HP Operations Manager Developer's Toolkit

Application Integration Guide

Software Version: 9.02

for the UNIX and Linux operating systems

Manufacturing Part Number: None
February 2010

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 at 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.

Intel®, Itanium®, and Pentium® are trademarks of Intel Corporation in the U.S. and other countries.

Java™ is a US trademark of Sun Microsystems, Inc.

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

1. An Introduction to Integrating Partner Applications with HPOM

2. Integrating Solutions with HPOM

3. Using the HPOM Application Programming Interfaces

6. Creating and Distributing an Integration Package

A. Syntax Used in HPOM Configuration Files

B. About HPOM 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: June 2009

Second Edition: February 2010

Conventions

The following typographical conventions are used in this manual:

Table 1 Typographical Conventions

Font	Meaning	Example
Italic	Book titles and manpage names	For more information, see the *HPOM Administrator's Reference* and the *opc(1M)* manpage.
	Emphasis	You *must* follow these steps.
	Variable that you must supply when entering a command (in angle brackets)	At the prompt, enter **`rlogin <username>`**.
	Parameters to a function	The *`oper_name`* parameter returns an integer response.
`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	Edit the `itooprc` file... `/opt/OV/bin/OpC/`
	Process names	Check to see if `opcmona` is running.
`Computer Bold`	Text that you enter	At the prompt, enter **`ls -l`**

Table 1 Typographical Conventions (Continued)

Font	Meaning	Example
Keycap	Keyboard keys	Press **Return**.
	Menu name followed by a colon (:) means that you select the menu, and then the item. When the item is followed by an arrow (->), a cascading menu follows.	From the menu bar, select **Actions: Filtering -> All Active Messages**.
	Buttons in the user interface	Click **OK**.

HPOM Documentation Map

HP Operations Manager (HPOM) provides a set of manuals and online help that help you to use the product and to 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 the manuals are available as Adobe Portable Document Format (PDF) files in the documentation directory on the HPOM product CD-ROM.

With the exception of the *HPOM Software Release Notes*, all the manuals are also available in the following HPOM 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 the English environment.

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

`http://support.openview.hp.com/selfsolve/manuals`

Watch this website regularly for the latest edition of the *HPOM Software Release Notes*, which is updated every two to three months with the latest news (for example, additionally supported operating system versions, the latest patches and so on).

HPOM Manuals

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

Table 2 HPOM Manuals

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

Table 2 HPOM Manuals (Continued)

Manual	Description	Media
Service Navigator Concepts and Configuration Guide	Provides information for administrators who are responsible for installing, configuring, maintaining, and troubleshooting the HP Operations Service Navigator. This manual also contains a high-level overview of the concepts behind service management.	PDF only
HPOM 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
HPOM Firewall Concepts and Configuration Guide	Designed for administrators. This manual describes the HPOM firewall concepts and provides instructions for configuring the secure environment.	PDF only
HPOM Web Services Integration Guide	Designed for administrators and operators. This manual describes the HPOM Web Services integration.	PDF only
HPOM Security Advisory	Designed for administrators. This manual describes the the HPOM security concepts and provides instructions for configuring the secure environment.	PDF only
HPOM Server Configuration Variables	Designed for administrators. This manual contains a list of the HPOM server configuration variables.	PDF only

Additional HPOM-Related Products

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

Table 3 Additional HPOM-Related Manuals

Manual	Description	Media
HP Operations Manager Developer's Toolkit If you purchase the HP Operations Manager Developer's Toolkit, you receive the full HPOM documentation set, as well as the following manuals:		
HPOM Application Integration Guide	Suggests several ways in which external applications can be integrated into HPOM.	PDF
HPOM Developer's Reference	Provides an overview of all the available application programming interfaces (APIs).	PDF

HPOM Online Information

The following information is available online.

Table 4 HPOM Online Information

Online Information	Description
HPOM Java GUI Online Information	HTML-based help system for the HPOM Java-based operator GUI and Service Navigator. This help system contains detailed information about general HPOM and Service Navigator concepts and tasks for HPOM operators, as well as reference and troubleshooting information.
HPOM Manpages	Manpages available online for HPOM. These manpages 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 manpages for the HP Operations HTTPS agents are installed on each managed node.

HPOM Online Help

This preface describes online documentation for the HP Operations Manager (HPOM) Java operator graphical user interface (GUI).

Online Help for the Java GUI and Service Navigator

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

Types of Online Help

The online help for the HPOM 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 might encounter while using the product.

- **Index**

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

Viewing 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.

Accessing the 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 HPOM to use your preferred browser.

1 An Introduction to Integrating Partner Applications with HPOM

Why Integrate with HPOM?

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 Operations Manager Developer's Toolkit** you have a powerful tool at your disposal to integrate your network solutions into HP Operations Manager. By employing the standard integration capabilities of HPOM, and the extended capabilities of the HPOM 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 Operations Manager (HPOM) 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;
- ❏ Web-based Administration GUI, as well as a Java-based operator GUI.

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

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

  Interface API to access HPOM 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 HPOM data directly in the database. The functions enable you, for example, to configure new HPOM policies or managed nodes, or to modify existing applications or users. In addition, functions are available to control access to HPOM data, and to distribute configuration changes to the managed nodes.

For more information about the HPOM User APIs, see Chapter 3, “Using the HPOM Application Programming Interfaces,” on page 73.

These features make HPOM ideally suited as an integration framework for other applications or solutions which address the system and network management market. Integration with HPOM 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 HPOM 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 Software Partnerships

The major benefit resulting from an integration with HPOM is the increased customer value of the integrated solution. HPOM 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 HPOM, 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 Software Platinum Partner**. Validation ensures that the integration is well-behaved and does not conflict with other integrated solutions. As an HP Software Platinum Partner, your solution is recommended by HP sales channels, you can leverage from the well-established HP Software brand name, and you receive immediate market exposure for your solution through HP market awareness and selling tools.

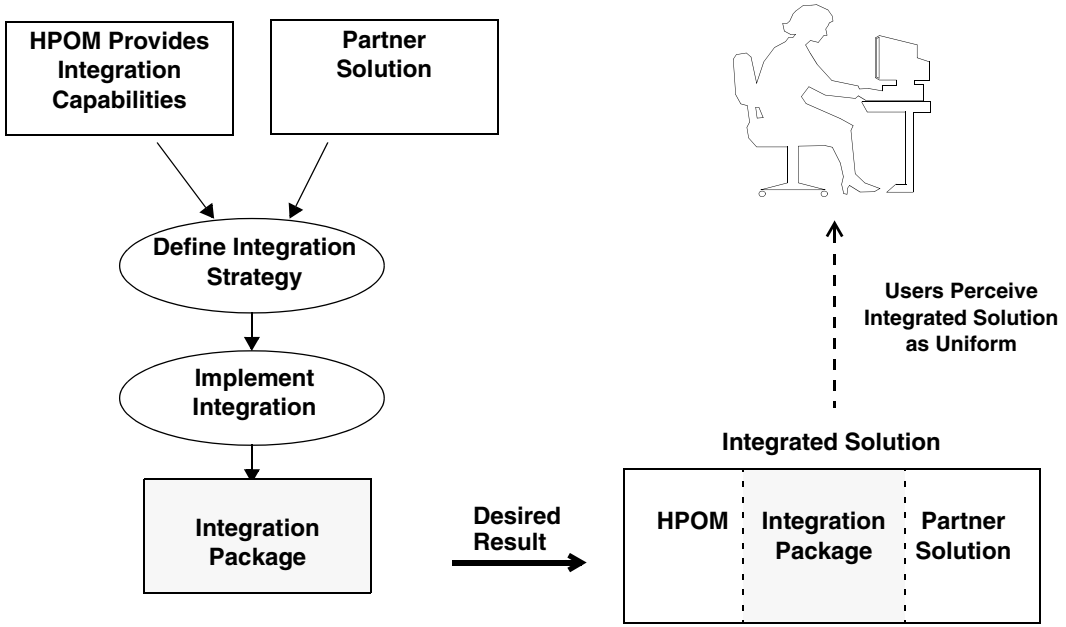
For more information about the HP Software partner programs, see our web site at `http://h20229.www2.hp.com/partner/index.html`, and select `partners`.

Integrating Partner Solutions with HPOM

The ultimate goal of any integration must be to create an **integration package** that enables HPOM 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 HPOM 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 Operations Manager



We use the term **tight integration** if the capabilities offered by HPOM are fully exploited to maximize the uniformity of the integrated solution. For example, consider integrating a business solution with HPOM. To achieve tight integration status, HPOM capabilities should be employed to ensure that the application is constantly monitored so that HPOM 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 HPOM functionality are highly configurable to enable a high degree of customization, and to provide unique opportunities for the tight integration of partner solutions.

HPOM 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, 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 HPOM application desktop, menu bar, submenus, or toolbar.
- ❏ APIs and command line interfaces to the agents and to the management server.

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

HPOM Conceptual Overview

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

The HP Software Product Family and HPOM

HP Software 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. HPOM is one of the key components of the HP Software Solution Framework that has become a leading Integrated Network and System Management (INSM) solution.

HPOM 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 HPOM to control the following elements and resources:

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

To achieve efficient operation and problem management, information must be gathered from the controlled elements and resources. HPOM 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 HPOM. Message filters and thresholds are used to ensure that only the relevant information is sent to the management server and presented to the responsible HPOM operators.

After collecting data, HPOM 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 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 HPOM Message Browser. This guarantees the maximum possible efficiency in local and central event analysis and handling. HPOM 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 HPOM domains.

You can easily adapt HPOM 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 HPOM operators to work together simultaneously in the same computing environment, without a duplication of effort.

The implementation of HPOM 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. HPOM also supports competitive management approaches, for example, **follow-the sun**.

Open APIs and external interfaces enable 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, HPOM 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.

HPOM Implementation

HPOM is a distributed client-server software solution and its architecture adheres to the manager/agent concept. Within a computing environment managed by HPOM 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 **HPOM agent software**. The HPOM agent on the managed node collects and processes management information, forwards pertinent information to the appropriate management server(s), and starts local actions.

HPOM 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 HPOM management server and the HTTPS-based managed nodes is based on 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 HPOM intelligent agent only when a problem is detected. This minimizes the network traffic and increases the performance of the management server.

Problem Management with HPOM

Regardless of the message source, HPOM 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

HPOM 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 HPOM 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.

Agent Message API

Management information, generated by applications or customer programs or scripts, can even be sent directly to the HPOM agent on any managed node by way of the Agent Message API, see Chapter 3, “Using the HPOM Application Programming Interfaces,” on page 73.

Threshold Monitors

The threshold monitoring capability of HPOM 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.

HPOM 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 HPOM agent on any managed node by the Agent Monitor API, to be locally checked against predefined thresholds, see Chapter 3, “Using the HPOM Application Programming Interfaces,” on page 73.

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

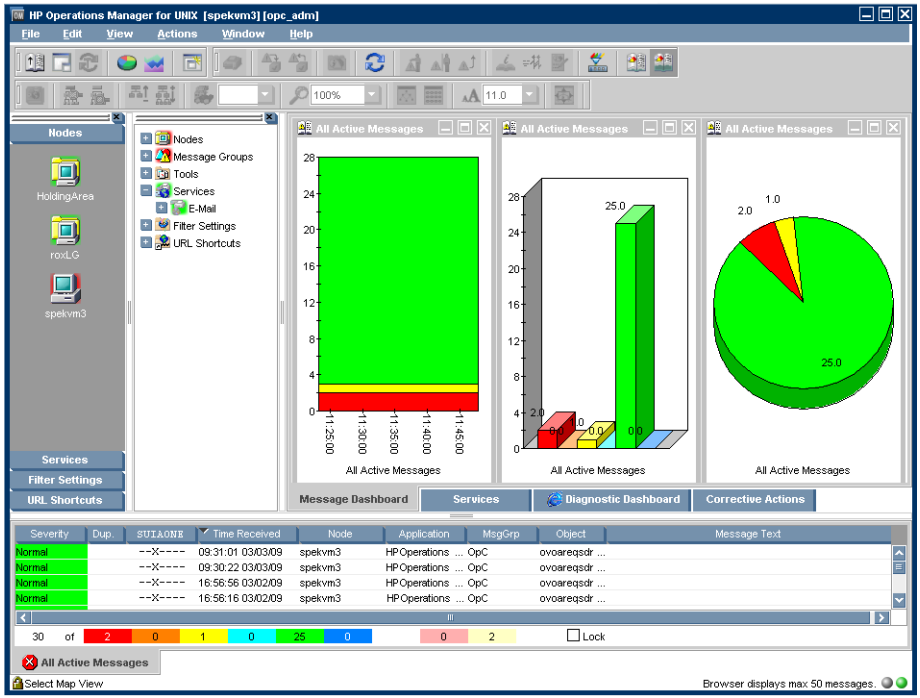
Legacy Link Interface API

To integrate hardware platforms that are not, or not yet, supported by HPOM, the **Legacy Link Interface API** is provided to receive and pass on management information, see “Overview of the Message Event Interface” on page 92.

The HPOM Java-based Operator User Interface

HPOM provides a Java-based operator GUI. If you are using the Java-based user interface, your working environment looks similar to the one in Figure 1-2. See the online documentation supplied with the Java-based GUI for more information about how to perform tasks in this environment.

Figure 1-2 The Java-based Operator GUI



You can operate with Java-based GUI remotely from other Java applications using the Java GUI Remote APIs. For more information, refer to Chapter 4, “Integrating with Java GUI,” on page 137.

Processing and Consolidating Information

HPOM 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 HPOM 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. HPOM provides several default message groups; see the *HPOM Administrator's Reference* for a complete list. As an administrator, you can add, review, and delete message groups. You can perform these tasks using the web-based Administration GUI. See the Administration GUI user documentation available for download in the HP Operations Manager for UNIX directory at: http://support.openview.hp.com/selfsolve/manuals.

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 policies at the management server and then download them to the managed nodes using the `opctmpldwn(1)` command. 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. HPOM 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 HPOM user is presented by Java GUI. For more information on Java GUI, refer to the *HPOM Java GUI Operator's Guide*. The Java GUI `Object Pane` displays the structure of your managed environment with the following objects:

- Managed Nodes
- Message Groups
- Tools

The Java GUI `Message Browser` displays information about your managed environment through the messages. A message represents an event that has occurred on a node within the managed environment (for example, a status change or a threshold violation) generated by the agents running on the node.

The content of the Java GUI 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 HPOM 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 HPOM 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 HPOM are complemented by integrated partner solutions. Examples of tightly integrated solutions are HP Data Protector, and HP OmniStorage.

Acting on the Provided Information

HPOM offers several mechanisms for responding to events. When an event occurs that requires a non-interactive, corrective action, you can configure HPOM 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, HPOM 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 Java GUI Message Browser.

In the Java GUI, you can start custom scripts, programs, and management applications.

The console login is under HPOM 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, HPOM also provides a broadcast facility.

HPOM 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 HPOM

HPOM 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.

HPOM 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 `opccfguser -assign_respons_user` command, 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 Java GUI `Message Browser` of that user. For more information, see the *opccfguser(1m)* man page.

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 HPOM server in London between 8 AM and 5 PM, and to an HPOM 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 HPOM environment with multiple management servers, see the *HPOM Concepts Guide.*

Using the **message forwarding** feature, messages can also be transferred between management servers.

The HPOM 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 the Tools list of that user.

