HP OpenView ServiceCenter Automation for HP OpenView Operations

For the UNIX Operating System

Software Version: 1.3

User Guide

Document Release Date: June 2006
Software Release Date: June 2006

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation. UNIX® is a registered trademark of The Open Group.

HP OpenView ServiceCenter, HP OpenView ServiceCenter Automation, and HP OpenView Operations are registered trademarks of Hewlett-Packard Company.

Contents

1 Introduction

CHAPTER

This chapter provides an introduction to HP OpenView ServiceCenter Automation (SCAuto) for HP Openview Operations (OVO).

It covers the following topics:

Overview

Welcome to the *HP ServiceCenter Automation for HP OpenView Operations User Guide*. This guide provides instructions on how to implement the interface between OVO and ServiceCenter.

SCAuto for OVO allows you to automate the process of creating, updating, and closing trouble tickets in ServiceCenter, based on OVO Message Stream Interface (MSI) and Message Event Interface (MEI) events. It has the capability of annotating, owning, and acknowledging OVO messages from modifications done on ServiceCenter incident tickets.

This product is part of the suite of SCAuto interface products that integrate ServiceCenter with premier Network and Systems Management tools. The interface is based on event messages sent over TCP connection to the ServiceCenter server. Additional information about SCAuto can be found in the *ServiceCenter Automation Applications for Windows NT and UNIX Guide*.

Note: HP Openview IT/Operations (IT/O) has been renamed to HP OpenView Operations (OVO) for UNIX. Note that the name change is not yet fully implemented across the OVO software and you may encounter the former name (IT/O) in this guide. The names OVO and IT/O are synonymous throughout this guide.

Knowledge requirements

This guide assumes the reader has:

- Working knowledge of ServiceCenter applications, ServiceCenter Client/Server, and the OVO graphical user interface, as well as a basic understanding of ServiceCenter applications and Event Services. While some procedures for these applications are explained, others are referenced. Refer to the appropriate ServiceCenter documentation for a more detailed explanation.
- Familiarity with OVO and its components including the Network Node Manager (NNM). Working knowledge of the operating system environment in which you are working (such as a GUI or text-based environment).
- (As an Administrator) a thorough knowledge of the operating system where ServiceCenter, SCAuto, and the SCAuto for OVO product will be installed and implemented.

Determining current product version

Knowing the current version of your SCAuto for OVO product is valuable when contacting HP Customer Support and deciding when to upgrade. From a ServiceCenter client, refer to the About menu to determine the current version of ServiceCenter.

Each of the executables `scfromitoMEI`, `scfromitoMSI`, `scfromitoTTI` and `sctoito` can be given a command line argument of "-v" to display their current versions and copyright information.

SCAuto for OVO

SCAuto for OVO offers integration between ServiceCenter applications and OVO. The interface of these two applications allows systems management functions of OVO to be extended and enhanced by the Incident Management functions of ServiceCenter.

This integration benefits customers of ServiceCenter and OVO by providing these features:

- A more robust environment for production IT operations
- Automated ticketing functions supported by the Incident Management processes
- The ability to structure and organize the real-time responses supported by the enterprise systems management functions.

The sections that follow give a high-level view of the operational concepts in this integration. These section are:

- ServiceCenter
- OVO
- SCAuto for OVO
- Operational concepts
- Planning your ServiceCenter and OVO integration
- OVO business logic topics
- ServiceCenter business logic topics
- SCAuto for OVO business logic topics

ServiceCenter

Core applications

ServiceCenter contains four core applications:

- Incident Management, which is a incident-tracking tool integrated with knowledge tools to speed resolution.
- Configuration Management, which maintains an operational database of assets used in the enterprise.
- Change Management, which allows you to manage the evolution of the enterprise to meet changing needs and requirements as they arise.
- Problem Management, which a problem-tracking tool integrated with knowledge tools to speed resolution.

Configuration Management tracks what the enterprise was expected to be, Incident Management works with the current state, Change Management offers a way to manage towards the desired final product, and Problem Management works with final resolution of issues and problems.

Additional applications

ServiceCenter provides additional applications that support and enhance the core applications.

The additional ServiceCenter applications are described next:

- Service Management provides a call-based front end applications.
- Service Level Management (SLM) offers significant value with service desk automation based on service level agreements (SLAs).
- Request Management maintains catalogs of services and items and organizes the delivery of these.
- Schedule Maintenance enables you to set up and execute recurring tasks.

OVO

OVO is the foundation of HP OpenView's operational control of IT resources. It is a part of HP's OpenView suite of management applications.

OVO features data, event, and process level integration with other OpenView applications. It provides a central management console for enterprise systems management actions. It is built around a robust framework architecture, with an event console at the core.

Functional areas

OVO has several functional areas that are built from the event messages handled by the event console. These functional areas are described next:

- Message groups and message templates directly impact the events.
- User groups collect OVO operators and allocate authorization roles.
- Node groups perform a similar function for the managed resources.
- Application groups structure the OVO monitored applications into organized, manageable components.

Network Node Manager

OVO incorporates the OpenView Network Node Manager (NNM) product. It performs SNMP-based network monitoring. It acts as a perfect companion for the event console of OVO because it generates event messages from incidents and faults on the network.

About SCAuto for OVO

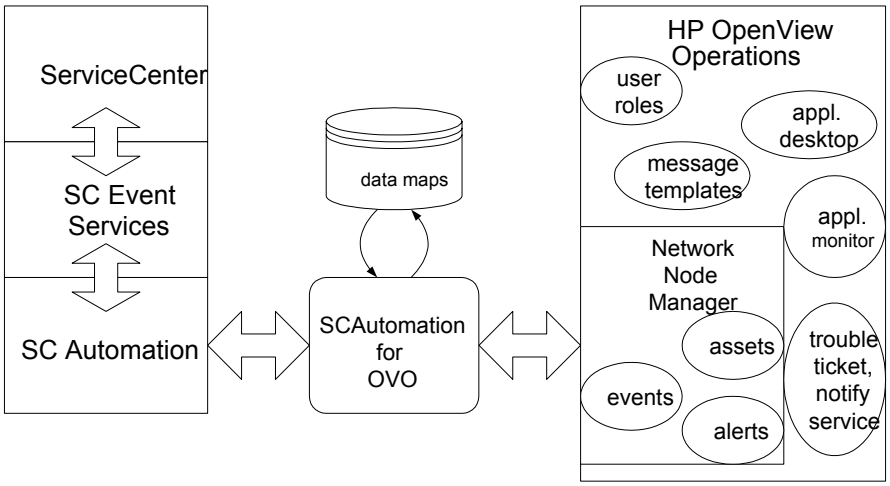
SCAuto for OVO is designed in a modular way. Specific modules perform specific data processing functions. There are three separate modules that connect to three specific APIs:

- The Message Stream Interface (MSI) is a registration API that delivers notice whenever new events arrive at the event console.
- The Message Event Interface (MEI) is the interface that delivers messages upon status changes to existing events.
- The Trouble Ticket Interface (TTI) facilitates the help desk integration feature of message templates. When event messages match the conditions of templates, incident tickets may be opened automatically. The use of this API minimizes the configuration of the SCAuto for OVO adapter and allows the user to configure OVO. This creates an extension to make OVO interact with ServiceCenter automatically.

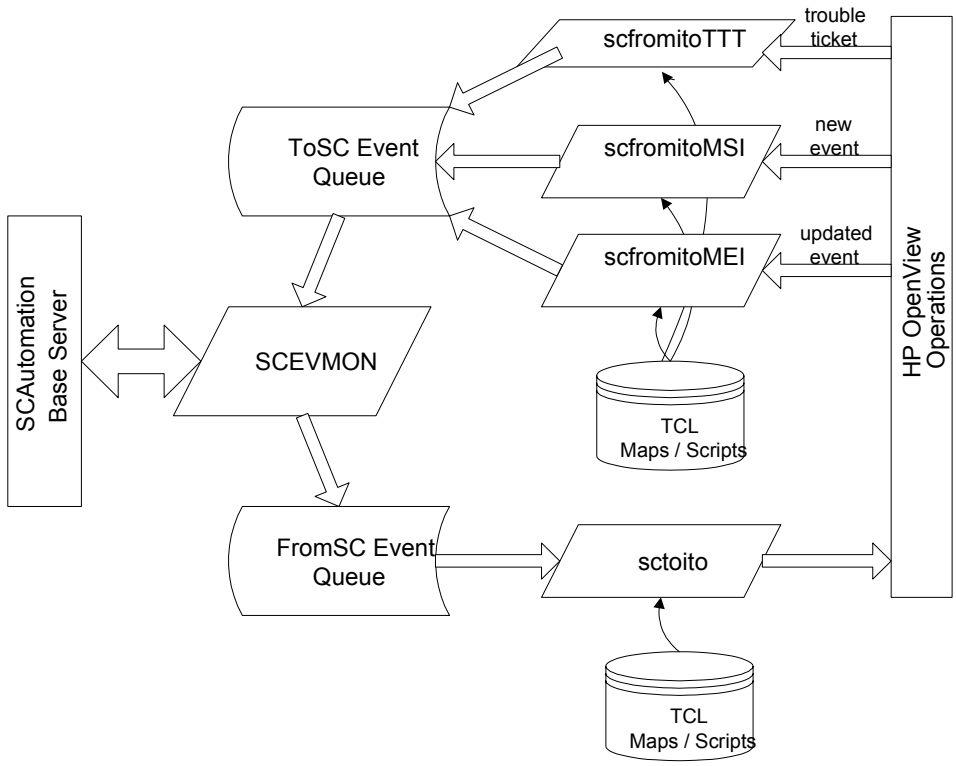
The MSI and the TTI function in a similar way. By registering with OVO at these APIs, new events received by OVO are output to SCAuto for OVO. This allows ServiceCenter to open new tickets. Subsequent actions against the event message (such as annotation or acknowledgment of the event) are output to SCAuto for OVO through the MEI API.

Individual APIs are used for individual data message exchange, but the combination of two separate APIs is used throughout the duration of the data exchange. Of the combinations, the TTI & MEI integration is preferred, and the MSI & MEI integration is also valid.

The following figure shows the architectural block diagram of the integration and the adapter



The following figure shows the modular components of SCAuto for OVO in a functional block diagram.



