL “OVO Status”
   DESCRIPTION ““
   APPL_CALL “/opt/OV/bin/OpC/opcragt -status $OPC_NODES”
   INTERN_APPL_ACTION 0
   NODE
   {
      _OPC_MGMTSV_
   }
   APPL_LOGIN
   {
   }
   APPLICATION_TYPE CSM_PLATFORM_INTEGRATED
   UUID “4314a356-4c40-71d0-172c-0f887bb10000”
   START_IN_TERM_FLAG 2
   PARAMETERS ““
   USER_NAME “root”
   PASSWORD “
```

Syntax and Length of OVO Object Names

Syntax checks are automatically performed when entering information in the corresponding fields of the GUI. For more information about length limitations of OVO object names, see the *OVO Reporting and Database Schema*.

B Symbols for Application Integration

Available Symbols for Application Integration

The OVO symbol type is a combination of the <*shape*> parameter and the <*bitmap_name*> parameter in the syntax:

<*shape*>_<*bitmap_name*>

The default symbol type used in the English version of the OVO Application Desktop is <*Software_applications*>.

You can view valid combinations by reviewing the contents of the directory:

`/etc/opt/OV/share/symbols/C`

For example, the following `Software` Symbols are available:

- ❏ `Software`
- ❏ `Software_applications`
- ❏ `Software_customer`
- ❏ `Software_database`
- ❏ `Software_directory`
- ❏ `Software_driver`
- ❏ `Software_file`
- ❏ `Software_filesystem`
- ❏ `Software_group`
- ❏ `Software_license`
- ❏ `Software_mail`
- ❏ `Software_opsystem`
- ❏ `Software_process`
- ❏ `Software_provider`
- ❏ `Software_software`
- ❏ `Software_user`

NOTE

These symbols are defined for the English version of OVO. Localized versions are available for the Japanese language OVO.

C About OVO Man Pages

In this Appendix

This appendix describes the man pages available in the following areas:

- ❏ Man Pages in OVO
- ❏ Man Pages for OVO APIs
- ❏ Man Pages for HP OpenView Service Navigator
- ❏ Man Pages for the OVO Developer’s Kit APIs

Accessing and Printing Man Pages

You can access the OVO man pages from the command line, from online help, or in HTML format on your management server.

To Access an OVO Man Page from the Command Line

To access an OVO man page from the command line, enter the following:

```
man <manpagename>
```

To Print a Man Page from the Command Line

To print an OVO man page from the command line, enter the following:

```
man <manpagename> | col -lb | lp -d printer_name
```

To Access the Man Pages in HTML Format

To access the OVO man pages in HTML format, from your Internet browser, open the following location:

```
http://<management_server>:3443/ITO_MAN
```

In this URL, <*management_server*> is the fully qualified hostname of your management server.

Man Pages in OVO

This section describes man pages in OVO.

Table C-1 **OVO Man Pages**

Man Page	Description
call_sqlplus.sh(1)	Calls SQL*Plus.
inst.sh(1M)	Installs OVO software on managed nodes.
inst_debug(5)	Debugs an installation of the OVO agent software.
ito_op(1M)	Launches the OVO Java-based operator or Service Navigator GUI.
ito_op_api_cli(1M)	Enables calling the Java GUI Remote APIs.
opc(1|5)	Starts the OVO GUI.
opc_audit_secure(1M)	Locks the audit level in the OVO database, and allows directories for the history and audit download to be set.
opc_backup(1M)	Interactively saves the OVO environment for Oracle.
opc_backup(5)	Backs up the OVO configuration.
opc_chg_ec(1M)	Changes circuit names in event correlation (EC) templates in the OVO database.
opc_recover(1M)	Interactively recovers the OVO environment for Oracle.
opc_recover(5)	Recovers the OVO configuration.
opcack(1M)	Externally acknowledges active messages.
opcackmsg(1M)	Externally acknowledges active messages using message IDs.
opcackmsgs(1M)	Externally acknowledges active messages using specific message attributes.
opcactivate(1M)	Activates a pre-installed OVO agent.
opcadddbf(1M)	Adds a new datafile to an Oracle tablespace.

Table C-1 OVO Man Pages (Continued)

Man Page	Description
opcagt(1M)	Administers agent processes on a managed node.
opcagtreg(1M)	Registers subagents.
opcagtutil(1M)	Parses the agent platform file, and performs operations with extracted data.
opcaudupl(1M)	Uploads audit data into the OVO database.
opcaudwn(1M)	Downloads audit data into the OVO database.
opccfgdwn(1M)	Downloads configuration data from the database to flat files.
opccfgout(1M)	Configures condition status variables for scheduled outages in OVO.
opccfgupld(1M)	Uploads configuration data from flat files into the database.
opcchgaddr(1M)	Changes the address of nodes in the OVO database.
opccltconfig(1M)	Configures OVO client filesets.
opcconfig(1M)	Configures an OVO management server.
opccsa(1M)	Provides the functionality for listing, mapping, granting, denying and deleting specified certificate requests.
opccsacm(1M)	Performs the ovcm's functionality for manually issuing new node certificate and using the installation key.
opcdbidx(1M)	Upgrades the structure of the OVO database.
opcdbinit(1M)	Initializes the database with the default configuration.
opcdbinst(1M)	Creates or destroys the OVO database scheme.
opcdbpwd(1M)	Changes the password of the OVO database user opc_op.
opcdbreorg(1M)	Re-organizes the tables in the OVO database.
opcdbsetup(1M)	Creates the tables in the OVO database.

Table C-1 OVO Man Pages (Continued)

Man Page	Description
`opcdcode(1M)`	Views OVO encrypted template files.
`opcerr(1M)`	Displays instruction text for OVO error messages.
`opcgetmsgids(1m)`	Gets message IDs to an original message ID.
`opchbp(1M)`	Switches heartbeat polling of managed nodes on or off.
`opchistdwn(1M)`	Downloads OVO history messages to a file.
`opchistupl(1M)`	Uploads history messages into the OVO database.
`opcmack(1)`	Acknowledges an OVO message by specifying the message ID.
`opcmgrdist(1M)`	Distributes the OVO configuration between management servers.
`opcmom(4)`	Provides an overview of OVO MoM functionality.
`opcmomchk(1)`	Checks syntax of MoM templates.
`opcmon(1)`	Forwards the value of a monitored object to the OVO monitoring agent on the local managed node.
`opcmsg(1)`	Submits a message to OVO.
`opcpat(1)`	Tests a program for OVO pattern matching.
`opcragt(1M)`	Remotely administers agent services for OVO on a managed node.
`opcskm(3)`	Manages secret keys.
`opcsqlnetconf(1M)`	Configures the OVO database to use an Net8 connection.
`opcsv(1M)`	Administers OVO manager services.
`opcsvreg(1M)`	Registers server configuration files.
`opcsvskm(1M)`	Manages secret keys on the management server.
`opcsw(1M)`	Sets the software status flag in the OVO database.
`opcswitchuser(1M)`	Switches the ownership of the OVO agents.

Table C-1 OVO Man Pages (Continued)

Man Page	Description
`opctempl(1M)`	Maintains templates in files.
`opctemplate(1M)`	Enables and disables templates.
`opctmpldwn(1M)`	Downloads and encrypts OVO message source templates.
`opcwall(1)`	Sends a message to currently logged in OVO users.
`ovocomposer(1M)`	Performs tasks related to OV Composer.
`ovocomposer(5)`	Describes the Correlation Composer, an HP OpenView Operations (OVO) event correlation feature.
`ovtrap2opc(1M)`	Converts the `trapd.conf` file and the OVO template file.

Man Pages for OVO APIs

This section describes man pages for OVO application program interfaces (APIs).

Table C-2 OVO API Man Pages

Man Page	Description
`opcmon(3)`	Forwards the value of a monitored object to the OVO monitoring agent on the local managed node.
`opcmsg(3)`	Submits a message to OVO.

Man Pages for HP OpenView Service Navigator

This section describes man pages for the HP OpenView Service Navigator.

Table C-3 Service Navigator Man Pages

Man Page	Description
`opcservice(1M)`	Configures HP OpenView Service Navigator.
`opcsvcattr (1M)`	Add, change or remove service attributes.
`opcsvcconv(1M)`	Converts service configuration files of HP OpenView Service Navigator from the previous syntax to the Extensible Markup Language (XML).
`opcsvcdwn(1M)`	Downloads service status logs of HP OpenView Service Navigator to a file.
`opcsvcterm(1M)`	Emulates an interface to HP OpenView Service Navigator. The interface inputs Extensible Markup Language (XML) markup into `stdin` and outputs Extensible Markup Language (XML) markup to `stdout`.
`opcsvcupl(1M)`	Uploads service status logs of HP OpenView Service Navigator into the OVO database.

Man Pages for the OVO Developer's Kit APIs

This section describes man pages for the OVO Developer's Kit application program interfaces (APIs).

Table C-4 **OVO Developer's Toolkit Man Pages**

Man Page	Description
msiconf(4)	Configures the OVO message manager.
opc_comif_close(3)	Closes an instance of the communication queue interface.
opc_comif_freedata(3)	Displays free data that was allocated by opc_comif_read().
opc_comif_open(3)	Opens an instance of the communication queue interface.
opc_comif_read(3)	Reads information from a queue.
opc_comif_read_request(3)	Reads information from a queue.
opc_comif_write(3)	Writes information into a queue.
opc_comif_write_request(3)	Writes information into a queue.
opc_connect_api(3)	Connects OVO.
opc_distrib(3)	Distributes the OVO agent configuration.
opcagtmon_send(3)	Forwards the value of a monitored object to OVO.
opcagtmsg_api(3)	Handles messages on OVO agents.
opcanno_api(3)	Manages OVO message annotations.
opcapp_start(3)	Starts an OVO application.
opcappl_api(3)	Configures and starts OVO applications.
opcapplgrp_api(3)	Configures OVO application groups.
opcconf_api(3)	Gets OVO configuration.

Table C-4 OVO Developer's Toolkit Man Pages (Continued)

Man Page	Description
opcdata(3)	Accesses the attributes of the OVO data structure.
opcdata_api(3)	Describes how to access the OVO data structure using the OVO Data API.
opcif_api(3)	API to work with the OVO Message Stream Interface.
opciter(3)	OVO iterator to step through opcdata container.
opcmsg_api(3)	Manages OVO messages.
opcmsggrp_api(3)	Manages OVO message groups.
opcmsgregrpcond_api(3)	Creates and modifies OVO message regroup conditions.
opcnode_api(3)	Configures OVO managed nodes.
opcnodegrp_api(3)	Configures OVO node groups.
opcnodehier_api(3)	Configures OVO node hierarchies.
opcprofile_api(3)	Configures OVO user profiles.
opcregcond(3)	Accesses fields of the OVO registration condition structure.
opcsvc_api(3)	C++ classes for Service Navigator.
opctempl_api(3)	Configures OVO message source templates.
opctempfile_api(3)	Configures OVO templates using template files.
opctemplgrp_api(3)	Configures OVO template groups.
opctransaction_api(3)	Starts, commits, and rolls back transactions.
opcuser_api(3)	Configures OVO users.
opcversion(3)	Returns the string of the OVO library that is currently used.