HPOM also provides secure operations. Each user has a password to ensure that only authorized people can access HPOM. User profiles defined by the HPOM administrator restrict the activities of each user on the management server and the managed nodes. All actions can be controlled because activities are initiated from the operator's GUI which is tailored to the responsibilities of the user.

Integration Benefits to Partners

HPOM is a leading framework for **Integrated Network and System Management (INSM)** so when integrating with HPOM, a partner solution becomes a component of an INSM solution used to manage a complete IT environment. Integration into HPOM 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.

HPOM as an INSM Framework

HPOM 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. HPOM 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.

HPOM 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 HPOM 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 HPOM 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)
- ❏ Network 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 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 HPOM in a similar way as into NNM. This integration would still benefit from features that are standard in HPOM. This type of solution can easily be migrated from NNM to an HPOM 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 HPOM to prepare a management solution to become part of an INSM solution. These solutions can take full advantage of the HPOM-specific integration capabilities from event integration via messages, to using the various HPOM APIs.

To integrate applications or databases for which no satisfactory problem management solutions are available, HPOM 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 with HPOM and to provide some additional shell scripts.

INSM Market Segment

The following solutions should also consider HPOM 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 HPOM.

These solutions may benefit considerably from an integration into HPOM, 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 HPOM.

These solutions can take full advantage of the HPOM-specific integration capabilities from event integration via messages to using the various HPOM APIs. Especially for trouble ticket and help desk systems and for event correlation engines, HPOM 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 HPOM integration:

- ❏ Service providers

Integration Facilities Provided by HPOM

HPOM is designed to provide the maximum flexibility for integrators, so the mechanisms underlying HPOM functionality are configurable to a high degree. This enables you to customize HPOM installations to the specific needs of your customers. In addition, HPOM 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 *HPOM Software Release Notes* for more information about the new features provided with this release of HPOM.

Integrating Events Using Messages

HPOM 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 HPOM agent detects that a monitor threshold has been exceeded.
- When functions of the Agent Message API or Agent Monitor API are called.

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

Threshold Monitoring

The preceding sections described the integration of events of which HPOM could read some type of text, for example, text written to a logfile or associated with an SNMP trap. In addition, HPOM 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, HPOM 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 HPOM agents directly. The functions of the Agent Monitor API allow other applications to pass monitor values to an HPOM 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 HPOM checks it against predefined upper or lower limits.

- **Performance matrices**

  Use the embedded performance component to collect performance counter and instance data. The platform-generic matrices can be used to answer most questions about a system's global configuration, CPU, disk, swap, and memory usage. The typical matrices 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 HPOM, 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 HPOM threshold monitors.

Working with Message Policies

You can use message policies to filter and suppress messages, reformat message text, and link actions and instructions with a message. Messages can be defined using policies by either uploading them using the `opcpolicy` command or creating them in the Administration UI. You can specify policies by editing the policy body. See Appendix A, “Syntax Used in HPOM Configuration Files,” on page 181.

By configuring HPOM policies, 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 policy specifies the composition of the message, its attributes, and the instructions, annotations, and actions to be associated with the message.

- **Basic Policy Properties**

  These include the policy 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 policy 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.

- **Policy Conditions**

  A list of **conditions** belongs to each message policy. 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 HPOM in the order in which they are listed. The first matching condition determines how HPOM 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.

❏ **Logging and Unmatched Messages**

In the policy body, 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 policy and message condition level. If different values for the advanced options are set at the message condition level, these will overwrite the policy 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 Policy

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 HPOM lets you completely rephrase the text of a message so that it can be easily understood by an operator. This feature can help HPOM 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.

HPOM 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 in the policy body, and then define more specific instructions for each message condition of a policy.

- **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**

  HPOM 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 HP Operations management server and thus are performed even if the network is down or the HP Operations management server is not available.

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

- **Operator-initiated Actions**

  HPOM 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 by editing policy bodies. For the information on policy body grammar, see Appendix A, “Syntax Used in HPOM Configuration Files,” on page 181.

Integrating Events using Trouble Ticket and Notification Services

HPOM includes interfaces to trouble-ticket and notification systems which enable communication from the HP Operations 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 HPOM 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 HPOM. The HPOM administrator can set up different applications (tools) for each HPOM operator. You can group similar applications together to avoid cluttering up the Java GUI.

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.

To integrate an application, you can also add your own entries into the HPOM 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 Java GUI window before a menu item becomes active. Inactive menus are automatically grayed-out. Toolbars provide a quick, intuitive means of invoking actions. HPOM provides a default set of toolbars for invoking actions such as panning or selecting the root map. When an application is added to an HPOM environment, it can add icons into existing toolbars, or create window-specific toolbars and icons.

Integrating via APIs

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

- **HPOM Operator APIs**

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

  - HPOM Data API
  - HPOM Interface API

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

❑ **HPOM Configuration APIs**

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

- Connection API
- Application Configuration API
- Application Group Configuration API
- Category Configuration API
- Instruction Text Interface Configuration API
- Message Group Configuration API
- Message Regroup Condition Configuration API
- Node Configuration API
- Node Hierarchy Configuration API
- Notification Schedule Configuration API
- Notification Service Configuration API
- Policy Configuration API
- Trouble Ticket Configuration API
- User Profile Configuration API
- User Configuration API
- Distribution API
- Server Synchronization API
- Pattern Matching API

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

The HPOM 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 HPOM 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 HPOM.

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

NNM Integration Through the HPOM GUI

HPOM features a default remote integration with NNM providing the operator with the ability to start `ovw` applications from the HPOM Java GUI. This functionality is available with the NNM installed on a remote system other than the HPOM management server.

2 Integrating Solutions with HPOM

Deciding Which Integration Capabilities to Use

The different integration capabilities provided by HPOM 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 policies for a particular message source that can be active at a given time. Templates for each source (logfiles, SNMP traps, HPOM Message API, and other) are evaluated in parallel.

When integrating a partner solution it is recommended that you create a new policy 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 not to use the `FORWARDUNMATCHED` keyword in policy bodies 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 HPOM agent whenever possible.
2. Prefer capabilities handled on the managed nodes to those handled on the management server only.
3. Define a new policy for each integration and/or message source; do not append new conditions to an existing message source policy.

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 HPOM 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, and Windows agent platforms only) or Management Server
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 HPOM (aimed at the NSM market segment).
- Starting with an existing integration into HP Network Node Manager (NNM) (aimed at the NIM market segment).
- Starting from scratch

  No previous HP Software integration available.
- Obtaining coexistence of NNM and HPOM

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

Adapting an Existing HPOM Integration for HPOM 9.xx

NOTE

If you need to integrate into HPOM a partner solution that was integrated into a previous (and recent) version of HPOM, any integration points introduced with HPOM 9.xx will not be used. In addition, you may have to recompile the application with the libraries from the latest version of the HPOM software, if those libraries have been changed.

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

- An existing integration was designed for the HPOM 8.xx release and takes full advantage of the integration capabilities offered by this release for a tight integration to be achieved.
- An existing HPOM integration does not take advantage of the integration capabilities offered by the HPOM 8.xx release.

In the second case, in which the integration does not take full advantage of HPOM 8.xx integration features, a complete verification of the integration is recommended.

If you can leverage from a tight HPOM 8.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 HPOM. Refer to the *HPOM Software Release Notes* for more information about the new features of HPOM.

Leveraging From an Integration into NNM

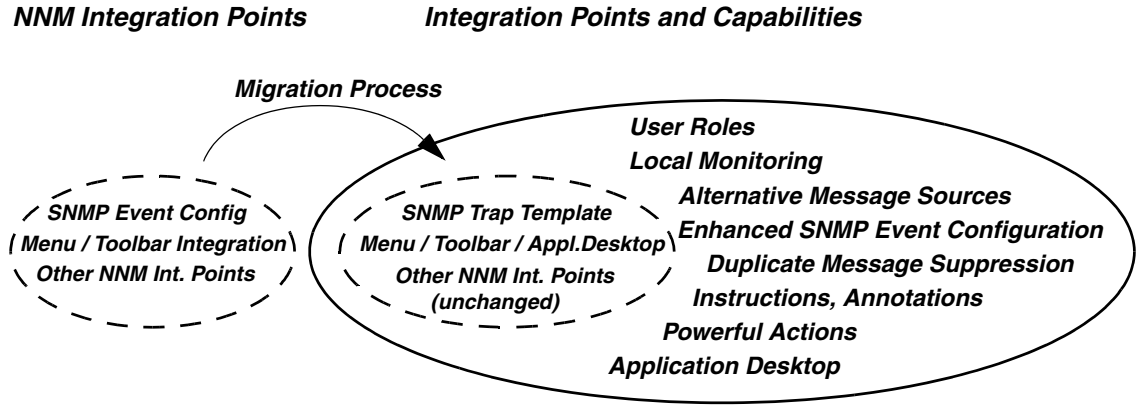
If you currently use NNM, it is worth remembering that, as an HPOM network operator, you can work in a way that is similar to the role of the typical NNM operator. HPOM provides the preconfigured network operators **netop** and **itop** to whom the `IPMap` application, and the Network and SNMP message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

Some applications that are a part of Network Node Manager (NNM) are automatically integrated into the HPOM. For a list and description of the default NNM application groups and applications, see the *HPOM Administrator's Reference*.

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.

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

Figure 2-1 SNMP Event Configuration



SNMP Event Configuration

SNMP traps are configured using the SNMP Trap policy. An SNMP Trap policy 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:

- HPOM 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 HPOM managed node running the agent software. The status of an action is visible in the HPOM 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 HPOM message which can either be statically configured for each condition or template, or dynamically created using the Instruction Text Interface.
- 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

HPOM applications are integrated in Java GUI and are displayed as tools. The integration method for HPOM 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 HPOM administrator.

Applications newly added to HPOM should always be integrated as HPOM applications.

Alternative Message Sources

SNMP traps are only one of several different message sources for HPOM. Logfile templates can be used to generate messages on HPOM 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 HPOM 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 HPOM 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 acknowledge the message.

Advantages of an INSM Solution

HPOM provides:

- ❏ Reflection of node status in IP Map windows
- ❏ HP Service Navigator integration on the management server

Starting from Scratch

To integrate a partner solution that is not currently integrated into either NNM or HPOM, 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 HP Software 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 the administrator (also using the Administration GUI) are the preferred choice.

Obtaining Coexistence of NNM and HPOM Integrations

HPOM provides an integration with the HP Network Node Manager (NNM) installed on the remote system. This integration enables users to execute HP Software applications from the HPOM Java GUI.

If you plan to integrate your product into both NNM and HPOM, consider that NNM cannot be installed on the same system as the HP Operations management server.

To make use of the remote HPOM integration with Network Node Manager (NNM), see the *HPOM Administrator's Reference*.

Summary of the Integration Process

The outcome of any integration is an **integration package** that contains HPOM 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 HPOM APIs. In this case, the integration package would contain other items in addition to the HPOM 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 HPOM 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 HPOM?
- ❏ Does the partner solution include tools that can be offered to HPOM operators in the Java GUI?

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 HPOM 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. This includes:

   - Templates for the selected message sources: logfiles, SNMP traps, and calls to the HPOM 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
- 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 HPOM 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.
5. Add other required files to the configuration information to create the integration package.

   Other required files can be the necessary executables, such as scripts, commands, actions.

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 HPOM 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 29 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 HPOM to work together smoothly.

An integration package always contains HPOM 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
- ❏ Policies and policy groups
- ❏ User and user profiles
- ❏ Action, command, and monitor executables
- ❏ Administrator configuration
- ❏ Database maintenance (configuration of HPOM internal database)
- ❏ 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 HPOM. Consequently, the configuration data required by HPOM to interact with a partner solution must be incorporated into the HPOM internal database using the configuration download and upload utilities of HPOM.

To download configuration data from HPOM use the configuration download command `opccfgdwn(1M)`. You can later upload configuration information into the HPOM internal database using the `opccfgupld(1M)` command. For more information, see the respective man pages.

Note that service data of the HP 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 HPOM 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 HPOM installation. From the customer's perspective, an "out-of-the-box" integration of the partner solution and HPOM is provided.

Figure 2-2 on page 71 summarizes how the HPOM configuration download and upload capabilities are used to define a configuration and then to distribute it to customers.

Figure 2-2 shows the following steps:

Step 1: Define the HPOM configuration.

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

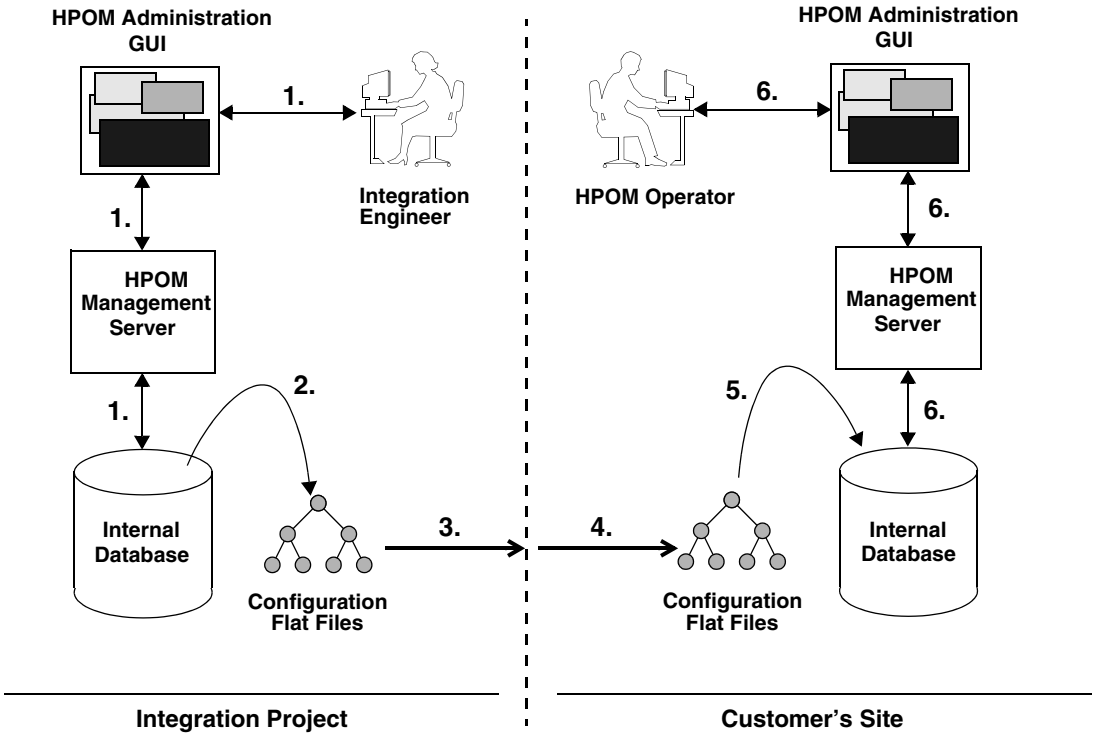
Step 3: Package the configuration with the product and ship to the customer.

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

Step 5: Upload the HPOM configuration from the flat files.

Step 6: Use the new, uploaded configuration.

Figure 2-2 Handling HPOM 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 Administration GUI and command-line interfaces.

The uploading and downloading capabilities, and the structure of the configuration file tree are described in more detail in Chapter 6, "Creating and Distributing an Integration Package," on page 167.

3 Using the HPOM Application Programming Interfaces

In This Chapter

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

Overview of the HPOM APIs

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

- **HPOM Operator APIs**

  The HPOM Operator APIs provide a set of functions that allow you to operate on HPOM 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 HPOM 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 HPOM Interfaces use the Interface API functions to register with HPOM to receive data. When the requested data is available, HPOM sends it to the instance of the interface which made the request.