In addition to the programmatic connections, other components within OVO are configured by the installation of SCAuto for OVO. For example, a ServiceCenter message group is configured to provide a preconfigured view of the event messages delivered by ServiceCenter to OVO. Upon installation, the function is available for selection and activation without any configuration or customization required.

SCAuto for OVO also uses components that are a part of the standard SCAutomation Software Development Kit (SDK). Key modules from the SDK are the event monitor (scevmon), the bi-directional event queues (ToSC and FromSC), and the event maps that formulate the conversion of incoming or outgoing messages.

Operational concepts

The background of the static components was discussed previously. This section discusses the dynamic nature of the integration, which is a critical part of understanding the overall integration.

OVO is essentially a real-time management tool, while ServiceCenter delivers a process-oriented framework for operations. The synthesis of these domains is a dynamic environment where events and state changes drive procedure and process, and vice versa. This synthesis offers dramatic benefits for the customers of ServiceCenter and OVO.

For more information about integration, refer to Chapter 3, Product Architecture.

Bi-directional Integration

The default mode of SCAuto for OVO operation is the bi-directional integration of ServiceCenter and OVO. In this mode, OVO events trigger ServiceCenter processes. By default, selected events automatically open incident tickets. Subsequent actions or event messages at OVO may update or close the tickets, or a similar exchange may occur from ServiceCenter into OVO.

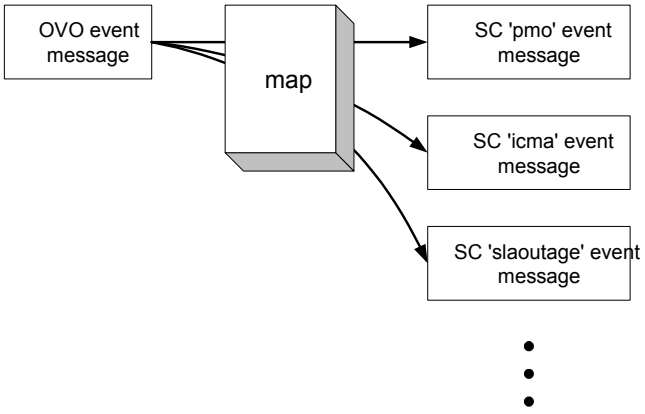
Because most ServiceCenter applications can accept and generate event messages, this same operational flow can be applied to these ServiceCenter applications as well. This default mode of operation can be configured to meet specific end user requirements. The section Planning Your ServiceCenter and OVO Integration on Planning your ServiceCenter and OVO integration on page 17 describes the modes of operation that can be derived from the default.

All event messages that are exchanged between ServiceCenter and OVO must be converted from their native format into a format compatible with the destination. Event messages sent to ServiceCenter must be formatted in the event message structure defined by the SCAuto SDK. SCAuto for OVO performs most of this formatting, but it relies upon ASCII text map files to specify how to convert specific OVO event message data fields into ServiceCenter event message data fields. A similar process is used for converting outbound event messages from ServiceCenter to OVO.

The adapter may create one-to-many relationships of OVO events to incoming SC event messages. This functionality is facilitated by the input maps that use the TCL scripting language. With this feature, for example, an SNMP Node Down event message can do any of the following:

- Open an incident ticket.
- Update a Configuration Management record.
- Start an SLA outage metric against a logical item, such as a database that runs on the node and is reported as down. (This is only available for installations with Service Level Management.)

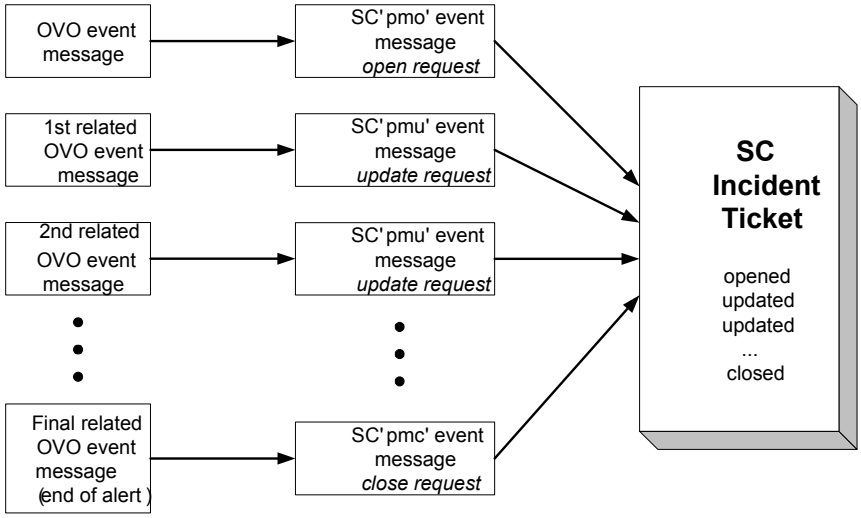
The following figure shows a one OVO Event Message to Many ServiceCenter (SC) Event Messages.



The adapter can also create one-to-many relationships of outgoing ServiceCenter events to OVO event messages or other actions. This output functionality is also supported by the use of TCL scripting language in the output maps. Since the maps are really script programs built to massage data fields, they can be edited easily to cause any desired actions.

The typical event message relationship will be one OVO event message to one ServiceCenter event message. Furthermore, many ServiceCenter event messages will be used to alter the state and the data of just one ServiceCenter incident ticket. For example, a OVO event message will create a ServiceCenter event message that will open an incident ticket. Then another OVO event message creates an update request type of ServiceCenter event message that adds new information to the original incident ticket. Then a final OVO event message will create a close request that ends the active life of the ticket.

The following figure shows One-to-One Event Message Relationships, with Many Event Messages Linked to Just One Incident Ticket.



Planning your ServiceCenter and OVO integration

The following sections describe some of the ways in which SCAuto for OVO can be used. These scenarios help you plan and design your own implementation of OVO integrated with ServiceCenter using SCAuto for OVO. Any mode, combination of modes, or all modes can be used in a single deployment of SCAuto for OVO. Furthermore, an end user can extend the product to implement unique operational processes not described in any of the following mode sections.

For more scenarios, refer to Chapter 6, Scenarios.

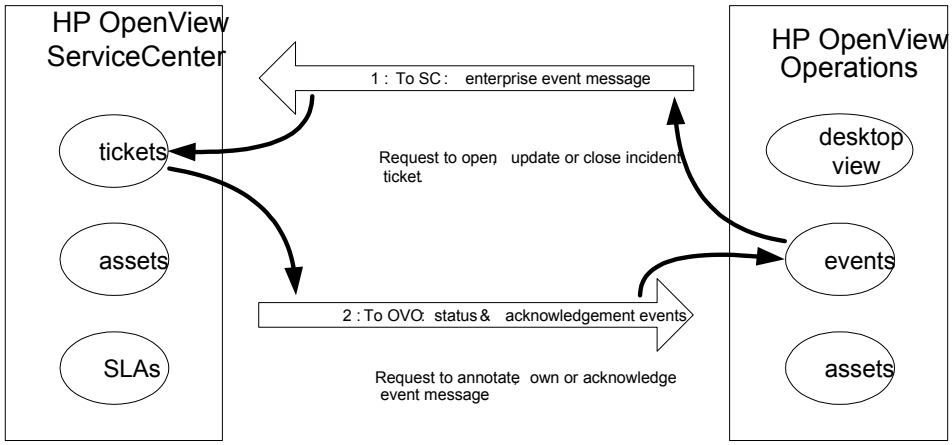
Mode 1: uni-directional automatic notification from OVO

In this mode, OVO events drive ServiceCenter incident tickets. To understand this mode, you should assume that:

- Business logic is applied to the OVO event console to select which events are to be forwarded to ServiceCenter (refer to OVO business logic topics on page 26).
- The only relationship between OVO and ServiceCenter consists of automatic tickets from OVO events. Operationally, this implies that the monitored systems and resources are proactively creating tickets when conditions are met. However, it also means that all actions on resolving the issue, fault or incident are coordinated and administered from ServiceCenter.
- The OVO event console becomes a "lights out" processing engine that feeds the service desk with intelligent real-time data, and then both expects and receives no further information or integration of service processes.

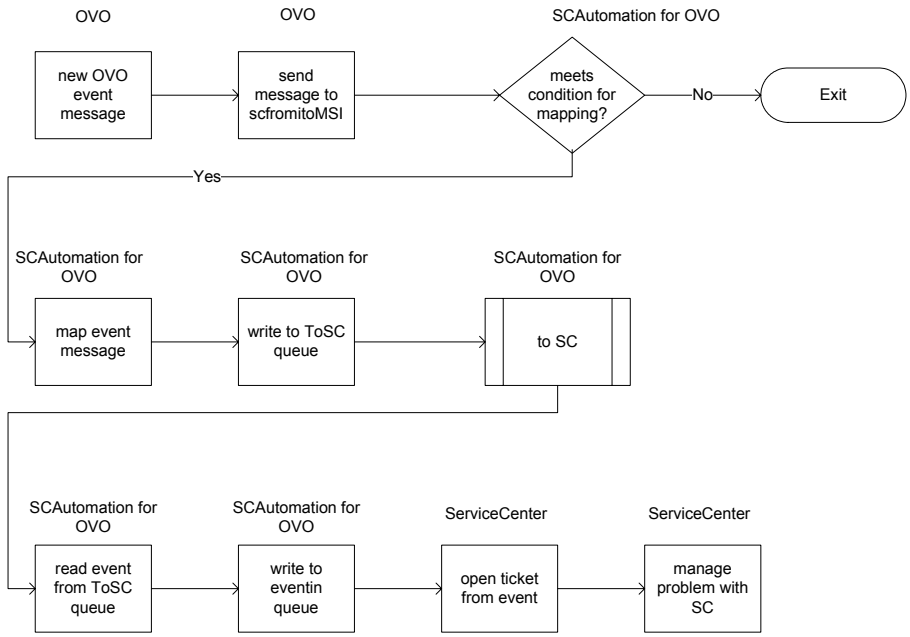
Process and Data Flow

This mode follows a process and data flow as shown in the following two flowchart diagrams.