- **HPOM Configuration APIs**

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

For a list of man pages available with the HPOM Developer's Toolkit, see Appendix B, "About HPOM Man Pages", on page 221.

Figure 3-1 illustrates the information exchange between HPOM and the APIs.

Figure 3-1 Overview of HPOM APIs on the Management Server and Managed Nodes

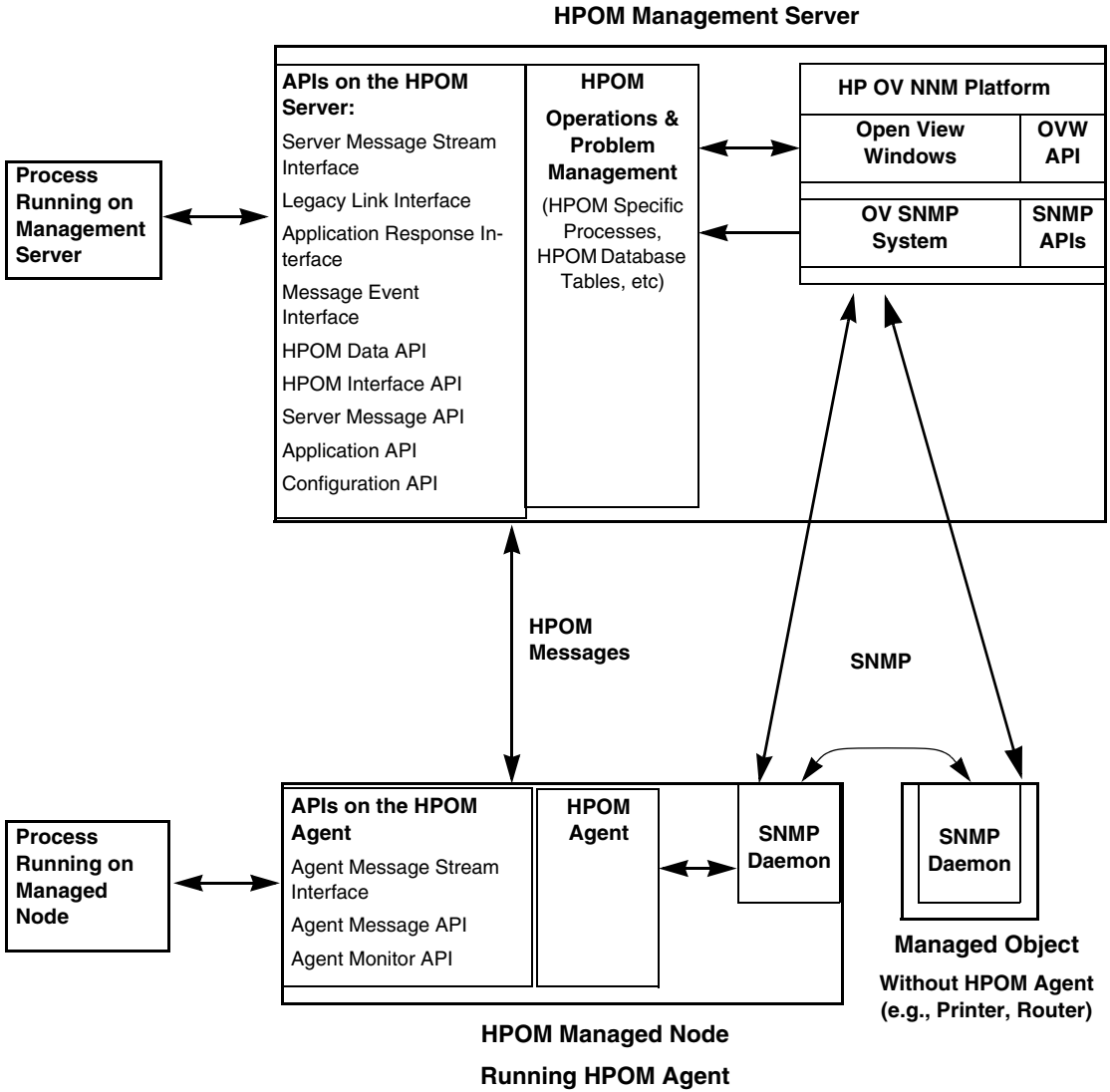


Table 3-1 on page 77 gives an overview of the location of the HPOM APIs.

Table 3-1 Location of the HPOM APIs

API	Location
HPOM Operator APIs	
HPOM Data API	Management Server and Managed Node
HPOM Interface API	Management Server and Managed Node
Server Message API	Management Server
Agent Message API	Managed Node
Agent Monitor API	Managed Node
HPOM 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
HPOM Configuration APIs	
Connection API	Management Server
Application Configuration API	Management Server
Application Group Configuration API	Management Server
Category Configuration API	Management Server
Instruction Text Interface 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
Notification Schedule Configuration API	Management Server

Table 3-1 Location of the HPOM APIs (Continued)

API	Location
Notification Service Configuration API	Management Server
Policy Configuration API	Management Server
Trouble Ticket Configuration API	Management Server
User Profile Configuration API	Management Server
User Configuration API	Management Server
Distribution API	Management Server
Synchronization API	Management Server
Pattern Matching API	Management Server

The HPOM Interfaces

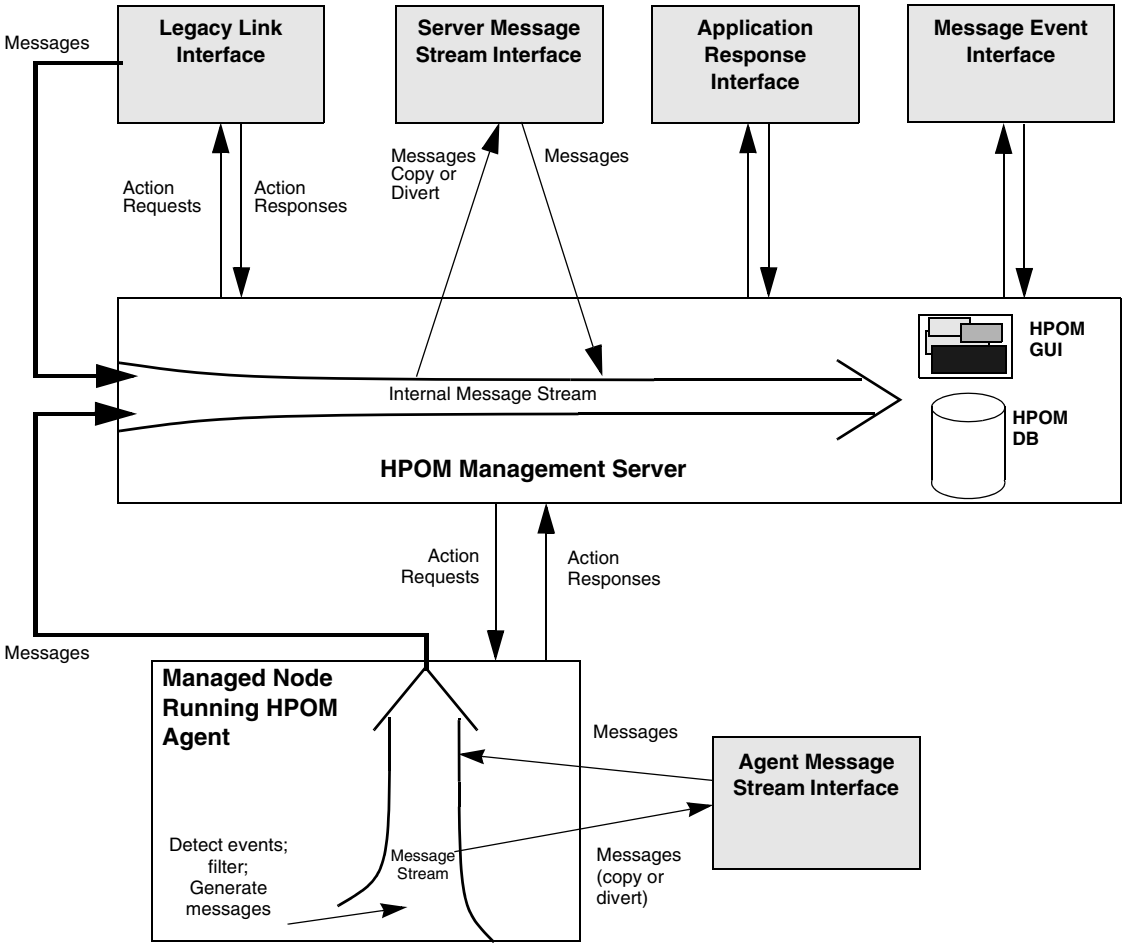
The following interfaces to HPOM can be accessed by way of the Interface API:

Interface	Description
Server Message Stream Interface	Enables HPOM messages to be output from the server message stream to an external application.
Agent Message Stream Interface	Enables HPOM messages to be output from the agent message stream to an external application.
Legacy Link Interface	Provides a link between HPOM and managed nodes for which HPOM agents are not currently available.
Application Response Interface	Allows external applications to receive application responses from HPOM applications that have been started.
Message Event Interface	Allows external applications to receive message events to display HPOM 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 3-2 on page 80 shows how the HPOM Interfaces interact with the internal message stream of the HPOM 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 HPOM 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 3-2 Interaction of HPOM 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 3-1 on page 77 for more information.

Overview of the Server Message-Stream Interface

The HPOM management server's message-stream interface provides access to the internal message stream of the HPOM management server. Processes may connect to the message-stream interface on the management server either in parallel or in series. By default, all MSIs connect in parallel. If you want to use MSIs in series, you must configure the `/etc/opt/OV/share/conf/OpC/mgmt_sv/msiconf` file, which is described in more detail in "Serial MSI Configuration File" on page 82.

Multiple connections to the MSI are organized by means of order numbers. Assigning an order number to each MSI connection allows the 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 reduce 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 HPOM interface first requires a parameter that specifies the type of interface to be accessed (server MSI, legacy-link interface, and so on), followed by a user-defined instance name.

The routines return an error/ok code, with errors being logged in the file `/var/opt/OV/log/System.txt`.

Configuration Stream Interface

Configuration Stream Interface (CSI) is an extension of the Message Stream Interface (MSI) for synchronizing the configuration changes. CSI provides registration for the configuration changes to the internal (server processes, Java GUI) and external (API clients) configuration consumers. Java GUI and server processes are registered for the configuration changes by default.

Access to the Server Message-Stream Interface

If an application registers at the Message Stream Interface (MSI), HPOM checks the database to verify whether the MSI is enabled. If it 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 the type `OPCSVIF_EXTMSGPROC_READ` and `OPCSVIF_EXTMSGPROC_READWRITE`. If MSI is configured to copy incoming messages by using the `opcsrvconfig -msi -enable -send copy` command, a copy of the message is immediately passed back to the internal flow of the management server, see Figure 3-6 on page 96.

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.

Serial MSI Configuration File

The name of each MSI instance and the order number assigned to it are stored in the MSI configuration file:

`/etc/opt/OV/share/conf/OpC/mgmt_sv/msiconf`

Entries in the MSI configuration file use the following format:

<MSI instance name> <order number>
<MSI instance name> <order number>
…
<MSI instance name>

User-defined name of the MSI instance.

The length of the string is restricted to twelve (12) ASCII characters.

The interface ID returned by the open routine is then used in subsequent calls to the other API functions.

<order number>

Order number of the MSI instance.

HPOM allows you to assign order numbers between -127 and 127. Gaps are also allowed in the sequence.

The start number zero (0) 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.

Both parallel and serial connections from the MSI are possible at any point in the serial chain.

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 manager process (`opcecm`), assuming the event-correlation manager is running.

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 `System.txt` log file.

Serial MSI Configuration: Example Scenario

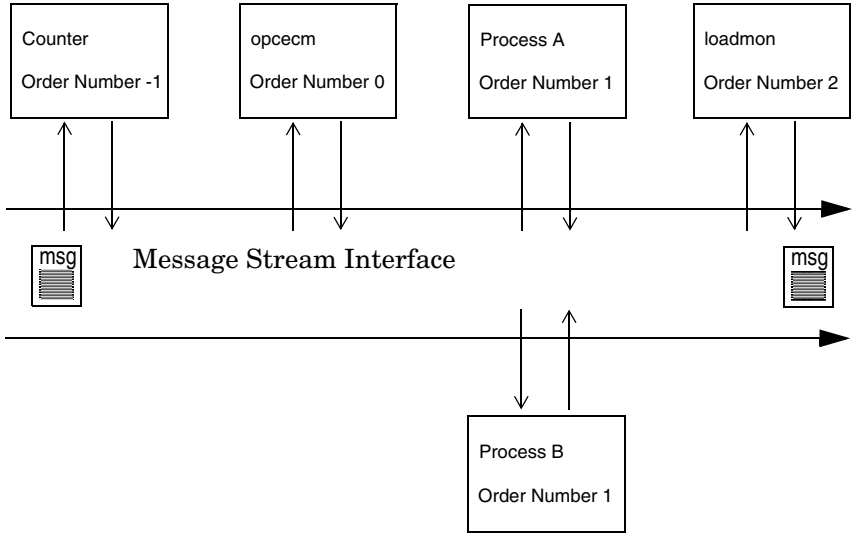
Figure 3-3 on page 84 illustrates how messages are distributed in a scenario where five processes are connected to the MSI: `counter`, `opcecm`, `procA`, `procB`, and `loadmon` (load monitor).

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 processes that are connected must have a number assigned 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 3-3 Message Distribution in the Serial MSI



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:

- **Diverted messages**

  No new message ID is generated if the message has been diverted to the MSI; that is, no other copy of the message exists either inside or outside the MSI.

- **Unmodified messages**

  No new message ID is generated if 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 your applications require read/write access to messages.

You can control the generation of message IDs by setting the parameter, `OPC_MSI_CREATE_NEW_MSGID` using the `ovconfchg` command-line tool.

The following values are possible:

Value	Description
1	Enable HPOM 4.00 behavior. Create a new message ID each time a message attribute is changed or a message is copied.

Value	Description
2	Enable HPOM 5.00 and higher behavior (default). Do not generate a new message ID if the attributes of a *diverted* message change; that is, no other copy of the message exists either inside or outside the MSI. When you copy a message within the MSI application using the `opcdata_copy()` function, the copied message is no longer considered *diverted*; subsequent attribute changes lead to a new message ID. The old message ID is stored in the original message ID field.
3	Same as 2 with the following exception: When you copy a message within the MSI application using the `opcdata_copy()` function, the copied message will immediately get a new message ID.
4	Custom: no automatic creation of new message IDs. You are responsible for creating new message IDs yourself as needed, using the `opcdata_generate_id()` or `opcdata_set_str()` functions. For more information about these functions, see the *HPOM Developer's Reference*.

Duplicate Message Suppression

Duplicate messages are counted and suppressed after they have passed through the Message Stream Interface (MSI) but before they are added to the database. This is why the duplicate count field `OPCDATA_NUM_DUPLICATES` of the `OPCDTYPE_MESSAGE` data structure is always zero (0) for messages passing through the MSI.

If the message is a duplicate of another message, the message is discarded after passing through the MSI and will not be added to the HPOM database. The duplicate count of the original message is increased by one (1).

It is possible to change the processing order so that duplicate processing of messages takes place before messages arrive at the MSI. In this case duplicate messages are never written to the MSI because they are discarded before they even get there.