This mode is configured through the registered connection from OVO to ServiceCenter via the MSI API. This mode will invoke the following components of SCAuto for OVO:

- scfromitoMSI and scevmon processes
- ToSC Queue
- Event Maps (`event.ini`, and maps referenced in `event.ini`)



Mode 2: Bi-directional exchange (default mode of operation)

In this mode, OVO events drive ServiceCenter tickets (records), and the ServiceCenter tickets control OVO events. This creates a partnership of managing the events, where each application has a significant contribution to the event and incident management process.

This mode incorporates Mode 1, but it extends it by generating ServiceCenter events that are sent to OVO. From this bi-directional interface, there are multiple scenarios based upon state transitions and the path of service processing. For example, there are three actions on OVO events that are able to be taken (via specific event messages): annotate, acknowledge, or own. There are many more

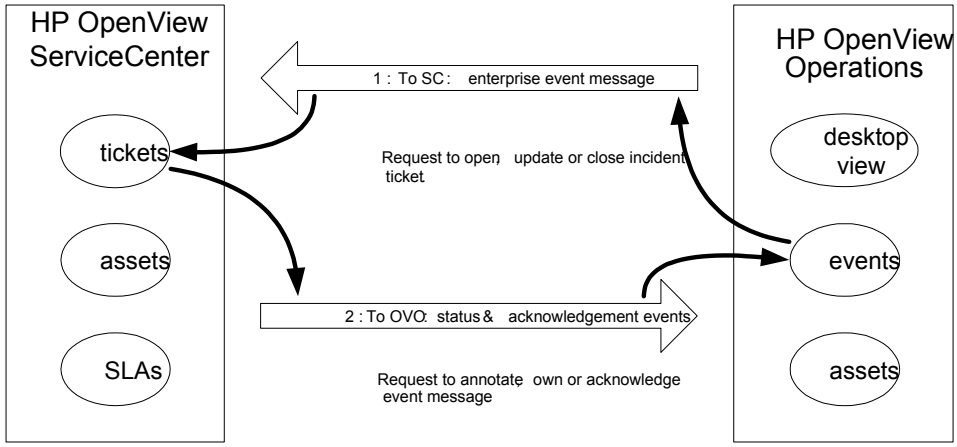
actions that can be taken on tickets, but the various instances always represent a generalized open, update, or close action on the ticket.

This mode of application partnership is the typical operational mode. It offers service desk functions to extend and enhance the real-time event management of OVO. It moves the service desk into more proactive fault management, allowing service desk analysts to respond to emerging issues, rather than reacting to fully developed faults.

This mode allows a richer protocol of integration. In effect, OVO may request the opening of tickets. In response, ServiceCenter acknowledges the open ticket and annotates the OVO event. Subsequent exchanges may inform the applications of changes in state and data values. This mode also supports a complete cycle of interaction in which a close ticket closes (acknowledges) an OVO event, and vice versa. Building this type of interaction involves business logic in both ServiceCenter and OVO.

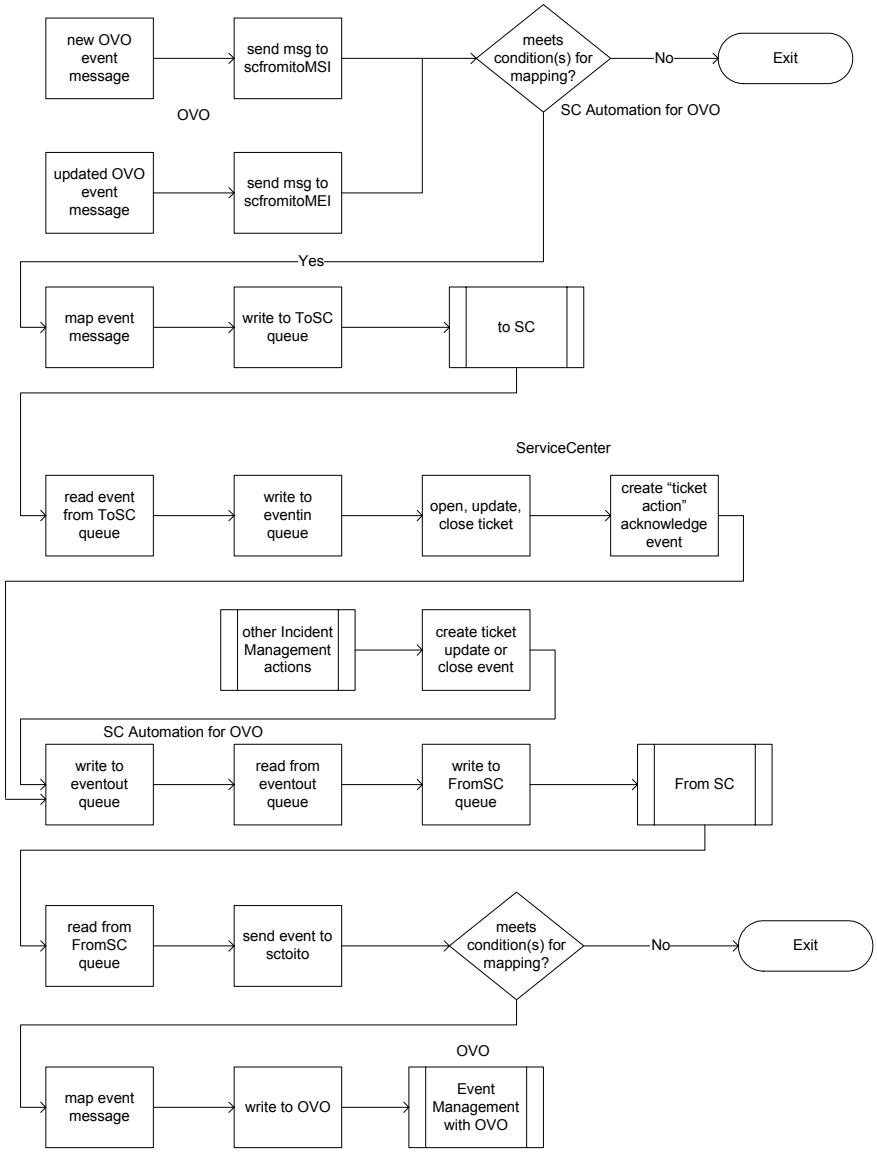
Process and data flow

This mode follows a process and data flow as shown in the following two flowchart diagrams.



This mode is constructed through the registered connection from OVO to ServiceCenter via the MSI API, as well as the MEI API. This mode will invoke the following components of SCAuto for OVO:

- scfromitoMSI, scfromitoMEI, sctoito and scevmon processes
- ToSC and FromSC Queues
- Event Maps (`event.ini`, and maps referenced in `event.ini`)



Mode 3: Automatic notification from OVO via Trouble Ticket Interface (TTI)

This mode is an alternative configuration of Mode 1 (uni-directional) or Mode 2 (bi-directional). It uses the OVO TTI API as opposed to the MSI API. This allows for a more focused configuration of OVO Message Source Templates, freeing the

MSI API to be used for other integration efforts, as well as avoiding the more complicated template formatting required with MSI configuration. With this mode, the MEI usage is identical to the previous modes.

This mode invokes the following components of SCAuto for OVO:

- scfromitoTTI, scfromitoMEI, sctoito and scevmon processes
- ToSC and FromSC Queues
- Event Maps (`event.ini`, and maps referenced in `event.ini`)

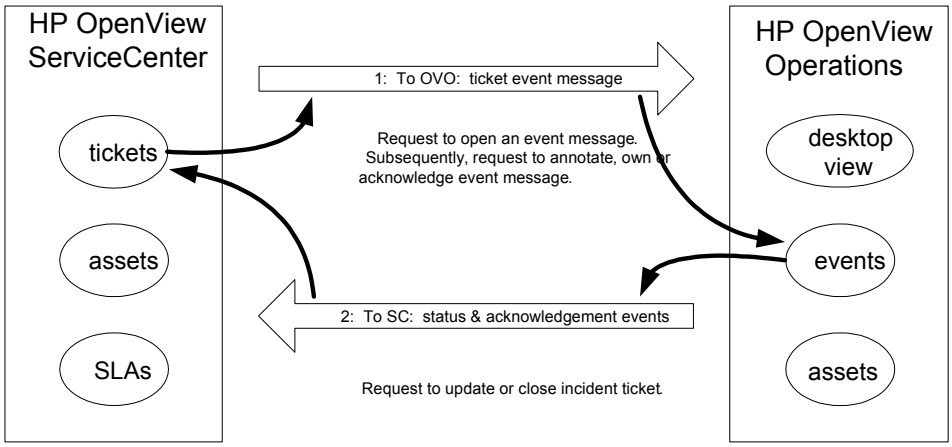
This mode uses the identical flowcharts shown earlier for Modes 1 and 2. The only difference is that references to scfromitoMSI are replaced with scfromitoTTI.

Mode 4: New ServiceCenter tickets generate OVO events

This mode treats ServiceCenter as an event source or event generator. Through the application of ServiceCenter business logic, new incident tickets activate logic, which generates event messages that are sent to OVO. OVO Message Source Templates can be configured to treat these events like any other systems or network management events handled at the OVO console.

This mode begins like Mode 1, but reverses the direction of the first event message. Subsequent interaction looks exactly like Mode 2. In operation, Mode 4 would allow an OVO-centric model of service desk and enterprise management interaction. If desired, this would allow OVO to track certain

ServiceCenter actions, as if they were SNMP traps, systems administration events, or any other typical OVO managed messages.