To process duplicate messages before writing them to the MSI, use the `ovconfchg` command-line tool to set the `OPC_SUPPRESS_DUPL_AFTER_MSI` setting to `FALSE`:

```
# ovconfchg -ovrg server -ns opc \
-set OPC_SUPPRESS_DUPL_AFTER_MSI FALSE
```

The default for `OPC_SUPPRESS_DUPL_AFTER_MSI` is `TRUE`.

CAUTION Use this setting only if no other MSI applications, for example ECS circuits, rely on receiving duplicate messages.

Overview of the Agent Message Stream Interface

The **Agent Message Stream Interface** allows you to tap the message flow of an HPOM 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 HPOM intelligent agent software is not available, such as legacy mainframe systems.

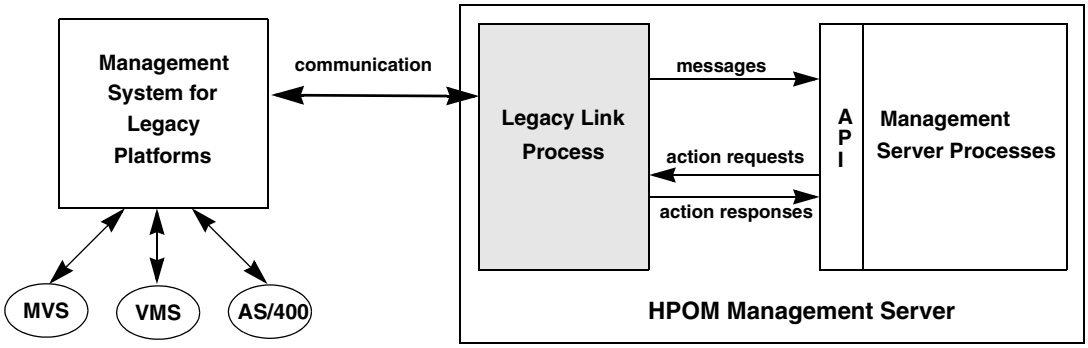
This API provides an interface to a process running on the management server node. This process can send (write) messages, receive action requests, and send action responses. It must handle all communication with the network node that lacks the intelligent agent, and it must ensure that the messages 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 3-4 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 3-4 Integrating a Legacy System Using the Legacy Link Interface



Structure of the Legacy Link Process

Figure 3-5 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 HPOM 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.

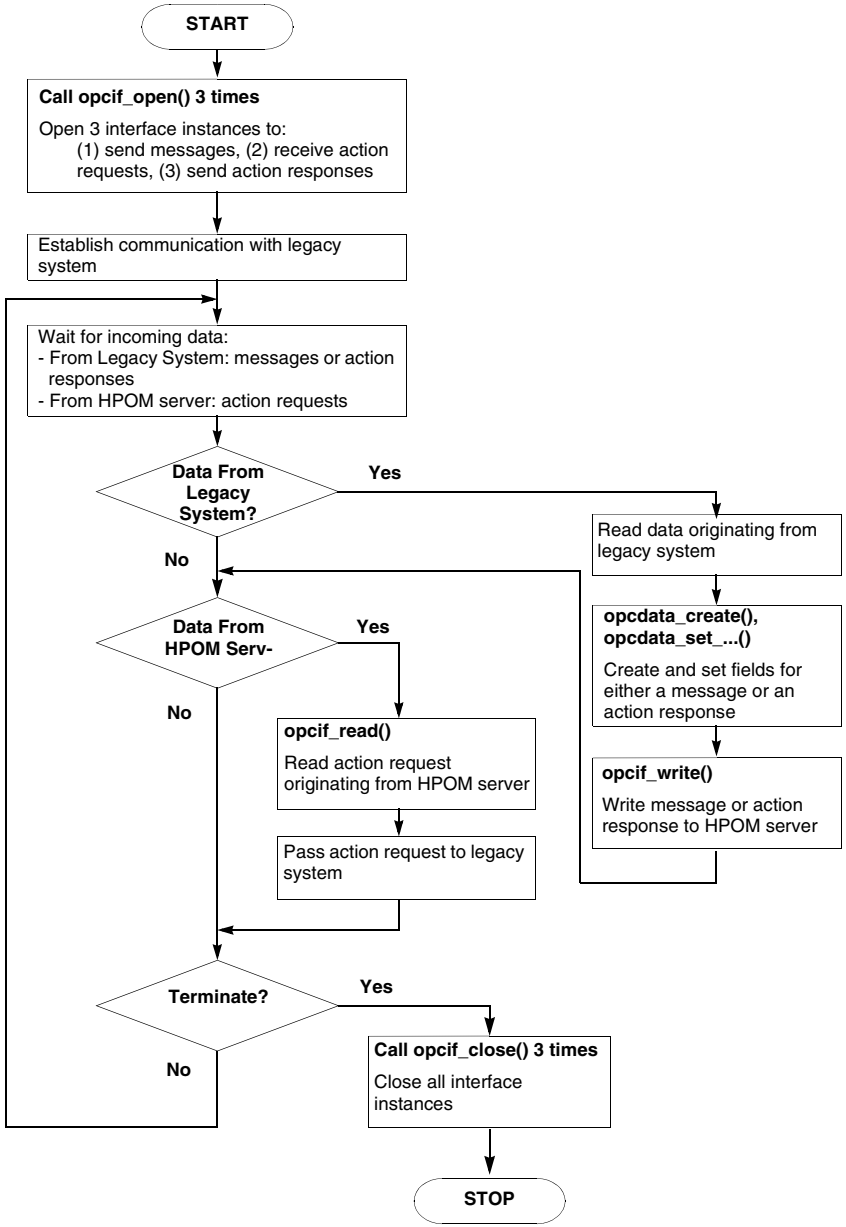
In addition, the legacy link process must establish a bidirectional communication channel with the legacy system to receive input for generating messages and action responses, and to pass on action requests received from the management server.

As soon as the communication channel between the legacy system and the HPOM 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 HPOM management server. If the incoming data originates from the HPOM 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 HPOM management server.

Figure 3-5 Using the Legacy Link Interface to Integrate a Legacy System



Overview of the Message Event Interface

The **Message Event Interface** enables an application running on the HPOM management server to register for and receive **Message Events**. A message event is generated whenever the status of an HPOM 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 HPOM 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 HPOM and accessed by the client, see Figure 3-7 on page 97. 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 3-2 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
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
Information messages from HPOM users	OPC_MSG_EVENT_INFORM_USER

Overview of the Application Response Interface

The **Application Response Interface** enables an external application to register for and receive **Application Responses** from HPOM applications that have been started by the Application API `opcappl_start()`. HPOM 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 3-7 on page 97.

Read and Write Access to the HPOM Message Stream

Figure 3-6 on page 96 shows the difference between the three types of process that access the HPOM 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 HPOM 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 HPOM in the same way as the HPOM agents. These messages should go through the Server or Agent Message Stream Interface.

Figure 3-6 on page 96 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 3-7 on page 97 shows how the display manager and the other processes communicate.

Figure 3-6 Overview of the Interface APIs

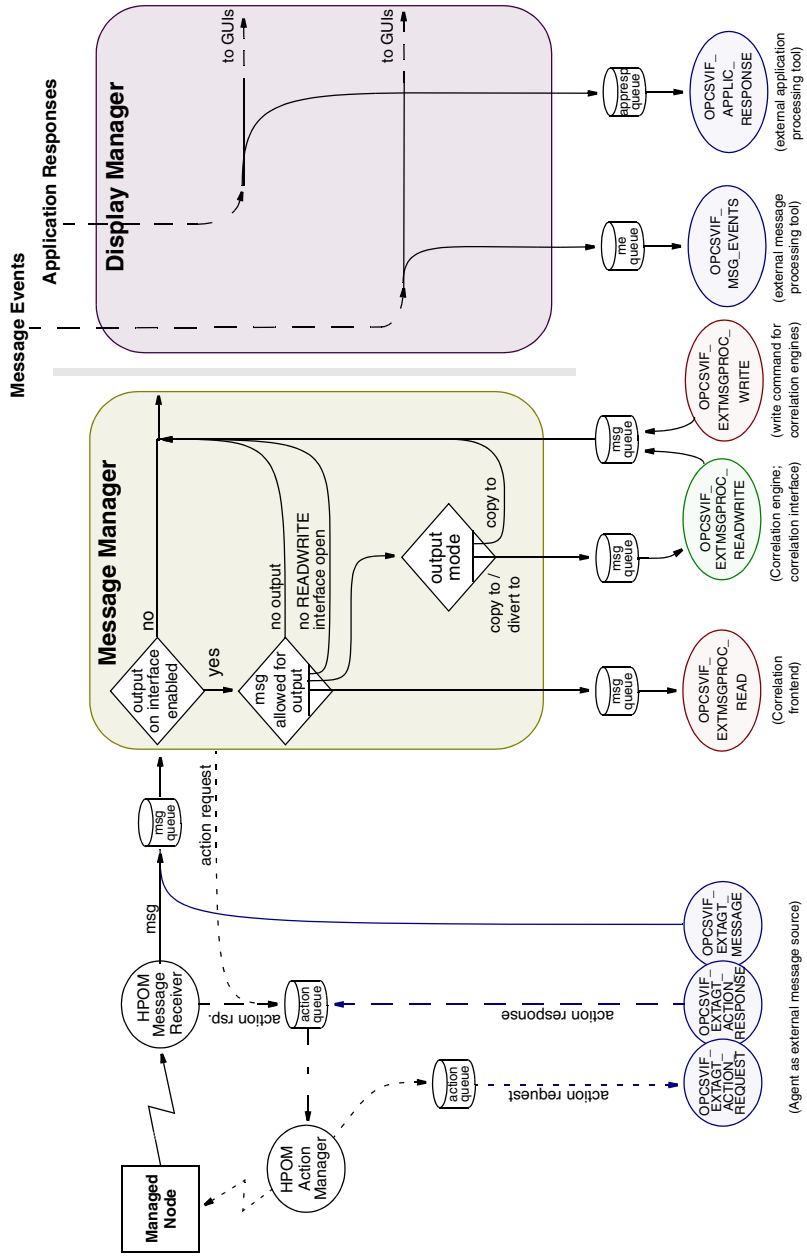
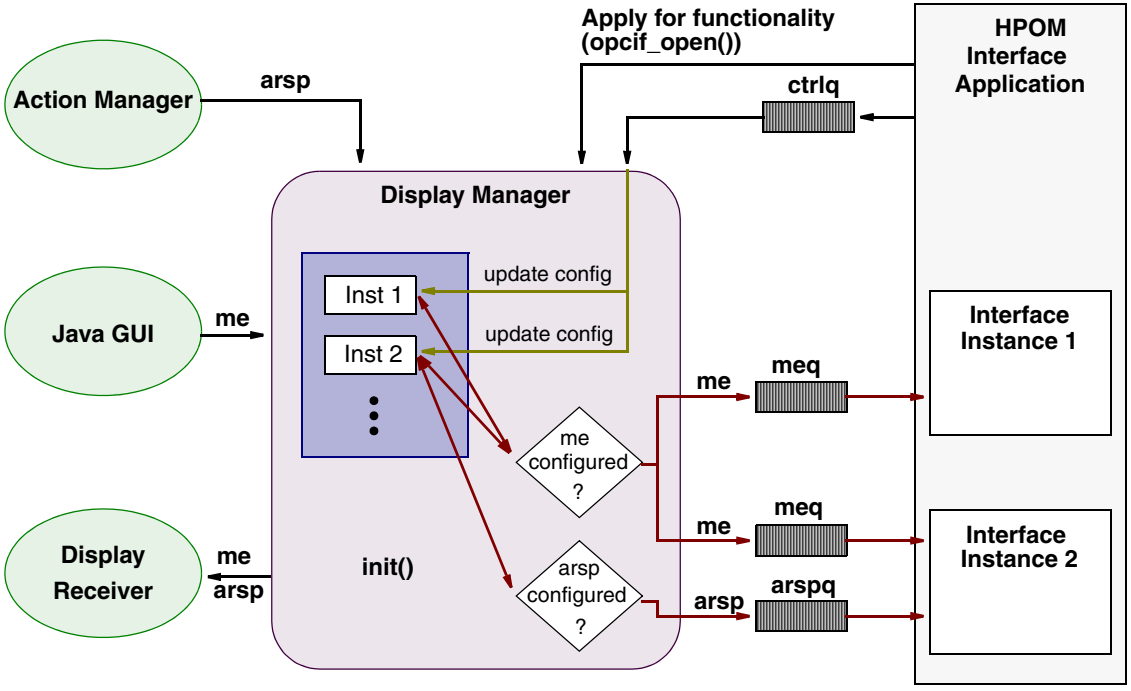


Figure 3-7 Communication With the Display Manager



Key:

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

The HPOM Operator APIs

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

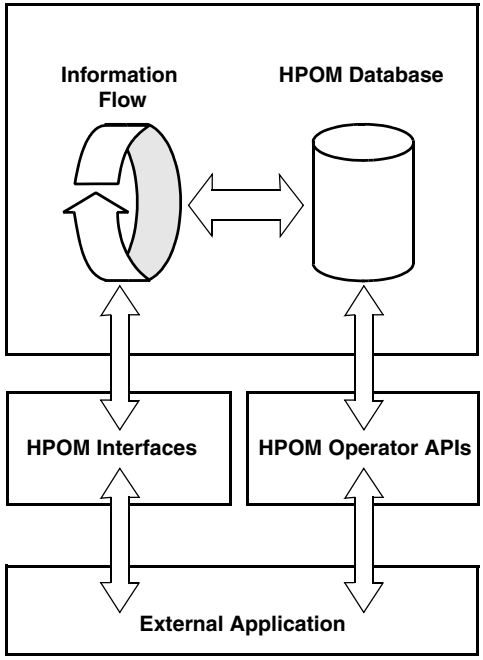
API	Description
HPOM Data API	API to get/set information in HPOM data structures. This API is used for: • HPOM Data Structures • Containers • Iterators
HPOM Interface API	API to connect to the HPOM Interfaces; it contains the functions: `opcif_*()` and `opcreg_*()`
Server Message API	API on the management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

HPOM Interfaces and HPOM Operator API

Figure 3-8 shows the different methods by which the HPOM Interfaces and the HPOM Operator API access HPOM data. The HPOM 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 HPOM Operator APIs, however, access the HPOM database directly, without opening an interface instance, to perform the required action(s).

Figure 3-8 Comparison of the HPOM Service APIs and the HPOM Operator APIs



The HPOM Configuration APIs

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

API	Description
Connection API	API to connect to and disconnect from the HPOM database; it contains the functions `opc_connect()` and `opc_disconnect()`.
Application Configuration API	API to configure HPOM applications and application groups; it contains the functions `opcappl_*()`.
Application Group Configuration API	API to configure HPOM application groups; it contains the functions `opcapplgrp_*()`.
Category Configuration API	API to configure HPOM categories; it contains the functions `opcinstrum_*()`, `opcnode_*()`, and `opcpolicy_*()`.
Instruction Text Interface Configuration API	API to configure HPOM instruction text interfaces; it contains the functions `opcinstruction_*()`.
Message Group Configuration API	API to configure HPOM message groups; it contains the functions `opcmsggrp_*()`.

Message Regroup Condition Configuration API

API to configure HPOM message regroup condition; it contains the functions `opcmsgregrp_*()`.

Node Configuration API

API to configure HPOM nodes and node groups; it contains the functions `opcnode_*()` and `opcnodegrp_*()`.

Node Hierarchy Configuration API

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

Notification Schedule Configuration API

API to configure HPOM notification schedules; it contains the functions `opcnotischedule_*()`.

Notification Service Configuration API

API to configure HPOM notification services; it contains the functions `opcnotiservice_*()`.

Policy Configuration API

API to configure message source policies and policy groups; it contains the functions `opcpolicy_*()` and `opcpolicygrp_*()`.

Trouble Ticket Configuration API

API to configure HPOM interfaces to trouble ticket services; it contains the functions `opctoubleticket_*()`.

User Profile Configuration API	API to configure HPOM user profiles; it contains the functions `opcprofile_*()`.
User Configuration API	API to configure HPOM users; it contains the functions `opcuser_*()`.
Distribution API	Distribution API to distribute agent configuration to managed nodes.
Synchronization API	Accesses, modifies, and synchronizes HPOM configuration data.
Pattern Matching API	API to match a string against a pattern.

Summary of HPOM API Functions

Functions of the HPOM Data API

Functions to Manipulate HPOM Data Structures

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

Table 3-3 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()`	Releases previously allocated memory
`opcdata_generate_id()`	Generates an HPOM 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 3-3 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_remove_cma()`	Removes custom message attributes from a message
`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_type()`	Returns the opcdata type

Functions of the HPOM Iterator

See the man page *opciter(3)* for more information about these functions.

Table 3-4 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 HPOM Data Structures

Table 3-5 Overview of the HPOM Data Structures

HPOM Data Type	Description
OPCDTYPE_ACTION_REQUEST	Defines an HPOM action request
OPCDTYPE_ACTION_RESPONSE	Contains the response of a previously started action
OPCDTYPE_ANNOTATION	Contains a message annotation
OPCDTYPE_APPLIC	Defines an HPOM application
OPCDTYPE_APPLIC_RESPONSE	Contains the response of a previously started HPOM application
OPCDTYPE_APPL_CONFIG	Contains the configuration information of an HPOM application
OPCDTYPE_APPL_GROUP	Contains the configuration information of an application group
OPCDTYPE_ASSIGNMENT	Defines the relationship between two assignable objects and the type of the assignment.
OPCDTYPE_CATEGORY_INFO	Defines an HPOM category. Contains category information for policies and instrumentation.
OPCDTYPE_DBMAINT	Contains a definition of the HPOM database maintenance configuration.
OPCDTYPE_CONTAINER	Contains a list of HPOM objects
OPCDTYPE_CONFIG_EVENT	Describes a configuration change event, retrieved via the Configuration Stream Interface.
OPCDTYPE_EMPTY	Creates an empty structure
OPCDTYPE_INFORM_USER	Contains the name of the user and the message
OPCDTYPE_INSTR_IF	Contains the definition of an HPOM Instruction Text Interface.
OPCDTYPE_LAYOUT_GROUP	Defines an HPOM node layout group
OPCDTYPE_MESSAGE	Contains information about the attributes of an HPOM message

Table 3-5 Overview of the HPOM Data Structures (Continued)

HPOM Data Type	Description
OPCDTYPE_MESSAGE_EVENT	Contains a message event
OPCDTYPE_MESSAGE_GROUP	Contains the name and description of an HPOM message group
OPCDTYPE_MESSAGE_ID	Contains an HPOM message ID
OPCDTYPE_MGMTSV_CONFIG	Contains the definition of an HPOM management server configuration.
OPCDTYPE_MONITOR_MESSAGE	Contains the most necessary information for a monitor value.
OPCDTYPE_NODE	Contains limited information about an HPOM managed node.
OPCDTYPE_NODE_CONFIG	Contains a complete definition of an HPOM managed node with all its attributes.
OPCDTYPE_NODE_GROUP	Defines an HPOM node group
OPCDTYPE_NODEHIER	Defines an HPOM node hierarchy
OPCDTYPE_NOTI_SCHEDULE	Contains the definition of an HPOM Notification Schedule
OPCDTYPE_NOTI_SERVICE	Contains the definition of an HPOM Notification Service.
OPCDTYPE_POLICY	Contains a complete definition of a policy, with references to policy bodies.
OPCDTYPE_POLICY_GROUP	Contains a complete definition of a policy group, with relations between it and its assigned objects.
OPCDTYPE_POLICY_TYPE	Contains a complete definition of a policy type, as known on the management server.
OPCDTYPE_REGROUP_COND	Contains configuration information about a message regroup condition.

Table 3-5 Overview of the HPOM Data Structures (Continued)

HPOM Data Type	Description
OPCDTYPE_TEMPLATE_INFO	Contains the most necessary information to define an HPOM message source template or template group
OPCDTYPE_USER_CONFIG	Contains the configuration of an HPOM user
OPCDTYPE_USER_RESP_ENTRY	Contains the responsibility information of an HPOM user

Functions of the HPOM Service APIs

Functions to Access the HPOM Interface

Table 3-6 Overview of the HPOM 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()`	Reads information from the specified interface
`opcif_register()`	Registers for certain attributes
`opcif_unregister()`	Unregisters 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 3-7 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()`	Releases 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 3-8 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_get()`	Gets detailed information about a message
`opcmsg_get_instructions()`	Gets instructions for a message
`opcmsg_get_msgids()`	Gets the current message ID to a given original message ID
`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_set_owner()`	Sets the owner of a message to the specified operator
`opcmsg_start_auto_action()`	Starts an automatic action
`opcmsg_start_op_action()`	Starts an operator-initiated action
`opcmsg_unack()`	Unacknowledges a message
`opcmsg_unbuffer()`	Unbuffers a pending message.

Functions of the Agent Message API

Functions to Send/Acknowledge Messages

Table 3-9 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 3-10 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 Application API

Function to Start an HPOM Application

Table 3-11 Overview of the Application API Function

Function Call	Description
`opcapp_start()`[a]	Starts a specified HPOM application on the given nodes in the container.

a. This function is obsolete with HPOM version A.08.10. Use the function `opcappl_start()` instead.

Functions of the Connection API

Functions to Connect to the Management Server

Table 3-12 Overview of the Connection API Functions

Function Call	Description
`opc_connect()`	Connect to the HPOM database to get access.
`opc_disconnect()`	Disconnect from the HPOM database.

Functions of the Application Configuration API

Functions to Configure HPOM Applications

Table 3-13 Overview of the Application API Function

Function Call	Description
`opcappl_add()`	Adds an HPOM application
`opcappl_delete()`	Deletes an HPOM application
`opcappl_get()`	Gets the full configuration of an HPOM application
`opcappl_get_list()`	Gets a list of all configured applications
`opcappl_modify()`	Modifies an HPOM application
`opcappl_start()`	Starts an HPOM application

Functions of the Application Group Configuration API

Functions to Configure HPOM Application Groups

Table 3-14 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 Category Configuration API

Functions to Configure HPOM Categories

Table 3-15 Overview of the Category API Function

Function Call	Description
`opcinstrum_add_categories()`	Adds categories to the HPOM database
`opcinstrum_del_categories()`	Deletes categories from the HPOM database
`opcinstrum_get_categories()`	Gets a list of all categories that exist in the HPOM database
`opcinstrum_get_category()`	Gets the full configuration of a specified category
`opcinstrum_modify_categories()`	Changes the name or description of categories in the HPOM database
`opcpolicy_assign_categories()`	Assigns a list of categories to a specified policy
`opcpolicy_deassign_categories()`	Deassigns a list of categories from a specified policy
`opcpolicy_get_categories()`	Gets a list of all categories that are assigned to a specified policy
`opcnode_assign_categories()`	Assigns a list of categories to a specified node
`opcnode_deassign_categories()`	Deassigns a list of categories from a specified node
`opcnode_get_categories()`	Gets a list of categories that are assigned to a specified node

Functions of the Instruction Text Interface Configuration API

Functions to Configure HPOM Instruction Text Interfaces

Table 3-16 Overview of the Instruction Text Interface API Function

Function Call	Description
`opcinstruction_add()`	Adds a new instruction text interface to the HPOM database
`opcinstruction_del()`	Deletes an instruction text interface from the HPOM database
`opcinstruction_get()`	Gets the full configuration of a specified instruction text interface from the HPOM database
`opcinstruction_modify()`	Modifies the configuration of a specified instruction text interface in the HPOM database

Functions of the Message Group Configuration API

Functions to Configure HPOM Message Groups

Table 3-17 Overview of the Message Group Configuration API Functions

Function Call	Description
`opcmsggrp_add()`	Adds a message group to the HPOM database
`opcmsggrp_delete()`	Removes a message group from the HPOM 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 HPOM Message Regroup Conditions

Table 3-18 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 HPOM Managed Nodes

Table 3-19 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 HPOM node bank hierarchy
`opcnode_assign_templates()`	Assigns policies or policy groups to a node
`opcnode_assign_policy()`	Assigns policies to a node.
`opcnode_assign_policy_group()`	Assigns policy groups to a node.
`opcnode_deassign_templates()`	Deassigns policies and policy groups from a node
`opcnode_deassign_policy()`	Deassigns policies from a node
`opcnode_deassign_policy_group()`	Deassigns policy 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 HPOM 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
`opcnode_modify_defaults()`	Changes the default values of the given network and machine type

Function to Configure HPOM Node Groups

Table 3-20 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 policies or policy groups to a node group
`opcnodegrp_assign_policy()`	Assigns policies to a node group
`opcnodegrp_assign_policy_group()`	Assigns policy groups to a node group
`opcnodegrp_deassign_nodes()`	Deassigns nodes from a node group
`opcnodegrp_deassign_templates()`	Deassigns policies or policy groups from node group
`opcnodegrp_deassign_policy()`	Deassigns policies from node group
`opcnodegrp_deassign_policy_group()`	Deassigns policy group 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 HPOM Node Hierarchies

Table 3-21 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 Notification Schedule Configuration API

Function to Configure HPOM Notification Schedules

Table 3-22 Overview of the Notification Schedule Configuration API Functions

Function Call	Description
`opcnotischedule_add()`	Adds a new notification schedule to the HPOM database
`opcnotischedule_del()`	Deletes a notification schedule from the HPOM database
`opcnotischedule_get()`	Gets the full configuration of a notification schedule from the HPOM database
`opcnotischedule_get_list()`	Gets a list of notification schedules from the HPOM database
`opcnotischedule_modify()`	Modifies the configuration of a notification schedule in the HPOM database

Functions of the Notification Service Configuration API

Function to Configure HPOM Notification Services

Table 3-23 Overview of the Notification Service Configuration API Functions

Function Call	Description
`opcnotiservice_add()`	Adds a new notification service to the HPOM database
`opcnotiservice_del()`	Deletes a notification service from the HPOM database
`opcnotiservice_get()`	Gets the full configuration of a notification service from the HPOM database
`opcnotiservice_get_list()`	Gets a list of notification service from the HPOM database
`opcnotiservice_modify()`	Modifies the configuration of a notification service in the HPOM database

Functions of the Policy Configuration API

Function to Configure HPOM Policies

Table 3-24 Overview of the Policy Configuration API Functions

Function Call	Description
`opcpolicy_add()`	Uploads a policy from a header file into the HPOM database
`opcpolicy_assignment_mode_set()`	Modifies the assignment mode of an existing policy to policy group, node, or node group assignment
`opcpolicy_copy()`	Creates a copy of an existing policy with a new name and/or version
`opcpolicy_copy_assignments()`	Copies assignments for one policy to another
`opcpolicy_delete()`	Deletes a policy or policies and associated bodies from the HPOM database
`opcpolicy_edit()`	Downloads the policy body referenced by its ordinal number to the filesystem and produces the edit string for it
`opcpolicy_edit_body()`	Downloads a policy body referenced by its suffix to the filesystem and produces the edit string for it
`opcpolicy_get()`	Retrieves policy header and bodies from the HPOM database and writes them into files
`opcpolicy_get_data()`	Retrieves policy header data from the HPOM database
`opcpolicy_get_list()`	Retrieves the list of all policies present in the HPOM database
`opcpolicy_get_list_by_type()`	Retrieves the list of all policies of provided type that are present in the HPOM database
`opcpolicy_header_create()`	Creates a policy header with provided data, or creates a policy header for an existing policy

Table 3-24 Overview of the Policy Configuration API Functions (Continued)

Function Call	Description
`opcpolicy_list_assignments()`	Retrieves all assignments for a policy or a policy group
`opcpolicy_modify()`	Changes policy header data in the HPOM database for an existing policy
`opcpolicy_update_assignments()`	Sets assignments for a policy or a policy group
`opctempl_delete()`*	Deletes an existing policy from the HPOM database
`opctempl_get_list()`	Gets a list of all policies from the HPOM database
`opctemplfile_add()`*	Adds policies to the HPOM database
`opctemplfile_get()`	Gets details of the policy and writes them into a file
`opctemplfile_modify()`*	Modifies already existing HPOM policies.

* This function is not implemented in the current HPOM release.

Functions to Configure HPOM Policy Groups

Table 3-25 Overview of the Policy Configuration API Functions

Function Call	Description
`opcpolicygrp_add()`	Uploads a policy group, and optionally policies assigned to it, from a file to the HPOM database
`opcpolicygrp_assign_policies()`	Assigns policies to a policy group
`opcpolicygrp_copy()`	Creates a copy of an existing policy group in the HPOM database with a possibility of merging the contents of existing groups
`opcpolicygrp_create()`	Adds a new policy group to the database
`opcpolicygrp_deassign_policies()`	Deassigns policies from a policy group
`opcpolicygrp_delete()`	Deletes a policy group from the HPOM database
`opcpolicygrp_get()`	Downloads a policy group, and, optionally, policies assigned to it, from the HPOM database to the filesystem
`opcpolicygrp_get_data()`	Retrieves policy group data from the HPOM database
`opcpolicygrp_get_list()`	Retrieves the list of all policy groups and their contents from the HPOM database
`opcpolicygrp_list_assignments()`	Retrieves all assignment data for a specified policy group from the HPOM database
`opcpolicygrp_modify()`	Modifies policy group data in the HPOM database
`opctemplgrp_assign_templates()`	Assigns policies to a policy group
`opctemplgrp_deassign_templates()`	Deassigns policies from a policy group
`opctemplgrp_delete()`	Deletes a policy group from the HPOM database
`opctemplgrp_get()`	Gets the configuration of a policy group from the HPOM database

Table 3-25 Overview of the Policy Configuration API Functions (Continued)

Function Call	Description
`opctemplgrp_get_templates()`	Gets a list of policies / policy groups assigned to a policy group
`opctemplgrp_modify()`	Modifies an already existing policy group

Functions of the Trouble Ticket Configuration API

Functions to Configure HPOM Trouble Tickets

Table 3-26 Overview of the Trouble Ticket API Function

Function Call	Description
`opctroubleticket_get()`	Retrieves information about HPOM configuration for trouble ticket systems from the HPOM database
`opctroubleticket_set()`	Configures HPOM to forward messages to a trouble ticket system and stores the configuration in the HPOM database

Functions of the User Profile Configuration API

Functions to Configure HPOM User Profiles

Table 3-27 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 HPOM Users

Table 3-28 Overview of the User API Function

Function Call	Description
`opcuser_add()`	Adds an HPOM user
`opcuser_assign_applgrps()`	Assigns application groups to an HPOM user
`opcuser_assign_appls()`	Assigns applications to an HPOM user
`opcuser_assign_nodehier()`	Assigns a node hierarchy to an HPOM user
`opcuser_assign_profiles()`	Assigns profiles to an HPOM user
`opcuser_assign_resps()`	Assigns responsibilities to an HPOM user
`opcuser_deassign_applgrps()`	Deassigns application groups from an HPOM user
`opcuser_deassign_appls()`	Deassigns applications from an HPOM user
`opcuser_deassign_profiles()`	Deassigns profiles from an HPOM user
`opcuser_deassign_resps()`	Deassigns responsibilities from an HPOM user
`opcuser_delete()`	Deletes an HPOM user
`opcuser_get()`	Gets the full configuration of an HPOM 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 HPOM 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 HPOM user

Functions of the Distribution API

Functions to Distribute Configuration to Managed Nodes

Table 3-29 Overview of the Distribution API Function

Function Call	Description
`opc_distrib()`	Distributes the HPOM agent configuration to managed nodes.