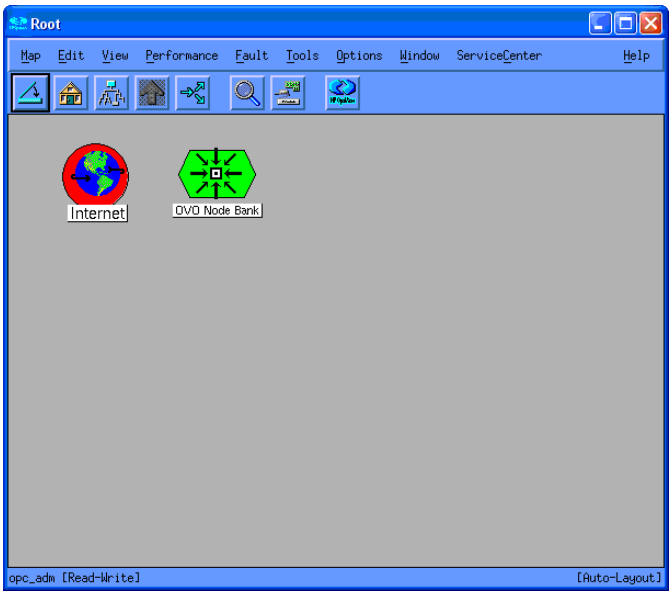
Mode 5: Application monitoring — OVO monitors ServiceCenter

This mode of operation acknowledges that a primary use of OVO is to monitor and manage computer applications. Upon installation of SCAuto for OVO, additional monitoring configuration functions are installed into the OVO Application Bank. These functions leverage the Application Response Measurement (ARM) specifications to offer consistent, standardized treatment of ServiceCenter. This mode also contains Message Source Templates, which allow OVO to be configured to generate events about the health of SCAuto for OVO and about the health of ServiceCenter itself. Further configuration of the Message Source Templates allows these events to become automatically generated incident tickets within ServiceCenter.

Mode 6: Combined user interface — launch ServiceCenter from OVO Windows

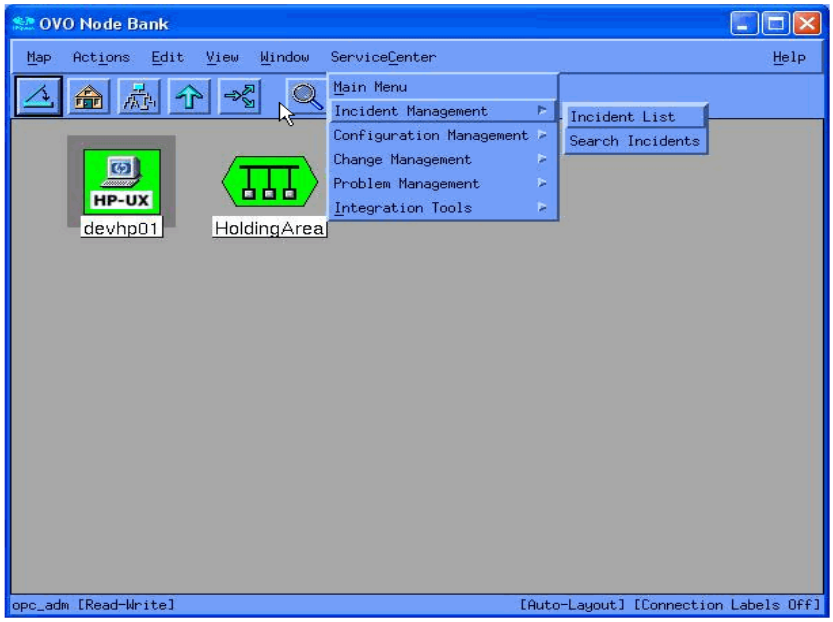
To facilitate greater ease of use, SCAuto for OVO features several options for accessing ServiceCenter (6.1 or later) through a web browser directly from OVO. From the OVO root window, an icon launches a web browser using the url for

the ServiceCenter web tier to open directly to the main menu. The function of this icon is also available from a pull-down menu on the root window.



In addition to the root window, the OVO Node Bank window has a similar menu and icon to launch a connection to ServiceCenter to display specific ServiceCenter windows. In the node bank, it is possible to select a node and

connect to the ServiceCenter web tier to view ServiceCenter data on the selected node. For example, this menu item lists incidents on the node.



OVO business logic topics

OVO automates repetitive tasks; it also enhances and supports operational processes through the tools of OVO. These benefits can be applied broadly because OVO has the ability to adapt to specific needs. This adaptability comes from OVO' flexible approach to operations support. It results in an efficient way of collecting the logic of operations and the logic of running a business and using this business logic in repeatable, sustainable ways.

OVO captures business logic within its operational tools. For example, if an IT department must scrutinize user login attempts for security reasons, OVO can automate the process of such monitoring, and it can also automate the response to unusual conditions. The business logic is stored in Message Source Templates. These templates begin with expected enterprise events, and they configure the OVO processing engines to consistently respond to the events.

Message Source Templates have a wide variety of configuration parameters. Some configuration parameters condition and massage the data of the event messages. Other parameters specify actions and tasks to perform in response to received event messages. These are generic examples of programming business logic into an OVO environment. SCAuto for OVO relies on specific configuration of the templates to invoke application programming interfaces.

In a basic sense, OVO assumes that all events from all sources are potential candidates for integration with a help desk. OVO creates an API called the Trouble Ticket Interface (TTI) API for this purpose. Message Source Templates offer an option button to select this integration function, which automatically sends event messages to all applications that are registered and listening to the TTI API.

An advanced configuration of Message Source Templates is also available. Under these advanced options, you can specify that event messages from specific sources extend themselves to the Message Stream Interface (MSI) API. You can also select whether this is activated at the server or at the agent. When this option is selected, all event messages from the message source are copied to all applications that are registered and listening to the MSI API.

SCAuto for OVO leverages the above API configuration options to invoke its programs. This amounts to a business logic decision of when events of this type occur, invoke the service desk. This logic, in conjunction with the generic OVO configuration options (which capture filtering, data massage, and event message routing business logic), offers the OVO resident component capability of synthesizing the service desk with the enterprise systems management console.

ServiceCenter business logic topics

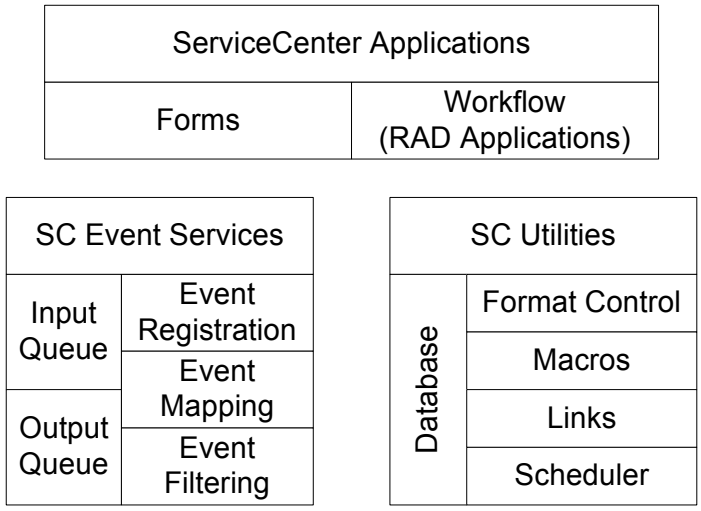
ServiceCenter facilitates and automates the process of supporting business operations. Much like OVO, these benefits can be applied broadly because of ServiceCenter's flexibility and adaptability to address specific needs. Like OVO, this reflects ServiceCenter's ability to capture business logic within its applications.

ServiceCenter provides a significant amount of generic business logic within the service desk applications. The Incident Management application, for example,

follows a well-defined help desk model of incident identification, tracking, and resolution. It can be used as is, or it can be customized to address specific organizational requirements. Custom business logic is most often applied to the escalation and notification aspects of the process model, although the process workflow is also frequently altered.

Business logic in ServiceCenter can be associated with several levels of ServiceCenter:

- At the bottom level are the ServiceCenter databases, where trigger mechanisms can dictate custom actions.
- Above the database level is a utilities level, which is composed of Schedules, Format Control, Macros, and Links. These tools offer the preferred location for applying custom business logic. All of the tools were designed to support this function. The ServiceCenter Event Services module essentially connects with ServiceCenter at the utilities level. This module provides equivalent functions related to processing incoming and outgoing event messages.
- The final level where business logic can be defined is at the ServiceCenter applications. This level either lends itself to easy customization of the user presentation of application data (via Forms Designer), or it involves more significant alterations that require a programming solution (via ServiceCenter's fourth generation language, RAD).



SCAuto for OVO utilizes the ServiceCenter Event Services module for its business logic integration with ServiceCenter. It leverages default configurations of event messages to open, update, or close an incident ticket. These event message requests are defined as event types of pmo, pmu, or pmc, respectively. The processing logic of these requests can be altered if needed, but the default configuration is satisfactory for integration with OVO.

Comparable to OVO's business logic decision of when events of this type occur, invoke the service desk, the SC Event Services response is when the service desk is invoked, receive and process the data, and then launch the appropriate service desk function. This logic is often combined with Format Control utility logic, which injects programmed queries, tests, and automated actions into the application layer (and therefore into process model of the service desk).

This configuration of utility level business logic offers the ServiceCenter resident component of synthesizing the service desk with the enterprise systems management console. When combined with the OVO portion, the result is an extremely powerful integration that facilitates more effective and efficient IT operations and the operations of the business as a whole.

SCAuto for OVO business logic topics

The previous two sections discussed capturing your business requirements in the configuration of OVO or ServiceCenter. As you define how your consolidated service desk functions, you will undoubtedly use the topics of the last two sections. However, when deploying such an enterprise solution, you may also need to put business logic into the middleware represented by SCAuto for OVO.

Other sections of this document discuss the features of this product that allow it to be customized and configured. Any of these features are sufficient for beginning to program business logic into the product. Realistically, any custom logic will be coded into the TCL scripts used for the data maps. Business logic captured in the map scripts will be able to affect many aspects of the overall integration: unique data items, selection of particular event messages, or creating many events from one input event. Regardless, refer to the earlier sections for more details on SCAuto for OVO's abilities of capturing business logic.

In today's business world, companies rely upon their IT resources. HP OpenView and ServiceCenter deliver the IT support that companies count on. The SCAuto for OVO product integrates these major applications, and creates a synthesis of real-time management and process oriented service delivery. Because of the natural dynamic nature of IT, there are constantly new challenges to be addressed in IT management. This new product helps address those challenges directly, and enables enterprise solutions built from the mature concepts of the consolidated service desk.

2 Installation

CHAPTER

This chapter gives the requirements for installing ServiceCenter Automation (SCAuto) for OpenView Operations (OVO) and provides step by step installation instructions.

System requirements

To use SCAuto for OVO, your installation must be on a UNIX platform and must include the following:

- ServiceCenter version 5.1 or later
- Client launch only supported with ServiceCenter 6.1 or later
- OVO 8.20 on HP-UX 11.23 Itanium
- OVO 8.10 on HP-UX 11.0 or 11.11 on HP 9000
- OVO 8.10 on Solaris 8 and 9
- 35 MB temporary unpack space
- 35 MB installed footprint

The ServiceCenter server can run on Windows, UNIX, or Linux. SCAuto for OVO presently runs exclusively on the UNIX platform.

OVO should be implemented and operational to the extent that meaningful events arrive at the Event Message browser, and at least one designated individual is familiar with the OVO product as well as defining OVO Event Message sources and OVO Message Actions.

SCAuto for OVO supports ServiceCenter 5.1 or later. By default, SCAuto for OVO is designed to work with the default ServiceCenter configurations. If ServiceCenter is tailored, you may also have to tailor SCAuto for OVO.

Customers with heavily customized or older systems should give consideration to use of Professional Services to implement SCAuto products.

Required kernel parameters

Your HP-UX or Solaris machine must have the following kernel parameter settings:

- The minimum value of any semaphore must be greater than or equal to 1 (SEMVMX).
- The maximum number of semaphore sets systemwide must be increased by 3 (SEMMNI).
- The maximum number of semaphores systemwide must be increased by 3 (SEMMNS).
- The minimum number of semaphores per semaphore set must be greater than or equal to 1 (SEMMSL).
- The maximum number of undo structures systemwide must be increased by 3 (SEMMNU).
- The minimum number of undo entries per undo structure must be greater than or equal to 1 (SEMUME).
- The minimum number of operations per semop must be greater than or equal to 1 (SEMOPM).

Installation requirements

To install SCAuto for OVO, you must have administrator-level access both to the OVO server and the ServiceCenter server. Installation takes only a few minutes if you have determined the correct host name and port number values beforehand.

Before you begin installation, you need the following information about your OVO and ServiceCenter installations:

- Administrator authority at the OVO server
- Administrator-level access to the ServiceCenter server
- An authorization code for ServiceCenter
- The host name or IP address of the ServiceCenter server and the TCP port number for use by ServiceCenter and the SCAutomation Base Server.
- Web tier host name and port number for the GUI.

Installing SCAuto for OVO

There are two major parts of the installation process:

- The Event Integration binaries and support files. This includes HP Local Registration Files (LRFs) that integrate with the OVsPMD.
- The GUI Integration components, which can be broken down into:
  - ServiceCenter Message Group object and Application Group Objects

During installation, these parts will be configured with the customer's options and settings.

The deliverable image is available as a download or on a CD-ROM (ISO9660 format). The installation begins by executing the `install.sh` script. Installation includes these steps:

1. Mount the CD.
   - On HP-UX use the command mount -o cdcase or pfs_mount.
   - On Solaris the CD is mounted automatically.
2. Go to the appropriate directory. The directory name is:
   - hp_11_ia64 for HP-UX 11.23 Itanium

- hp_11 for HP-UX 11.
- solaris for Solaris 8 or 9.

3 Execute install.sh.

Install procedure

Important: The installation must be done as the root user.

The installation process consists of two scripts: `install.sh` and `config/install2.sh` in `scito.tar`. Executing `install.sh` unpacks `scito.tar` into a directory and then executes `config/install2.sh`. Interactive prompts enable you to stop the installation at any time. During this process, files are placed into a directory for potential manual installation. Please note that stopping the installation is not recommended.

Provide the following information when prompted:

- Root directory of OVO (for example, /opt/OV).
- Target directory for product installation. (for example, /opt/OV/scauto).
- The ServiceCenter server host name and port number.
- The ServiceCenter web tier host name and port number.

Note: For ServiceCenter 6.1 or later, the ServiceCenter web tier information is required only if you want ServiceCenter Client GUI integration.

The installation script then takes you through the following steps:

1 Prompt for root directory of OVO.

2 Prompt for the target directory for installation, and it copies Event Maps and binaries over.

3 Create and configure `/etc/scauto/SCITO.init` file with the home directory of product.

4 Configure product ini file `scito.ini` with ServiceCenter Server information.

5 Configure and install HP Local Registration Files with product start/stop information.

6 Configure and add Menu/Toolbar GUI customizations to user opc_adm and opc_op. (Optional)

7 Create ServiceCenter Message Group Object in Message Group Bank and Application Object in Application Bank. (Optional)

If you encounter a problem that you are unable to resolve, please contact Customer Support.

CHAPTER 3

Basic Operations

Starting and stopping SCAuto for OVO processes

The ServiceCenter Automation (SCAuto) for OpenView Operations (OVO) applications are installed into OVO from Local Registration Files (LRFs). This means that you can start and stop the processes using the HP ovstart/ovstop facilities. The SCAuto for OVO processes consists of the event monitor (scevmon), ServiceCenter to OVO adapter (sctoito), and the OVO to ServiceCenter adapters (scfromitoMEI, scfromitoMSI, and scfromitoTTI).

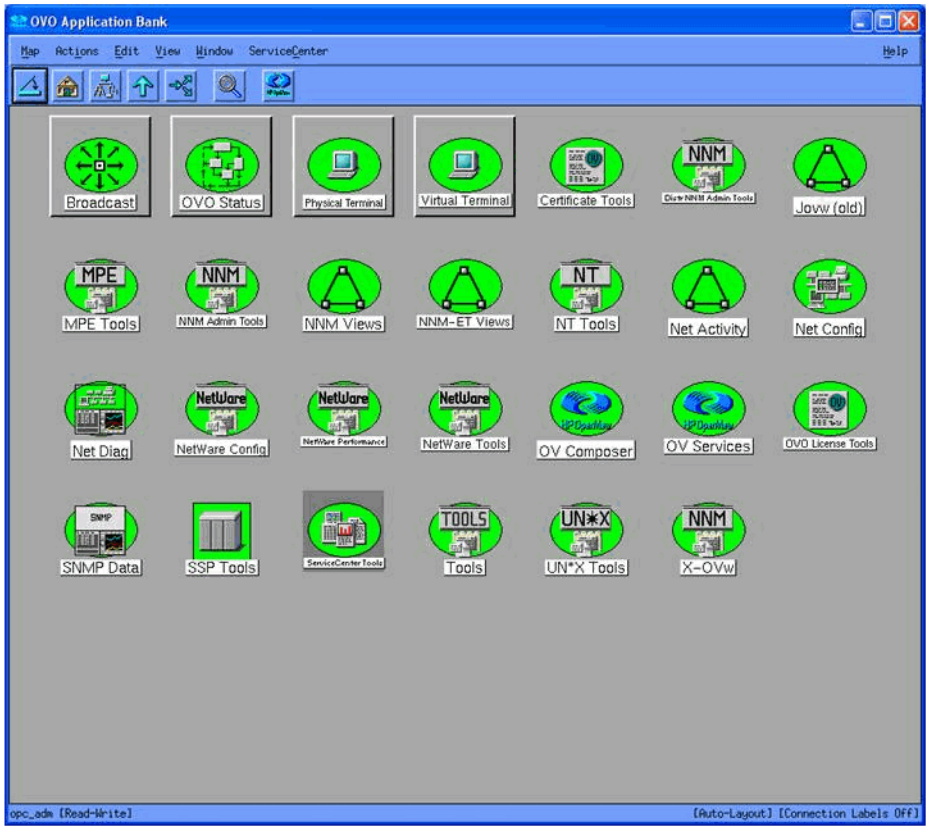
Starting SCAuto for OVO

There are three ways to start the SCAuto for OVO adapter processes:

- If you chose the GUI integration during installation of SCAuto for OVO, you can start all processes by selecting the following options, in order, from the menu on the OVO Root Window or the OVO Node Bank Window:
  - ServiceCenter
  - Integration Tools
  - Start All Adapters
- You can use the OVO command line `ovstart` to start individual processes. For example, use `ovstart scevmon` to start the scevmon process and

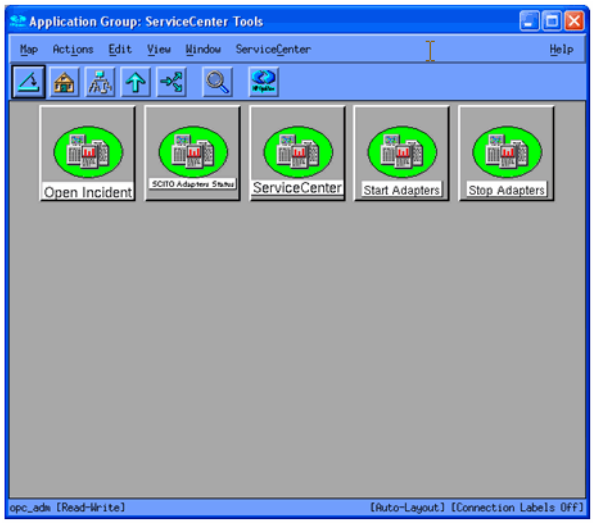
`ovstart scevmon scfromitoMSI` to start the scevmon and scfromitoMSI processes.

- SCAuto for OVO can also be started from its icon (called ServiceCenter Tools) in the OVO Application Bank window.



You can start or stop SCAuto for OVO by selecting the ServiceCenter Tools icon in the OVO Application Bank window.

The ServiceCenter Tools screen opens.



This window contains a set of icons that controls SCAuto for OVO and offers some control over ServiceCenter.

- Open Incident - allows user to open an incident ticket on selected messages in the Message Browser.
- SCITO Adapters Status - shows the operational status of the adapter processes.
- ServiceCenter - opens a connection to the ServiceCenter web tier (for ServiceCenter 6.1 or later only) if GUI integration chosen.
- Start Adapters - starts the SCAuto for OVO adapter processes.
- Stop Adapters - stops the SCAuto for OVO adapter processes.

Stopping SCAuto for OVO

There are three ways to stop the SCAuto for OVO adapter processes:

- If you chose the GUI integration during installation of SCAuto for OVO, you can stop all processes by selecting the options listed below, in order, from the menu on the OVO Root Window or the OVO Node Bank Window.
- You can use the OVO command line `ovstop` to stop individual processes. For example, use `ovstop scevmon` to start the `scevmon` process and `ovstop scevmon scfromitoMSI` to stop the `scevmon` and `scfromitoMSI` processes.
- SCAuto for OVO can also be stopped from its icon (in the ServiceCenter Tools) in the application group Application Bank window.