Functions of the Server Synchronization API

Functions to Modify and Update Configuration Data

Table 3-30 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
`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

Functions of the Pattern Matching API

Functions to Match Strings against a Patterns

Table 3-31 Overview of the Pattern Matching API Function

Function Call	Description
`opcpat_match()`	Returns the result of an HPOM pattern-matching operation and lists the matched strings, assigned to variables

Using APIs in Internationalized Environments

All HPOM 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.

4 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 HPOM 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 151.

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 148.

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 Mozilla. 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 141 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 141.

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 HPOM 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 at the following locations:

- UNIX: `/opt/OV/www/htdocs/ito_op/`
- Windows: `C:\Program Files\HP Operations for UNIX\/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 HPOM 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 *HPOM 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 Java2SDK1.5 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 143 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 144 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:

- `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 146 and "To Run the Client" on page 147.

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\HPOM 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 151.

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\\HPOM 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 149.

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 144 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 148.

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 150 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 151 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 2003/XP 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 4-1 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 4-1 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 4-2 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 4-2 Overview of OV_JGUI_JavaBridge Class Methods (Continued)

Methods	Description
`toString()`	Prints out the current `OV_JGUI_JavaBridge` instance.

5 Integrating with Service Navigator

In This Chapter

HP Service Navigator is an add-on component of HP Operations Manager. 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 HPOM 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

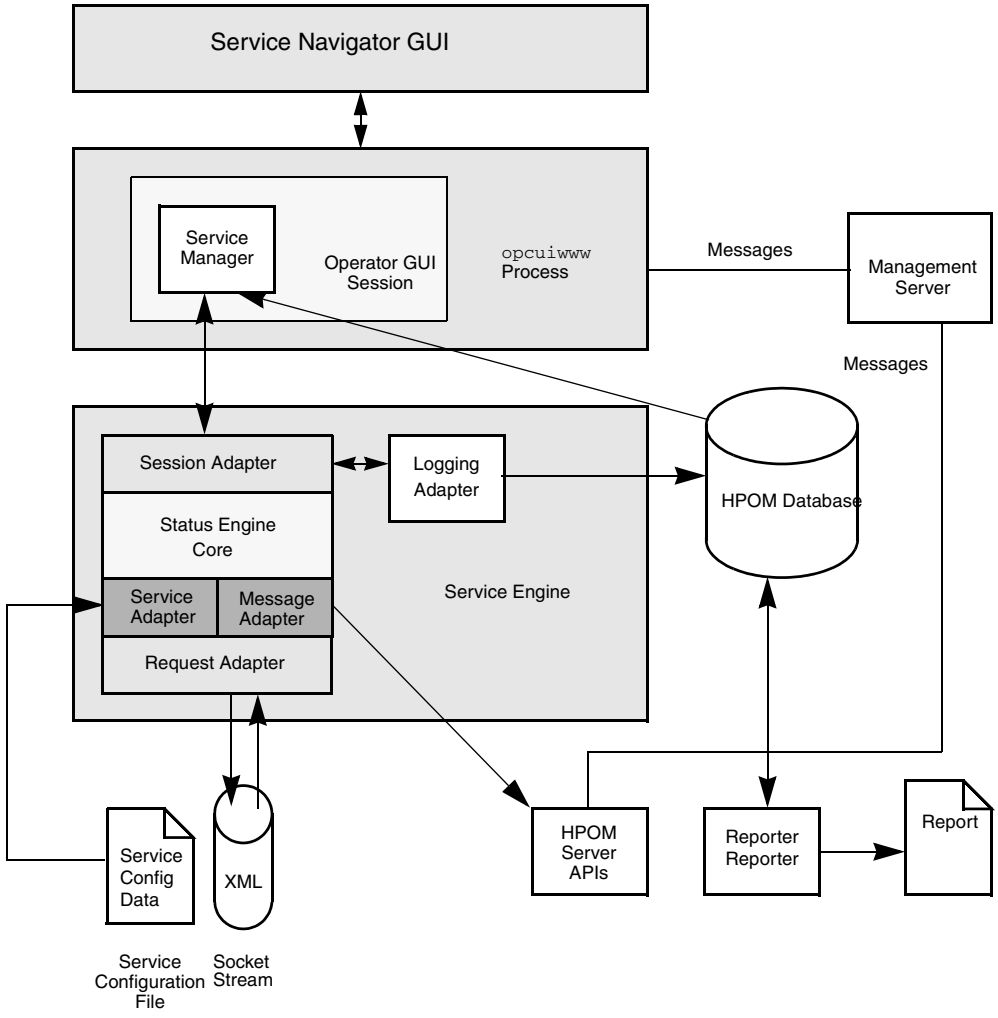
Use an ANSI C++ (aCC) compiler. See also the man page *opcsvc_api(3)* for more information.

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 5-1 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 HPOM 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 *HPOM Developer's Reference* for more information about the XML syntax.

The C++ APIs

The HPOM 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 *HPOM 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.

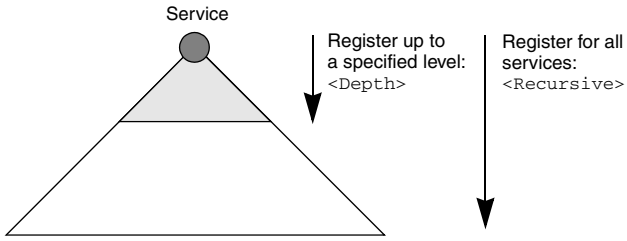
The Registration Conditions

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 5-2 on page 164 for an illustration and examples.

Figure 5-2 Registration Condition for Service Structure



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>
```

- ❏ Register depending on severity:
  - <Ascend>
  - <Decend>

  See Figure 5-3 on page 166 for an illustration and example.

  Service Status transition can be either ascending (from lower to a higher severity) or descending (from a higher to a lower severity). A service status change event is seNt whenever a new severity is reached for which a client has registered.
- ❏ Register depending on patterns in service name

- `<Pattern>`

See Figure 5-4 on page 166 for an illustration and example.

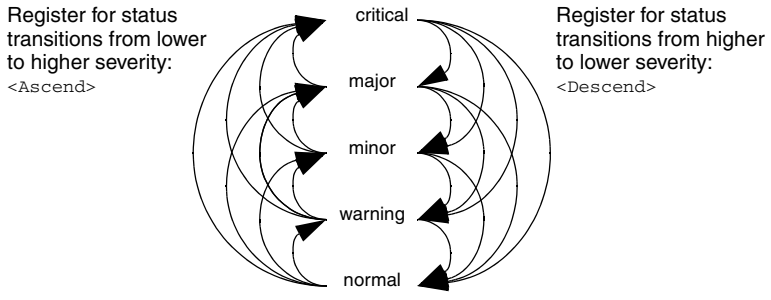
Patterns on service names in the registration allow you to select services based on a pattern instead of only recursively.

❑ Register depending on attributes (especially service attributes):

- parameter exists
- parameter has value

- parameter value matches pattern

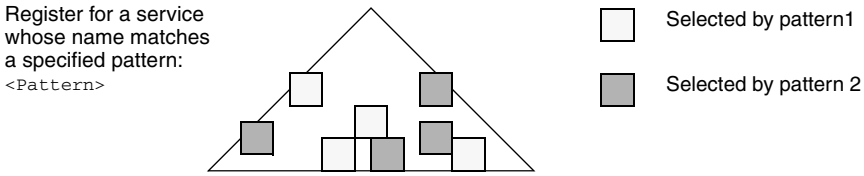
Figure 5-3 Registration Condition for Service Status Transitions



For example:

```
<Registration>
   <RegCondition>
      <ServiceRef>svc_1</ServiceRef>
      <Ascend/>
   </RegCondition>
   <RegCondition>
      <ServiceRef>svc_2</ServiceRef>
      <Descend/>
   </RegCondition>
</Registration>
```

Figure 5-4 Registration Condition for Service Name Patterns



For example:

```
<Registration>
   <ServiceStruct>
      <Name>name</Name>
      <Recursive/>
   </ServiceStruct>
   <ServiceStruct>
      <Pattern>ora_.*tblspc_*</Pattern>
   </ServiceStruct>
</Registration>
```

6 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 HPOM will play an important role in the creation of the integration package.

This chapter describes the structure of HPOM 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 HPOM.

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 HPOM administrator's GUI. For example, define new policies, or add new conditions to existing policies; define message groups, node groups, default operator(s); set up applications, etc.
   - In addition, it may sometimes be necessary to place files at predefined locations in the file trees maintained by HPOM, for example, if monitor scripts, or programs for performing certain actions are to be distributed by HPOM.
2. Download the configuration information into a file tree.

   Use the command `opccfgdwn(1M)` to download predefined configuration sets from the command line. This facility enables you to choose the full HPOM configuration or selected parts that are relevant to the integration package.
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 HPOM management server, for example, if APIs on the management server are used, installation scripts, man pages, etc.

The HPOM configuration upload command `opccfgupld(1M)` can be used at the customer's site to upload the configuration information into the local HPOM installation.

Structure of HPOM Configuration Files

The behavior and capabilities of HPOM 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 6-1.

Figure 6-1 Structure of an HPOM Configuration File Tree

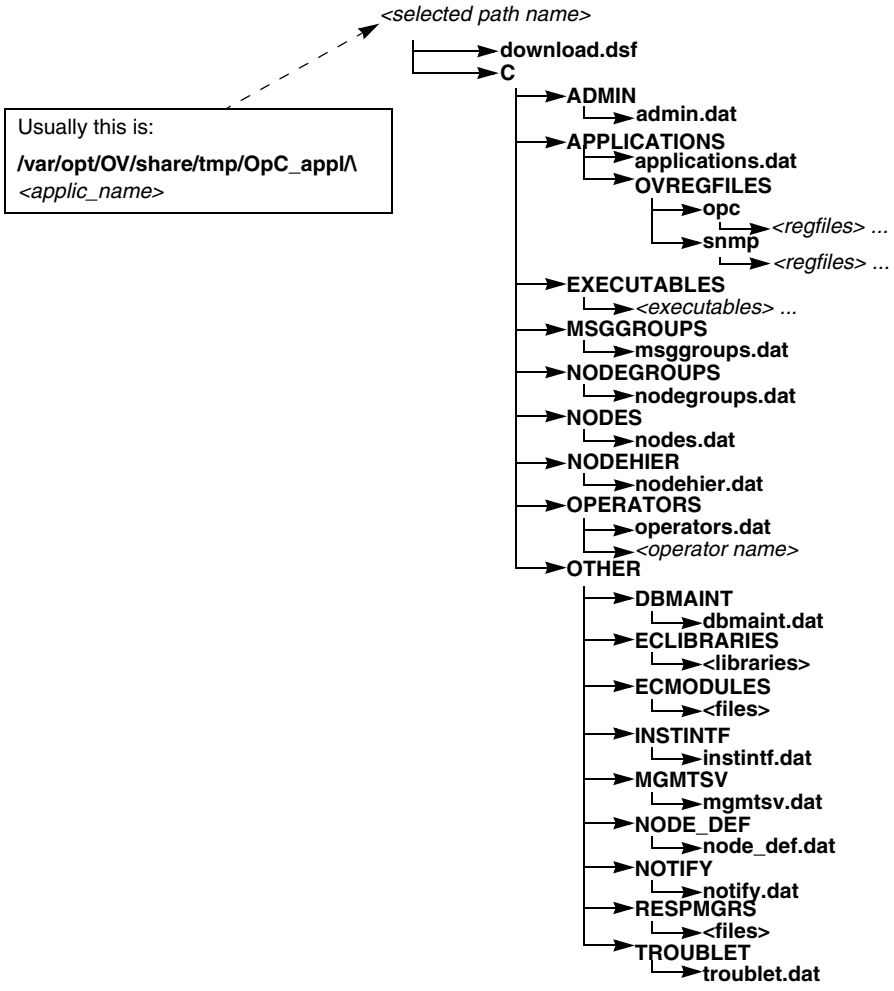
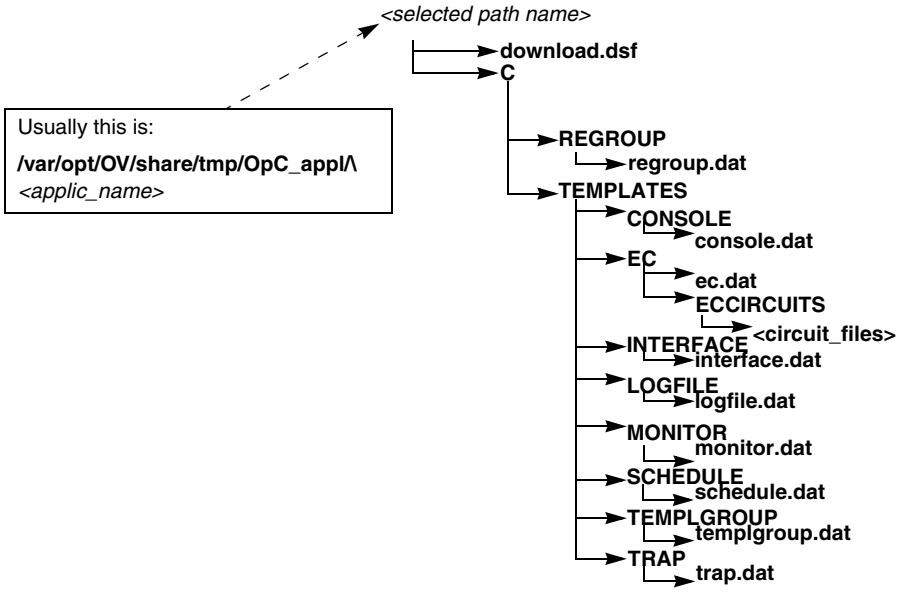


Figure 6-2 Structure of an HPOM Configuration File Tree (cont.)



Downloading Configuration Information

You can download configuration information 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" ;
LOGFILE_TEMPLATE "dflt_SecEvLog" ;
LOGFILE_TEMPLATE "dflt_SysEvLog" ;
```

This specification file is required as a parameter by the `opccfgdwn(1M)` command. For more information, see the *opccfgdwn(1m)* man page.

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 HPOM, 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 HPOM internal database at any time 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 HPOM 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 HPOM to distribute the program/script to the managed node(s)

  HPOM will distribute the program/script to directories on the managed node which are maintained by HPOM and which are automatically in the command search path of the HPOM agents.

This section explains how to use HPOM to distribute programs/scripts.

NOTE Distribution by HPOM may only be used for small executables, preferably small wrapper scripts for the reasons stated below.