Basic maintenance

In the product installation directory, the SCAuto for OVO adapter contains a log file called `scito.log`, as well as a parameter configuration file called `scito.ini`, where all informational and error messages from the product are stored.

The product requires very little maintenance once installed and running. By default the log file is archived to `scito.log.archive` once it reaches 5M bytes. In addition, the event queues purge processed events to remove them once they reach 5M bytes each, retaining the unprocessed events.

scito.ini parameters

This is a subset of the commonly used parameters that can be set in the `scito.ini` file:

- sessid - By default, this is set to OPCENTER. This is the license string for the SCAuto for OVO product. Do not modify.
- scauto - This is the `hostname.port` of the SCAuto server for SCAuto for OVO. This parameter is configured by the installation script.
- event_map_dir - This is the root directory name (starting from the installation directory) of the event mapping scripts and files. By default, it is configured as “EventMap”.
- log - This is the file name for the log file. By default, it is configured as `scito.log`.

- debugMEI :1, 2 - Set =1 turns on tracing for the scfromitoMEI process. Set =2 turns on more verbose tracing for the scfromitoMEI process
- debug MSI: 1, 2 - Set=1 turns on tracing for the scfromitoMSI process. Set =2 turns on more verbose tracing for the scfromitoMSI process.
- debugTTI: 1, 2 - Set=1 turns on tracing for the scfromitoTTI process. Set=2 turns on more verbose tracing for the scfromitoTTI process.
- debugTOITO: 1, 2 - Set=1 turns on tracing for the sctoito process. Set=2 turns on more verbose tracing for the sctoito process.
- debugscautoevents: - This is the SCAuto event debugging flag to output more debugging messages of the event type. Enable the parameter by setting it to 1.
- eventlogmaxlen - This parameter defines the maximum size, in bytes, that the event queue files can reach before being purged of processed events. The default is 5M. The minimum size is 1M.
- logmaxlen - This parameter defines the maximum size in bytes that the log file can reach before it will be wrapped down to `logpreservelen`. The default is 5M. It is not recommended to set this value to less than 1M.
- logpreservelen - This parameter defines the size of the log file to wrap down to in the event it reaches logmaxlen. The default is 0 which causes the `scito.log` file to be archived to `scito.log.archive` when the `logmaxlen` is reached and empties the current `scito.log` file. The maximum is 128K.
- usersepchar - Allows user to define a separator character other than "^" (^ is the default value). To specify a different character, enter a decimal value of an ASCII character between 1 and 255.
- scevents - Specifies an optional list of event types which are to be retrieved from ServiceCenter's EVENTOUT queue. The default is to retrieve all types, however, this can be inefficient, because it can result in bringing over certain types of events, such as outbound page messages or email messages, which have no meaning to Operations. To restrict the types of events that are retrieved, code them in a list separated by commas and enclosed in parentheses. For example, `scevents:(pmo,pmu,pmc)`. To disable all retrieval of outbound events from ServiceCenter, use the "`noeventsfromsc:`" option.
- noeventsfromsc - Specifies whether or not scevmon should retrieve events from ServiceCenter's EVENTOUT queue. The default is 0 (which means events will be retrieved). To disable retrieval of all outbound events from ServiceCenter, code "`noeventsfromsc:1`".

- scevusers - This parameter restricts the events which are to be retrieved from ServiceCenter's EVENTOUT queue based on the value in evuser field. The default is to retrieve all events regardless of the value of the evuser field, however, this can be inefficient, because it can result in bringing over certain events which are not intended for OVO. To use this parameter, code the values in a list separated by commas and enclosed in parentheses.

  For example, `scevusers:falcon` only get events that have falcon as the evuser. `scevusers:(falcon, SCITO)` only get events that have falcon or SCITO as the evuser.

- scevmon_sleep_interval - Specifies the amount of time in seconds that scevmon should sleep. The default is 5 seconds. Fore example, `scevmon_sleep_interval:1` tells scevmon to sleep for one second.

Basic configuration

The SCAuto for OVO product is configured during installation. After the product is installed, it will work out of the box with the current OVO configuration. To further tailor the system to your business needs, refer to Chapter 5, "Configuration."

Troubleshooting

If incident tickets in ServiceCenter are not being created by OVO event messages, follow these steps to troubleshoot the problem:

1. Verify that the event monitor process scevmon is running. If this process is not running, execute `ovstart scevmon` to start it.

2. Verify that the SCAuto for OVO event monitor is communicating with the SCAuto server. If the `scito.log` file contains the message

   - ... `scevmon` : unable to connect ..., do the following:
   - Check the `scito.ini` file for the parameter scauto:. Verify that it exists and points to the `<hostname>.<port number>` of the SCAuto server you are trying to connect to.

- Log in to the ServiceCenter server host and verify that the SCAuto server is running.
- Check the network by pinging the ServiceCenter Server host to see if it is reachable.

3 Verify that the `scfromitoMSI` and `scfromitoMEI` processes are currently running. If not, execute `scfromitoMSI` and `scfromitoMEI` to start them.

4 Verify that OVO event message sources are correctly set up to communicate with SCAuto for OVO. (Refer to Chapter 5, "Configuration," for more details on configuring OVO message sources.)

- If using TTI interface, verify that the event source has Trouble Ticketing enabled. Also verify that the Trouble Ticketing interface is correctly configured to use the `TTI.sh` script in the installed scauto directory.
- If you are using the MSI interface, verify that the event source has Server MSI or Agent MSI message copy/divert enabled.
- Verify that the `event.ini` configuration file in the `EventMap/ToSC` directory is properly configured.

5 Finally, check the `scito.log` file for any unusual error messages.

If ServiceCenter incident ticket modifications are not reaching the OVO message browser, follow these steps to troubleshoot the problem:

1 Verify that a record is generated in the eventout table in ServiceCenter. Configure format control to output equivalent records accordingly.

2 Check to see if the event monitor process `scevmon` is running at the OVO host. Execute `ovstart scevmon` to start it.

3 Check to see if the SCAuto for OVO event monitor is communicating with the server. If the `scito.log` file contains the message ... scevmon : unable to connect ..., do the following:

- Check the `scito.ini` file for the parameter scauto:. Verify that it exists and points to the `<hostname>.<port number>` of the SCAuto server you are trying to connect to.

- Log in to the ServiceCenter server host and verify that the SCAuto server is running.
- Check the network by pinging the ServiceCenter Server host to see if it is reachable.

4 Verify that the `sctoito` process is running using `ovstatus sctoito` on the OVO host. If it is not running, execute `ovstart sctoito` to start it.

5 Check the configuration file `EventMap/FromSC/event.ini`, and verify that the user name and password are correct.

6 Finally, check the `scito.log` file for any unusual error messages.

4 Product Architecture

CHAPTER

This chapter covers the architecture of HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Operations (OVO), focusing on two primary topics:

- Application Integration
- Event Integration

Application integration

SCAuto for OVO provides an enhanced operator interface to ServiceCenter that can run under OVO. From an OVO window, you can access a number of ServiceCenter windows to gather information related to the current window or selected object.

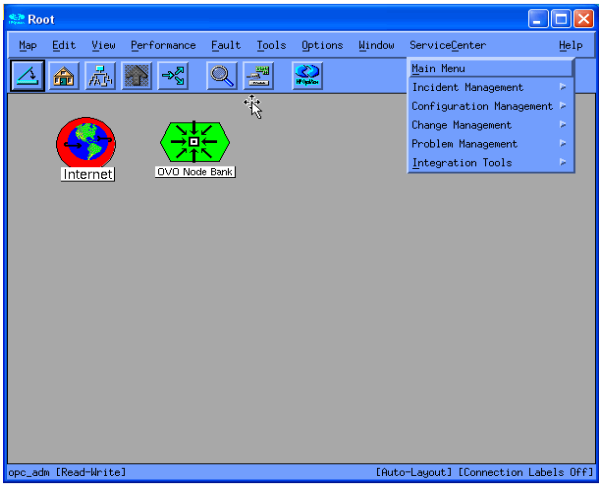
This capability is only available if the ServiceCenter GUI interface is enabled during the SCAuto for OVO installation. If the GUI interface is not enabled, you can start a ServiceCenter client from the UNIX command line instead.

Note: Refer to the appropriate ServiceCenter documentation for more information on using ServiceCenter.

ServiceCenter clients are started through the ServiceCenter menu in a OVO window. Depending upon the menu item selected, the client window will be opened to different ServiceCenter applications and windows.

All menu options are available if an icon for an object is selected in a OVO window. Specific requests requiring an object selection are grayed out if an icon is not selected.

The following windows are a tutorial representation of SCAuto for OVO general operations. Windows and functions may change from release to release, so reference the help files on your specific platform for the latest operational details.



ServiceCenter menu options

The ServiceCenter menu options take you directly to the ServiceCenter applications from within OVO. The following sections provide a brief description of the windows.

Note: While some of the ServiceCenter application options are mentioned in this manual, you should refer to the ServiceCenter documentation for complete instructions on using the ServiceCenter applications.

To use a ServiceCenter application under SCAuto for OVO:

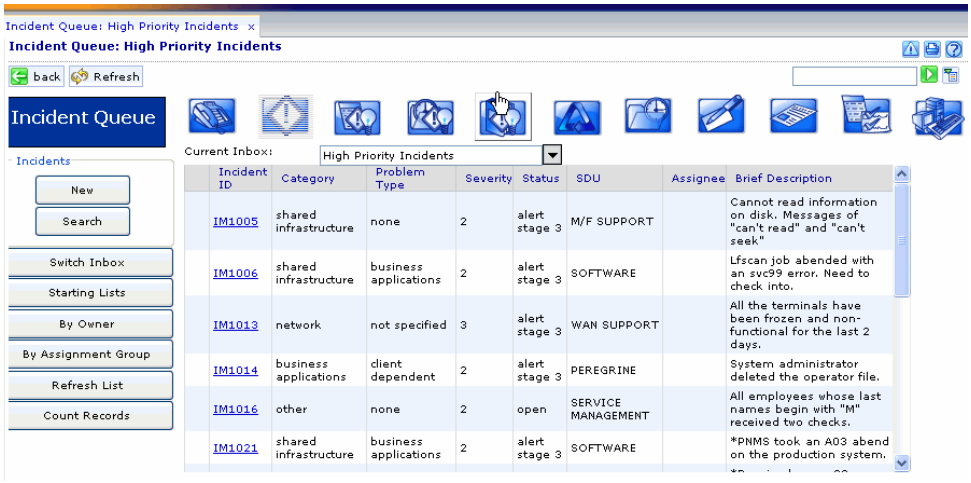
1. Select the ServiceCenter menu in the OVO window and select the appropriate menu option. Some ServiceCenter menu options are not available unless an object is selected in the OVO window.