Prepare an executable for distribution by HPOM as follows:

1. Place the executable into a source directory from which HPOM will fetch it during the configuration downloading. On the HPOM 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, for example, NCS_RPC, 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.

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, HPOM 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 HPOM to copy the executables into an HPOM-maintained directory on the customer's management server.
5. Advise the HPOM administrator, for example, in the documentation of the integration package, to distribute the new configuration information to the required managed nodes.

Warnings

It is important to consider the following issues when distributing executables using HPOM:

- ❏ 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 HPOM 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 HPOM

  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 11i 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 machine running HP-UX 11i, 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 HPOM database or not:

-replace	Overwrite any information that already exists in the HPOM internal database.
-add	Insert additional information that is different from information that already exists in the HPOM 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 HPOM internal database:

1. Stop the HPOM management server processes using the following command:

   #opcsv -stop

2. 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.

3. After successfully uploading the configuration information, restart the HPOM management server processes:

   /opt/OV/bin/OpC/opcsv -start

4. Distribute the new configuration information to the managed nodes using the `opcragt -distrib` command. For the available distribution options, see the *opcragt(1m)* man page.

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 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 HPOM 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 HPOM 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 HPOM 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 HPOM 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 HPOM 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 HPOM 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. HPOM, therefore, lets you insert new conditions at specific positions within the list of conditions.

Assume that, after analyzing the default template for intercepting HPOM 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 HPOM default policy 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.

1. Stop the HPOM management server processes using the following command:

   #opcsv -stop

2. 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 HPOM adds information to an existing template, it compares the conditions to be added with the existing conditions. If HPOM 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.

1. Start the HPOM management server processes using the following command:

   #opcsv -start

2. 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 HPOM Configuration Files

In This Chapter

This appendix provides a detailed description of the syntax used by the configuration download command `opccfgdwn(1M)` to store HPOM 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 HPOM Syntax Rules

The following syntax rules apply to all HPOM 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 Policy Bodies

This section describes the syntax used in configuration files that describe policy bodies for the following message sources: logfiles, SNMP traps, and messages passed to HPOM 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 policy distribution time:

Table A-1 Policy Syntax Versions

Syntax Version	HPOM Version	Keyword	Description
8	7.00	`CUSTOM`	Keyword for custom message attributes.
7	6.00	`SUPP_DUPL_IDENT_OUTPUT_MSG`	Keyword for suppress identical output messages option.
6	Used by HP Operations Manager for Windows.		
5	5.00	`CONDITION_ID`	Keyword for condition ID.
		`MSGKEY` `MSGKEYRELATION`	Keywords for message correlation.
		`SERVICE_NAME`	Keyword for service name mapping for the HP Service Navigator.

Table A-1 Policy Syntax Versions (Continued)

Syntax Version	HPOM Version	Keyword	Description
4	4.00	ECS CIRCUIT_FILE	Keywords for event correlation.
		SCHEDULE	Keywords for scheduled actions.
3	3.00	Major Minor	Keyword for new severity states.
		MPI_AGT_DIVERT_MSG MPI_AGT_COPY_MSG	Keywords for the agent message stream API.
		UNICODE ANSI_L1 ANSI_JP OEM_L1 OEM_US OEM_JP	Character sets used by Windows 2003/XP managed nodes
2	2.00	The syntax version is set to 2 (also used in version 2.xx) if none of the keywords mentioned above is used.	

Since the syntax descriptions for the policy bodies of all these policies share most rules, they are presented here as one grammar. The following restrictions apply:

- a policy body of a logfile policy must fit the syntax definition <logsources>
- a policy body of an SNMP trap policy must fit the syntax definition <snmpsources>
- a policy body of an HPOM message interface policy must fit the syntax definition <opcsources>
- a policy body of an event correlation policy must fit the syntax definition <ecsources>
- a policy body of a scheduled action policy must fit the syntax definition <schedsources>
- a policy body of a monitoring policy must fit the syntax definition <advmonsource>

The policy body grammar is as follows:

```
file:           ε |
                SYNTAX_VERSION syntax_number |
                file logsource |
                file snmpsource |
                file csmsource |
                file monsource |
                file advmonsource |
                file schedsource |
                file ecsource |
                file wbemsource

syntax_number: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11

logsource:      LOGFILE <string (name)> DESCRIPTION <string
                (description)> logdefopts conditions

snmpsource:     SNMP <string (name)> DESCRITPION <string
                (description)> snmpdefopts snmpconditions

csmsource:      OPCMSG <string (name)> DESCRIPTION <string
                (description)> csmdefopts conditions

monsource:      MONITOR <string (name)> DESCRIPTION <string
                (description)> mondefopts monconditions
```

```
advmonsource:  ADVMONITOR <string (name)> DESCRIPTION
               <string (description)> advmondefaults
               advmonsourcedef advmonconditions

schedsource:   SCHEDULE <string (name)> DESCRIPTION <string
               (description)> schedsetopts

ecsource:      ECS <string (name)> DESCRIPTION <string
               (description)> ecopts ecver CIRCUIT_FILE
               <string (file)> circuit

wbemsource:    WBEM <string (name)> DESCRIPTION <string
               (description)> wbemdefopts wbemconditions

logdefopts:    ε | logdefopts logdefault | logdefopts
               logoption | logdefopts sourceoption

logdefault:    stddefault | NODE node

logoption:     LOGPATH <string (path to logfile)> |
               EXEFILE <string (path to file to execute)> |
               READFILE <string (path to file containing
               logfile paths)> |
               INTERVAL <string (time between logfile
               checks)> |
               CHSET <string (character set of the logfile)> |
               FROM_LAST_POS |
               FIRST_FROM_BEGIN |
               NO_LOGFILE_MSG |
               CLOSE_AFTER_READ

snmpdefopts:   ε | snmpdefopts stddefault | snmpdefopts
               sourceoption

snmpconditions: ε |
               snmpconditions MSGCONDITIONS snmpmsgconds |
               snmpconditions SUPPRESSCONDITIONS
               snmpsuppressconds |
               snmpconditions SUPP_UNM_CONDITIONS
               snmpsupp_unm_conds

snmpmsgconds:  ε |
               snmpmsgconds DESCRIPTION <string
               (description)> condsuppdupl condition_id
               CONDITION snmpconds SET sets
```

```
snmpsuppressconds: ε |
                snmpsuppressconds DESCRIPTION <string
                (description)> condition_id CONDITION
                snmpconds

snmpsupp_unm_conds: ε |
                snmpsupp_unm_conds DESCRIPTION <string
                (description)> condition_id CONDITION
                snmpconds

snmpconds:      ε |
                snmpconds $e <string (enterprise)> |
                snmpconds $G <number (generic trap)> |
                snmpconds $S <number (specific trap)> |
                snmpconds $(<number (variable)>) pattern |
                snmpconds NODE nodelist

csmdefopts:     ε | csmdefopts stddefault | csmdefopts
                sourceoption

mondefopts:     ε | mondefopts mondefault | mondefopts
                monoption | mondefopts sourceoption

mondefault:     stddefault | NODE node

monoption:      INTERVAL <string (time between checks)> |
                MONPROG <string (path to monitor executable)> |
                MIB <string (MIB variable)> |
                MIB <string (MIB variable)> NODE node |
                EXTERNAL |
                MINTHRESHOLD |
                MAXTHRESHOLD |
                GEN_BELOW_THRESHOLD |
                GEN_BELOW_RESET |
                GEN_ALWAYS |
                AUTOMATIC_MSGKEY

monconditions: ε |
                monconditions MSGCONDITIONS monmsgconds |
                monconditions SUPPRESSCONDITIONS
                monsuppressconds |
                monconditions SUPP_UNM_CONDITIONS
                monsupp_unm_conds
```

```
monmsgconds:    ε |
                monmsgconds DESCRIPTION <string> condition_id
                CONDITION monconds SET sets

monsuppressconds: ε |
                monsuppressconds DESCRIPTION <string>
                condition_id CONDITION monconds

monsupp_unm_conds: ε |
                monsupp_unm_conds DESCRIPTION <string>
                condition_id CONDITION monconds

monconds:       ε |
                monconds THRESHOLD numval duration |
                monconds RESET numval |
                monconds OBJECT pattern

advmondefaults: ε | advmondefaults sourceoption |
                advmondefaults stddefault | advmondefaults
                NODE node | advmondefaults advmonoption

advmonoption:   INTERVAL <string (time between checks)> |
                INSTANCEMODE ALL | INSTANCEMODE SAME |
                INSTANCEMODE ONCE |
                MULTISOURCE |
                INSTANCERULES |
                AUTOMATIC_MSGKEY |
                AUTOMATIC_MSGKEY <string (default message key)> |
                MINTHRESHOLD |
                MAXTHRESHOLD |
                GEN_BELOW_THRESHOLD |
                GEN_BELOW_RESET |
                GEN_ALWAYS |
                SCRIPTTYPE <string (type of script)> |
                DDF DATASOURCE <string> |
                DDF OBJECT <string>

advmonsourcedef: ε |
                advmonsourcedef PROGRAM <string (name)>
                DESCRIPTION <string (description)> advmonprog |
                advmonsourcedef EXTERNAL <string (name)>
                DESCRIPTION <string (description)> ddf |
                advmonsourcedef NTPERFMON <string (name)>
                DESCRIPTION <string (description)> advmonperfmon |
                advmonsourcedef SNMP <string (name)>
```

```
                DESCRIPTION <string (description)> advmonsnmp |
                advmonsourcedef MEASUREMENT <string (name)>
                DESCRIPTION <string (description)> advmonme |
                advmonsourcedef CODA <string> DESCRIPTION
                <string (description)> advmonme |
                advmonsourcedef WBEM <string (description)>
                DESCRIPTION <string (description)> advmonwbem

advmonprog:     MONPROG <string (path to executable)> ddf

advmonperfmon:  OBJECT <string (name)> COUNTER <string>
                INSTANCE <string> ddf

advmonsnmp:     MIB <string (MIB variable)> ddf

advmonme:       COLLECTION <string> metrics |
                COLLECTION <string> GUID <string (UUID)>
                metrics |
                DATASOURCE <string> COLLECTION <string>
                metrics

advmonwbem:     NAMESPACE <string> CLASS <string> ATTRIBUTE
                <string> instancefilter ddf |
                WMI_USERNAME <string> WMI_PASSWORD <string>
                NAMESPACE <string> CLASS <string> ATTRIBUTE
                <string> instancefilter ddf

instancefilter: ε | INSTANCE_FILTER <string>

ddf:            DDF DATASOURCE <string> OBJECT <string> METRIC
                <string>

metrics:        ε |
                metrics METRIC <string> metricguid
                useforinstance

metricguid:     ε | GUID <string (UUID)>

useforinstance: ε | USEFORINSTANCE

advmonconditions: ε |
                advmonconditions tMSGCONDITIONS
                advmonmsgconds |
                advmonconditions tSUPPRESSCONDITIONS
                advmonsuppressconds |
                advmonconditions tSUPP_UNM_CONDITIONS
                advmonsupp_unm_conds
```

```
advmonmsgconds: ε |
                advmonmsgconds instancerule tDESCRIPTION
                <string (description)> condition_id CONDITION
                advmonconds advmonmsgsets

instancerule:   ε |
                INSTANCERULE <string> ID <string> |
                INSTANCERULE <string>

advmonmsgsets: ε |
                advmonmsgsets SETSTART sets |
                advmonmsgsets SETCONT sets |
                advmonmsgsets SETEND sets

advmonsuppressconds: ε |
                advmonsuppressconds DESCRIPTION <string>
                condition_id CONDITION advmonconds

advmonsupp_unm_conds: ε |
                advmonsupp_unm_conds DESCRIPTION <string>
                condition_id CONDITION advmonconds

advmonconds:    ε |
                advmonconds THRESHOLD numval duration |
                advmonconds THRESHOLD condscript duration |
                advmonconds RESET numval |
                advmonconds RESET condscript |
                advmonconds OBJECT pattern |
                advmonconds OBJECT condscript

condscript:     SCRIPTTYPE <string> SCRIPT <string> |
                SCRIPT <string>

duration:       ε | FOR <string (condition duration)>

numval:         <integer number> | <floating number>

schedsetopts:   ε |
                schedsetopts DISABLED |
                schedsetopts TEMPLATE_ID <string  (UUID of the
                template)> |
                schedsetopts VERSION <number> |
                schedsetopts SCRIPTTYPE <string (type of
                script)> SCRIPT <string (actual script)> |
                schedsetopts SCHEDPROG <string (path to
                executable to run)> |
                schedsetopts USER <string (username)> |
```

```
                    schedsetopts USER <string (username)>
                    PASSWORD <string  (password)> |
                    schedsetopts MONTH <string (month)> |
                    schedsetopts MONTHDAY <string (day in the
                    month)> |
                    schedsetopts WEEKDAY <string (day in the
                    week)> |
                    schedsetopts HOUR <string (hour)> |
                    schedsetopts MINUTE <string (minute)> |
                    schedsetopts TIMEZONE_VALUE <string (timezone)> |
                    schedsetopts YEAR <number> |
                    schedsetopts INTERVAL <string (time between
                    actions)> |
                    schedsetopts LOGLOCAL |
                    schedsetopts SEND_OUTPUT |
                    schedsetopts TIMEZONE_TYPE tz_type |
                    schedsetopts BEFORE SET sets |
                    schedsetopts FAILURE SET sets |
                    schedsetopts SUCCESS SET sets

ecopts:             ε |
                    ecopts DISABLED |
                    ecopts TEMPLATE_ID <string (UUID of the
                    template)> |
                    ecopts VERSION <number (template version)> |
                    ecopts ECS_LOG_INPUT |
                    ecopts ECS_LOG_OUTPUT

ecver:              VERIFIED | UNVERIFIED

circuit:            ε | circuit <string>

wbemdefopts:        ε |
                    wbemdefopts wbemdefault |
                    wbemdefopts wbemoption |
                    wbemdefopts sourceoption

wbemdefault:        stddefault | NODE node

wbemoption:         NAMESPACE <string (WBEM namespace)> |
                    CLASS <string (WBEM class)> |
                    WITHIN <string (interval)> |
                    WHERE_CLAUSE <string (where clause)> |
                    QUERY_LANGUAGE <string (language for query)> |
                    QUERY <string (query)> |
```

```
                    INSTANCE_CREATION_EVENT |
                    INSTANCE_MODIFICATION_EVENT |
                    INSTANCE_DELETION_EVENT |
                    CLASS_CREATION_EVENT |
                    CLASS_MODIFICATION_EVENT |
                    CLASS_DELETION_EVENT |
                    NAMESPACE_CREATION_EVENT |
                    NAMESPACE_MODIFICATION_EVENT |
                    NAMESPACE_DELETION_EVENT |
                    INTERVAL <string (interval)>

wbemconditions: ε |
                    wbemconditions MSGCONDITIONS wbemmsgconds |
                    wbemconditions SUPPRESSCONDITIONS
                    wbemsuppressconds |
                    wbemconditions SUPP_UNM_CONDITIONS
                    wbemsupp_unm_conds

wbemmsgconds:  ε |
                    wbemmsgconds DESCRIPTION <string
                    (description)> condsuppdupl condition_id
                    CONDITION wbemconds SET sets

wbemsuppressconds: ε |
                    wbemsuppressconds DESCRIPTION <string>
                    condition_id CONDITION wbemconds

wbemsupp_unm_conds: ε |
                    wbemsupp_unm_conds DESCRIPTION <string
                    (description)> condition_id CONDITION
                    wbemconds

wbemconds:      ε |
                    wbemconds <string (condition name)> ~= pattern |
                    wbemconds <string (condition name)> wbemop
                    wbemval

wbemop:         ==  |  !=  |  >=  |  >  |  <  |  <=

wbemval:        <string> | <number (floating)> | <number
                (int)>

condefopts:     ε |
                    condefopts stddefault |
                    condefopts sourceoption
```

```
conditions:     ε |
                conditions MSGCONDITIONS msgconds |
                conditions SUPPRESSCONDITIONS suppressconds |
                conditions SUPP_UNM_CONDITIONS supp_unm_conds

msgconds:       ε |
                msgconds DESCRIPTION <string> condsuppdupl
                condition_id CONDITION conds SET sets

suppressconds:  ε |
                suppressconds DESCRIPTION <string>
                condition_id CONDITION conds

supp_unm_conds: ε |
                supp_unm_conds DESCRIPTION <string>
                condition_id CONDITION conds

condsuppdupl:   ε |
                SUPP_DUPL_COND suppdupl |
                SUPP_DUPL_IDENT suppdupl |
                SUPP_DUPL_IDENT_OUTPUT_MSG suppdupl

conds:          ε |
                conds SEVERITY severities |
                conds NODE nodelist |
                conds APPLICATION <string> |
                conds MSGGRP <string> |
                conds OBJECT <string> |
                conds TEXT pattern

suppdupl:       <string> |
                <string> RESEND <string> |
                <string> COUNTER_THRESHOLD <number> |
                <string> COUNTER_THRESHOLD <number>
                RESET_COUNTER_INTERVAL <string> |
                <string> RESEND <string> COUNTER_THRESHOLD
                <number> |
                <string> RESEND <string> COUNTER_THRESHOLD
                <number> RESET_COUNTER_INTERVAL <string> |
                COUNTER_THRESHOLD <number> |
                COUNTER_THRESHOLD <number>
                RESET_COUNTER_INTERVAL <string>

stddefault:     SEVERITY severity |
                APPLICATION <string> |
                MSGGRP <string> |
```

```
                    OBJECT <string> |
                    SERVICE_NAME <string> |
                    MSG_KEY <string> |
                    HELPTEXT <string (instruction text)> |
                    HELP <string (instruction UUID)> |
                    INSTRUCTION_TEXT_INTERFACE <string> |
                    INSTRUCTION_PARAMETERS <string>

sourceoption:       LOGMATCHEDMSGCOND | LOGMATCHEDSUPPRESS |
                    LOGUNMATCHED | FORWARDUNMATCHED |
                    UNMATCHEDLOGONLY | 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 | ICASE |
                    DISABLED |
                    SUPP_DUPL_COND suppdupl |
                    SUPP_DUPL_IDENT suppdupl |
                    SUPP_DUPL_IDENT_OUTPUT_MSG suppdupl |
                    SEPARATORS <string> |
                    TEMPLATE_ID <string> |
                    TEMPLATE_VERSION <number>

severities:         ε | severities severity

severity:           Unknown | Normal | Warning  | Critical | Major
                    | Minor

nodelist:           nodelist node | node

node:               IP <string (IP address)> |
                    IP <string (IP address)> <string (node name)> |
                    OTHER <string (variable or other)>

tz_type:            MGR_LOCAL | AGT_LOCAL | FIX

sets:               ε | sets set

set:                SEVERITY severity |
                    NODE node |
                    APPLICATION <string (application to which
                    message relates)> |
                    MSGGRP <string (message group)> |
                    OBJECT <string (object to which message
                    relates)> |
                    MSGTYPE <string (type of message)> |
```

```
                    TEXT <string (message text)> |
                    SERVICE_NAME <string (name of the service to
                    which the message relates)> |
                    MSGKEY <string (message key)> |
                    MSGKEYRELATION ACK pattern |
                    CUSTOM <string (name of the custom attribute)>
                    <string (value of the custom attribute)> |
                    SERVERLOGONLY |
                    AUTOACTION action |
                    OPACTION action |
                    TROUBLETICKET acknowledge |
                    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 (text of the instruction
                    message)> |
                    HELP <string (UUID of the stored instruction
                    message)> |
                    INSTRUCTION_TEXT_INTERFACE <string (name of
                    instruction text interface)> |
                    INSTRUCTION_PARAMETERS <string (parameters
                    for instruction text interface)>

condition_id:       ε | CONDITION_ID <string (UUID)>

action:             <string (path to executable)> actionnode
                    annotate acknowledge msgsendmode signature

actionnode:         ε | ACTIONNODE node

acknowledge:        ε | ACK

msgsendmode:        ε | SEND_MSG_AFTER_LOC_AA msgsendok
                    msgsendfailed

msgsendok:          ε | SEND_OK_MSG logonly

msgsendfailed:      ε | SEND_FAILED_MSG

logonly:            ε | LOGONLY

signature:          ε | SIGNATURE <string (signature)>
```

```
pattern:      <string> separators icase

separators:   ε | SEPARATORS <string (separators)>

icase:        ε | ICASE

chset:        ε | ASCII | ACP1250 | ACP1251 | ACP1252 |
              ACP1253 | ACP1254 | ACP1255 | ACP1256 |
              ACP1257 | ACP1258 | NT_ANSI_JP | NT_OEM_JP |
              ACP874 | NT_OEM_L1 | NT_ANSI_LP | NT_OEM_US |
              NT_UNICODE | OEMCP437 | OEMCP720 | OEMCP737 |
              OEMCP775 | OEMCP850 | OEMCP852 | OEMCP855 |
              OEMCP857 | OEMCP860 | OEMCP861 | OEMCP862 |
              OEMCP863 | OEMCP864 | OEMCP865 | OEMCP866 |
              OEMCP869 | OEMCP932 | ROMAN8 | ISO8859 |
              ISO88591 | ISO885910 | ISO885911 | ISO885913 |
              ISO885914 | ISO885915 | ISO88592 | ISO88593 |
              ISO88594 | ISO88595 | ISO88596 | ISO88597 |
              ISO88598 | ISO88599 | TIS620 | UCS2 | EBCDIC |
              SJIS | EUC | EUCJP | EUCKR | EUCTW | GB2312 |
              BIG5 | CCDC | UTF8
```