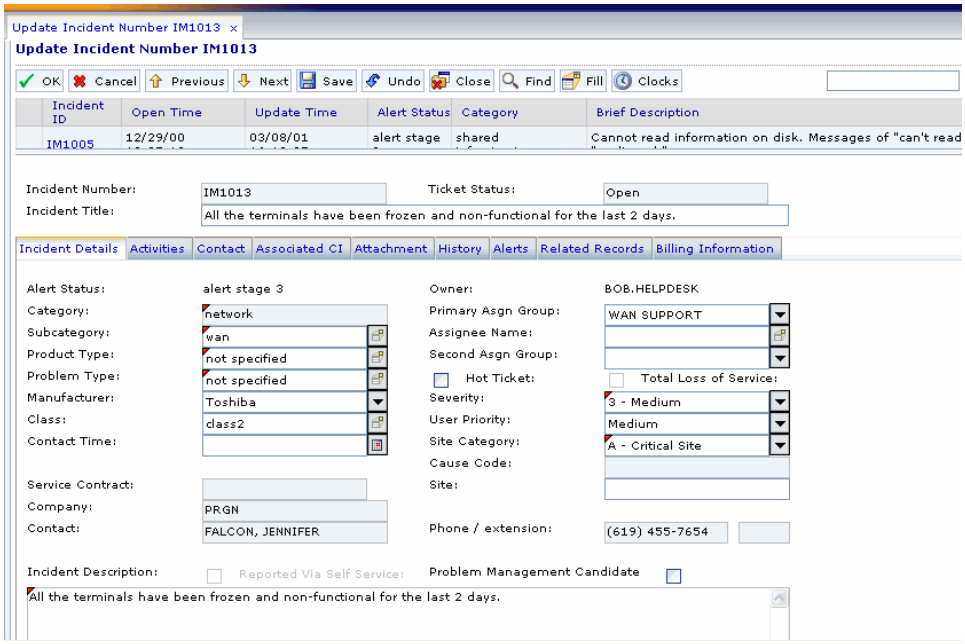
2. Use the mouse or keyboard to navigate through a window.

3 To leave the application, select the Back button or press F3. This takes you to the previous window or to a logout window.

Incident List

The Incident List menu option provides a list of incidents that are currently active in ServiceCenter for the selected object. When this option is selected, an incident list opens. Use the Options menu to display a list of operations to perform on the incidents. Double-click an item the incident ID to display the Incident Detail window.





Event Integration

SCAuto for OVO Event Integration is a collection of tools or components that is highly customizable for integrating event messages from OVO to create, update, or close a ServiceCenter incidentticket.

Event Message integration provides these capabilities:

- Ability to tap into the Message Stream Interface (MSI) for all new eventmessages based on event registration conditions and to create new ServiceCenter problem tickets.
- Ability to tap into the Message Event Interface (MEI) for modifications done to existing event messages based on event registration conditions, and to update or close existing ServiceCenter problem tickets.
- Provide a trouble ticketing interface from which configured OVO events may create ServiceCenter problem tickets.
- Ability to create new OVO event messages from ServiceCenter eventout output events.

- Ability to modify existing OVO event messages based on modification being done to related problem tickets in ServiceCenter using the eventout output events.
- Ability to correlate ServiceCenter problem tickets to one or more OVO event messages.
- Ability to correlate OVO event messages to a ServiceCenter incident ticket.

Integration components

The following table shows the integration components for SCAuto.

File Name	File Type	File Description
scevmon	executable for manipulating queue files to and from ServiceCenter.	This is the core technology of the SCAuto product that facilitates bi-directional queueing of messages to and from ServiceCenter.
sctoito	executable for reading the *from* queue file and forwarding to OVO	This executable functions to take an event forwarded from ServiceCenter eventout and generate/modify OVO event messages using a series of customizable TCL8.0 scripts. 80% of the OVO API library has been converted into TCL callable functions accessible by these scripts. This executable runs in the background waiting for events. The customer can do more than is provided by the OVO APIs, such as remote controlled processes.
scfromitoTTI	executable to interface with OVO Trouble Ticket Interface	Basically, OVO needs to customize interested events to forward to Trouble Ticketing first. Trouble Ticketing Interface will then call a script and pass as arguments static data describing the event. We are only interested in the first argument, which is the message ID. This executable is called once via a shell script indicated in the Trouble Ticketing custom dialog for every intended event.
scfromitoTTI	executable to interface with OVO Trouble Ticket Interface	This executable will register events for the Message Stream Interface and run in the background waiting for event notification. Upon notification, it will use event maps and generate the appropriate SC events to the *to* queue. The Message Stream Interface will notify only new events in OVO.

File Name	File Type	File Description
scfromitoMSI	executable for writing the *to* queue from OVO Message Stream Interface	This executable will register events for the Message Stream Interface and run in the background waiting for event notification. Upon notification, it will use event maps and generate the appropriate SC events to the *to* queue. The Message Stream Interface will notify only new events in OVO.
scfromitoMEI	executable for writing the *to* queue from OVO Message Event Interface	This executable will register events for the Message Event Interface and run in the background waiting for event notification. Upon notification, it will use event maps and generate the appropriate SC events to the *to* queue. The Message Event Interface will only notify about changes done to existing events in OVO.
various map files/TCL scripts	configuration text files	TCL scripts that are customizable for creating/modifying OVO events from ServiceCenter events as well as from OVO to ServiceCenter. Also, there are *ini* config files that need to be set up to facilitate scevmon and the locations of these files.

scevmon

This is the core of the SCAuto SDK V3. It is best described as an event message processor using to (ServiceCenter) and from (ServiceCenter) queue files to cache events generated by the other components. The basis of this design is to facilitate a fault recovery system that enables OVO event messages to be queued up even while ServiceCenter is currently not available, as well as queueing up ServiceCenter events while waiting for OVO to be available. The contents of the queue files include the actual ServiceCenter events in "^" delimited format preceded by a header. These files should not be modified by hand.

The scevmon component typically runs in the background, makes a TCP connection to the SCAuto server, and waits for events to appear in the queue files. Once any of the other integration components writes an event string to the to queue file, scevmon picks it up, marks it as processed and forwards it to ServiceCenter's eventin table. On the other hand, if a problem ticket is modified and an entry gets created in the eventout table in ServiceCenter, scevmon picks it up, updates the sync file to point to the last entry in the eventout table, and writes the event string into the from queue file. At this point, the sctoito

integration component picks it up, marks it as processed, and forwards it through the TCL mapping scripts to OVO.

The config files used by scevmon are:

- `/etc/scauto/SCITO.init` - a file that contains a variable pointing to the installed directory of the product.
- `<installation directory>/scito.ini`

The scevmon component reads these files:

- `$SCITOHOME/scevents.to.<host>.<port>` - contains ServiceCenter eventin events created by scfromito[MSI|MEI|TTI] programs. This file is updated to show processed events.
- `$SCITOHOME/syncfile.<host>.<port>` - synfile for the from queue with ServiceCenter eventout table.

The scevmon component writes these files:

- `$SCITOHOME/scito.log` - log file for informational and error messages.
- `$SCITOHOME/scevents.from.<host>.<port>` - contains ServiceCenter eventout events to be passed on to OVO. This file is read in by the sctoito program.
- `$SCITOHOME/syncfile.<host>.<port>` - synfile for the from queue with ServiceCenter eventout table.

sctoito

This is the background process that waits on the from queue, and reads in any new incoming events from ServiceCenter eventout table. It processes the event by passing it to a configurable TCL script and marks the event as processed.

This program is customized by <installation directory>/EventMap/FromSC/event.ini and will execute the TCL scripts pointed to by the event type names in the [EVENT] section. The [CONFIG] section is a place to specify the OVO username and passwords to connect with. Please refer to Chapter 5, “Configuration,” for more information about customizing the component.

The following are the components in SCAuto for OVO that read OVO messages using the Message Stream Interface or through direct OVO database APIs. They are customized by event.ini for parameters to register with the Message Stream Interface, as well as TCL scripts to execute for mapping the events to ServiceCenter event strings.

scfromitoTTI

This is the Trouble Ticket Interface adapter that can be executed using a shell script. It is the program defined for the OVO Trouble Ticketing Interface. The event source templates are customized to forward events to this application when it is running.

scfromitoMSI

This is the background process that writes to the from queue when registered OVO events appear in the Message Stream Interface.

scfromitoMEI

This is the background process that writes to the from queue when registered OVO events appear in the Message Event Interface.

These programs are customized by `$SCITOHOM/EventMap/ToSC/event.ini`.

5 Configuration

CHAPTER

This chapter describes how to configure HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Operations (OVO). It covers three main topics:

- SCAuto for OVO Business Logic Configuration
- OVO Business Logic Configuration
- ServiceCenter Business Logic Configuration

SC Auto for OVO business logic configuration

This section discusses TCL language event mapping from OVO to ServiceCenter. This scripted approach to configuring the SCAuto for OVO product results in a flexible method of configuring or customizing it to meet specific business goals and requirements.

The mapping mechanism to map OVO message events into ServiceCenter incident tickets is made up of an active facility, which is a TCL script, acting on variables populated with OVO message elements to create a ServiceCenter TCL event object, using a passive facility, a static map file positionally defining each kind of interested ServiceCenter event. The send command of this ServiceCenter TCL object is then invoked to queue the event to be sent to ServiceCenter. In addition, the messages being handed to these facilities are themselves configurable to be filtered using OPCREG parameters at the interface program level using an `event.ini` configuration file.