Policy Body Examples

Example of an HPOM Logfile Policy Body

The following example is a policy body for a Logfile Entry policy:

```
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 HPOM Message Source Specification

The following example is a policy body for the HPOM 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 HPOM 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."
```

Example of an HPOM Monitor Policy Body

The following text shows the definition of a monitor for which the HPOM 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 "HPOM"
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 HPOM Monitor Agent (`opcmona`).

Syntax for Message Pattern Matching

Table A-2 shows the components of the HPOM 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 HPOM 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: [ ] < > | ^ $ they must be prefaced with a backslash (\ ) to mask their usual function. If ^ 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: [ ] < > | ^ $ A special character preceded by \ results in an expression that matches the special character itself. Note that because ^ 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 (^) is used as the first character of the pattern, only expressions discovered at the beginning of lines are matched. For example, "^ab" 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 HPOM 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 HPOM 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 HPOM 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 HPOM 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: HPOM'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 HPOM 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 HPOM 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 HPOM 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

HPOM 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 HPOM Application Configuration File

The following is an example of an Application Configuration File for an HPOM application. This file is part of the HPOM default configuration.

```
APPLICATION "HPOM Status"
   SYMBOL "Software:Process"
   START_ON_SEL_FLAG FALSE
   LABEL "HPOM 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 HPOM Object Names

Syntax checks are automatically performed when entering information in the corresponding fields of the GUI. For more information about length limitations of HPOM object names, see the *HPOM Reporting and Database Schema*.

B About HPOM Man Pages

In this Appendix

This appendix describes the man pages available in the following areas:

- Man Pages in HPOM
- Man Pages for HPOM APIs
- Man Pages for HP Operations Service Navigator
- Man Pages for the HPOM Developer's Kit APIs

Accessing and Printing Man Pages

You can access the HPOM man pages from the command line, from online help, or in HTML format on your management server.

To Access an HPOM Man Page from the Command Line

To access an HPOM man page from the command line, enter the following:

```
man <manpagename>
```

To Print a Man Page from the Command Line

To print an HPOM 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 HPOM 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 HPOM

This section describes man pages in HPOM.

Table B-1 HPOM Man Pages

Man Page	Description
call_sqlplus.sh(1)	Calls SQL*Plus.
inst.sh(1M)	Installs HPOM software on managed nodes.
inst_debug(5)	Debugs an installation of the HP Operations agent software.
ito_op(1M)	Launches the HPOM Java-based operator or Service Navigator GUI.
ito_op_api_cli(1M)	Enables calling the Java GUI Remote APIs.
opcbackup_offline(1M)	Interactively saves the HPOM environment for Oracle.
opcbackup_offline(5)	Backs up the HPOM configuration.
opc_chg_ec(1M)	Changes circuit names in event correlation (EC) policies in the HPOM database.
opcdelmsg(1M)	Removes messages from the Message Manager queue with management server processes running. Only messages matching all specified criteria will be deleted.
opcrecover_offline(1M)	Interactively recovers the HPOM environment for Oracle.
opcrecover_offline(5)	Recovers the HPOM 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 HP Operations agent.

Table B-1 HPOM Man Pages (Continued)

Man Page	Description
opcadddbf(1M)	Adds a new datafile to an Oracle tablespace.
opcagt(1M)	Administers agent processes on a managed node.
opcagtutil(1M)	Parses the agent platform file, and performs operations with extracted data.
opccfgdwn(1M)	Downloads configuration data from the database to flat files.
opccfgout(1M)	Configures condition status variables for scheduled outages in HPOM.
opccfgupld(1M)	Uploads configuration data from flat files into the database.
opccfguser(1M)	Configures HPOM for UNIX operators and is used for assigning user profiles, unassigning user profiles and configuring the responsibility matrix.
opccltconfig(1M)	Configures HPOM client filesets.
opcconfig(1M)	Configures an HPOM 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 HPOM database.
opcdbinit(1M)	Initializes the database with the default configuration.
opcdbinst(1M)	Creates or destroys the HPOM database scheme.
opcdbpwd(1M)	Changes the password of the HPOM database user opc_op.
opcdbsetup(1M)	Creates the tables in the HPOM database.
opcdcode(1M)	Views HPOM encrypted policy files.
opcerr(1M)	Displays instruction text for HPOM error messages.

Table B-1 HPOM Man Pages (Continued)

Man Page	Description
opcgetmsgids(1m)	Gets message IDs to an original message ID.
opchbp(1M)	Switches heartbeat polling of managed nodes on or off.
opchistdwn(1M)	Downloads HPOM history messages to a file.
opchistupl(1M)	Uploads history messages into the HPOM database.
opcinstrumcfg(1M)	Manages category info in the filesystem and database level simultaneously.
opcinstrumdwn(1M)	Copies all instrumentation files together which would be deployed to an HPOM HTTPS agent.
opcmack(1)	Acknowledges an HPOM message by specifying the message ID.
opcmom(4)	Provides an overview of HPOM MoM functionality.
opcmomchk(1)	Checks syntax of MoM policies.
opcmon(1)	Forwards the value of a monitored object to the HPOM monitoring agent on the local managed node.
opcmsg(1)	Submits a message to HPOM.
opcownmsg(1M)	Sets, unsets, and changes HPOM messages ownership.
opcpat(1)	Tests a program for HPOM pattern matching.
opcragt(1M)	Remotely administers agent services for HPOM on a managed node.
opcskm(3)	Manages secret keys.
opcsqlnetconf(1M)	Configures the HPOM database to use an Net8 connection.
opcsv(1M)	Administers HPOM manager services.
opcsw(1M)	Sets the software status flag in the HPOM database.
opcswitchuser(1M)	Switches the ownership of the HP Operations agents.

Table B-1 HPOM Man Pages (Continued)

Man Page	Description
`opcpolicy(1M)`	Maintains policies in files.
`opcpolicy(1M)`	Enables and disables policies.
`opctmpldwn(1M)`	Downloads and encrypts HPOM message source policies.
`opcwall(1)`	Sends a message to currently logged in HPOM users.
`ovocomposer(1M)`	Performs tasks related to OV Composer.
`ovocomposer(5)`	Describes the Correlation Composer, an HPOM event correlation feature.
`ovtrap2opc(1M)`	Converts the `trapd.conf` file and the HPOM policy file.

Man Pages for HPOM APIs

This section describes man pages for HPOM application program interfaces (APIs).

Table B-2 HPOM API Man Pages

Man Page	Description
`opcmon(3)`	Forwards the value of a monitored object to the HPOM monitoring agent on the local managed node.
`opcmsg(3)`	Submits a message to HPOM.

Man Pages for HP Operations Service Navigator

This section describes man pages for the HP Operations Service Navigator.

Table B-3 Service Navigator Man Pages

Man Page	Description
`opcservice(1M)`	Configures HP Operations Service Navigator.
`opcsvcattr (1M)`	Add, change or remove service attributes.
`opcsvcconv(1M)`	Converts service configuration files of HP Operations Service Navigator from the previous syntax to the Extensible Markup Language (XML).
`opcsvcdwn(1M)`	Downloads service status logs of HP Operations Service Navigator to a file.
`opcsvcterm(1M)`	Emulates an interface to HP Operations 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 Operations Service Navigator into the HPOM database.

Man Pages for the HPOM Developer's Kit APIs

This section describes man pages for the HPOM Developer's Kit application program interfaces (APIs).

Table B-4 HPOM Developer's Toolkit Man Pages

Man Page	Description
`msiconf(4)`	Configures the HPOM 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 HPOM.
`opc_distrib(3)`	Distributes the HP Operations agent configuration.
`opcagtmon_send(3)`	Forwards the value of a monitored object to HPOM.
`opcagtmsg_api(3)`	Handles messages on HP Operations agents.
`opcanno_api(3)`	Manages HPOM message annotations.
`opcapp_start(3)`	Starts an HPOM application.
`opcappl_api(3)`	Configures and starts HPOM applications.
`opcapplgrp_api(3)`	Configures HPOM application groups.
`opcconf_api(3)`	Gets HPOM configuration.

Table B-4 HPOM Developer's Toolkit Man Pages (Continued)

Man Page	Description
opcdata(3)	Accesses the attributes of the HPOM data structure.
opcdata_api(3)	Describes how to access the HPOM data structure using the HPOM Data API.
opcif_api(3)	API to work with the HPOM Message Stream Interface.
opciter(3)	HPOM iterator to step through opcdata container.
opcmsg_api(3)	Manages HPOM messages.
opcmsggrp_api(3)	Manages HPOM message groups.
opcmsgregrpcond_api(3)	Creates and modifies HPOM message regroup conditions.
opcnode_api(3)	Configures HP Operations managed nodes.
opcnodegrp_api(3)	Configures HP Operations node groups.
opcnodehier_api(3)	Configures HP Operations node hierarchies.
opcprofile_api(3)	Configures HPOM user profiles.
opcregcond(3)	Accesses fields of the HPOM registration condition structure.
opcsvc_api(3)	C++ classes for Service Navigator.
opctempl_api(3)	Configures HPOM message source policies.
opctempfile_api(3)	Configures HPOM policies using policy files.
opctemplgrp_api(3)	Configures HPOM policy groups.
opctransaction_api(3)	Starts, commits, and rolls back transactions.
opcuser_api(3)	Configures HPOM users.
opcversion(3)	Returns the string of the HPOM library that is currently used.

Index