Configuration overview

By default, the mapping files reside in the EventMap directory in the product installation directory. The hierarchy and explanation of these files follows:

- EventMap
- EventMap/FromSC/ -- files to configure events from ServiceCenter to OVO.
- EventMap/ToSC/ -- files to configure events from OVO to ServiceCenter

The following files are located in the `EventMap/ToSC/` directory:

`event.ini`	Configuration file to specify MSI and MEI registration parameters to filter messages as well as which TCL map script to invoke by which application subcomponent. A detailed description will appear under the heading OVO Message Filtering The event.ini file.
`eventmapMSI.tcl`	The event map script for the MSI interface (scfromitoMSI).
`eventmapMEI.tcl`	The event map script for the MEI interface (scfromitoMEI).
`eventmapTTI.tcl`	The event map script for the TTI interface (scfromitoTTI).
`pmo.map, pmu.map, pmc.map`	Static map file templates used in the scripts to positionally define ServiceCenter event elements in the ServiceCenter TCL event object.
`util.tcl`	A utility script that contains commands to convert OVO severity values to ServiceCenter priority codes.

Basically, the `event.ini` file configures what the MSI and MEI interfaces will get from OVO and which TCL scripts to invoke when there is an OVO message. Then, the TCL scripts will use the static map files as a template to create ServiceCenter TCL event objects, populate its members, and finally send it off to the queue file to be sent to ServiceCenter eventin table.

OVO variables

The available variables are the same as the attribute names of the OPCDTYPE_MESSAGE. This definition can be found in Chapter 4 of the *HP*

OpenView Developer's Toolkit Application Integration Guide. These variables are made available as TCL_GLOBAL and will be accessible via the $VARIABLE syntax globally from within the called TCL script.

Variable Name	Short Definition
OPCDATA_MSGID	The unique OVO message ID.
OPCDATA_NODENAME	Name of the node producing the message. The message is only handled by the OVO manager if this system is part of the OVO Node Bank.
OPCDATA_CREATION_TIME	(Local) Time the message was created in seconds.
OPCDATA_RECEIVE_TIME	Time the message was received by the management server in seconds.
OPCDATA_MSGTYPE	Message type. This attribute is used to group messages into subgroups, e.g., to denote the occurrence of a specific incident. This information may be used by event correlation engines.
OPCDATA_GROUP	Message group.
OPCDATA_OBJECT	Object name to use for the OVO message.
OPCDATA_APPLICATION	Application which produced the message.
OPCDATA_SEVERITY	Severity of the message. Possible values are: 0 - OPC_SEV_UNCHANGED 4 - OPC_SEV_UNKNOWN 8 - OPC_SEV_NORMAL 16 - OPC_SEV_WARNING 32 - OPC_SEV_CRITICAL 64 - OPC_SEV_MINOR 128 - OPC_SEV_MAJOR
OPCDATA_AACTION_NODE	Defines the node on which the automatic action should run.
OPCDATA_AACTION_CALL	Command to use as automatic action for the OVO message.

Variable Name	Short Definition
OPCDATA_AACTION_ANNOTATE	Defines whether OVO creates start and end annotations for the automatic action. 0 - do not create annotations 1 - create annotations
OPCDATA_AACTION_ACK	Auto Acknowledge after successful execution of the Automatic Action. 0 - do not auto-acknowledge 1 - auto-acknowledge
OPCDATA_OPACTION_NODE	Defines the node on which the operator initiated action should run.
OPCDATA_OPACTION_CALL	Command to use as operator-initiated action for the OVO message.
OPCDATA_OPACTION_ANNOTATE	Define whether OVO creates start and end annotations for the operator initiated action. 0 - do not create annotations 1 - create annotations
OPCDATA_OPACTION_ACK	Auto Acknowledge after successful execution of the operator initiated action. 0 - do not auto-acknowledge 1 - auto-acknowledge
OPCDATA_MSG_LOG_ONLY	Message is Server Log Only.
OPCDATA_UNMATCHED	Defines whether or not the message matches a condition. 0 - the message was sent to the server because it matched a match condition 1 - the message did not match a match condition of the assigned templates, but was forwarded nevertheless
OPCDATA_TROUBLETICKET	Forward message to Trouble Ticket System.
OPCDATA_TROUBLETICKET_ACK	Acknowledge message after forwarding it to the Trouble Ticket System
OPCDATA_NOTIFICATION	Notification
OPCDATA_INSTR_IF_TYPE	Type of the instruction interface 0 - OPC_INSTR_NOT_SET 1 - OPC_FROM_OPC 2 - OPC_FROM_OTHER 3 - OPC_FROM_INTERNAL

Variable Name	Short Definition
OPCDATA_INSTR_IF	Name of the external instruction text interface. The external instruction text interface must be configured in OVO.
OPCDATA_INSTR_PAR	Parameters for a call to the external instruction text interface.
OPCDATA_MSGSRC	Message Source. For example, the name of the encapsulated log file if the message originated from log file encapsulation or the interface name if the message was sent via an instance of the Message Stream Interface.
OPCDATA_MSGTEXT	Message Text.
OPCDATA_ORIGMSGTEXT	Original Message Text. Allows you to set additional source information for a message. It is only useful if the message text was reformatted but the OVO operator needs to have access to the original text as it appeared before formatting.
OPCDATA_ANNOTATIONS	This is not a OVO data element. It is created by the adapter product to contain a text string of all the annotations of the message.
OPCDATA_LAST_ANNOTATION	This is not a OVO data element. It is created by the adapter product to contain a text string of the last annotation of the message.

There are also three non-OPC variables added for programmability.

Variable name	Short definition
INSTALLDIR	The home directory where the product was installed.
SC_MSGTYPE	A converted message type string denoting which type of message it is. Valid values are: ■ New message ■ Message Owned by a user ■ Message Disowned by a user ■ Message Acknowledged ■ Message has new annotation(s) ■ All annotations of Message deleted ■ Message was escalated ■ Message was escalated from another server ■ Automatic action of message started ■ Automatic action of message finished ■ Operator initiated action of message started ■ Operator initiated action of message finished
SC_PROBLEM_NUMBER	The ServiceCenter incident number embedded in the OVO Message annotation text when its a new message, or the Object of message Group ServiceCenter when its a ServiceCenter created message.

To check for the OVO Group of OS only to create a new ticket in eventmapsMSI.tcl, do the following:

```
....
if { $OPCDATA_GROUP != "OS" } { return }
....
```

This statement in the beginning of the script returns from the script without creating and queueing a ServiceCenter event to be sent.

ServiceCenter TCL event object

Legend:

<> required parameter

[] optional parameter

Summary of ServiceCenter TCL commands

```
create_sc_event <object name>
<object name> set_evtype <event type> [ use_template <template
name> ]
<object name> set_evfield <0...80>|<field name> <field value>
<object name> print
<object name> send
tcl_logprint
```

create_sc_event

```
create_sc_event <ObjectName>
```

The resultant object from this command will have methods to populate its data members and a send method to queue the event to ServiceCenter. The methods are invoked by using the newly created object name itself and therefore will have its own context. There is a limit of 10 objects that can be created in each script.

```
create_sc_event eventObject
```

The object named eventObject is created and can be used to set values and finally send it to ServiceCenter.

set_evtype

```
<ObjectName> set_evtype <evtype name> [use_template <template
name>]
```

This command sets the evtype field of the event to <evtype name> (e.g., pmo, pmu pmc etc.). There are two optional parameters that can be specified for the event to use a Static Map File for setting the event values:

- use_template
- <template name>

If these optional parameters are not specified, setting of object fields can still be done using integer indexes., as shown in the following examples:

```
create_sc_event eventObject
# create the event object
eventObject set_evtype pmo use_template "EventMap/ToSC/pmo.map"
```

```
# make it of type pmo, and use the template for
# named indexes.
```

The event object eventObject is created. It is then set to type pmo, essentially, the evtype field of the event is set to pmo. The static map file EventMap/ToSC/pmo.map is used to define the named indexes that will be used later to set the other fields.

```
create_sc_event eventObject
# create the event object
eventObject set_evtype pmo
# set the evtype to pmo. do not use templates for indexing
```

The event object eventObject is created. It is then set to pmo for evtype. Because a template was not used here, subsequent setting of evfields will have to use integer indexes.

set_evfield

```
<ObjectName> set_evfield <0 ... 80>|<evfield name> <value>
```

This command sets the event fields with values. If the use_template option was used during the set_evtype command, then you may use field names defined in the static map file templates to set the values, otherwise, you have to use positional integer indexes to set the evfield values. See the following examples:

```
create_sc_event eObject
# create the event object
eObject set_evtype pmo
# set the evtype to "pmo". do not use templates for indexing
eObject set_evfield 1 $OPCDATA_NODENAME
# set evfield position 1 to ITO nodename variable
```

The event object is created, set to evtype “pmo” without using a static map file for template. The first field of the event object is set to the OVO variable for nodename.

```
create_sc_event eObject
# create the event object
eObject set_evtype pmo use_template "EventMap/ToSC/pmo.map"
# set the evtype
# use a templates for indexing
eObject set_evfield logical.name $OPCDATA_NODENAME
# set logical.name evfield to ITO nodename variable
```

The event object is created, set to evtype “pmo” using a static map file for template. The logical.name field of the event object is set to the OVO variable for nodename.

set_evuser

```
<ObjectName> set_evuser <value>
```

This command sets the evuser field of the event to <value>. The maximum length of <value> is 24 characters. The default value for the evuser field is SCITO. For example:

```
# create the event object
create_sc_event eventObject

# set evuser to SCITO-WEST
eventObject set_evuser “SCITO-WEST”
```

print

```
<ObjectName> print
```

This command outputs to stderr all the contents of the object including the template field names if used.

send

```
<objectName> send
```

This command queues the event object to be sent to ServiceCenter's eventin table.

For more examples on how to use the commands, see the example script file.

tcl_logprint

```
tcl_logprint "Message"
```

This command is used to write messages from the tcl script to the SCAuto for OVO log file (`scito.log`).

Example:

```
tcl_logprint "eventmapTTI.tcl: queueing new ticket to be opened.
($OPCDATA_MSGID)"
```

Static map file

The static map file that is usable in a set_evtype command as the template is an ascii text file. Each row denotes a positional field name of the intended ServiceCenter event map starting at index 1, for example, row 1 = field 1 in event map.

By default, three maps for event types pmo, pmu and pmc are delivered. These maps match the ServiceCenter event maps.

Example `pmo.map` used in `eventmapMSI.tcl`.

```
-------------------------- begin file ----------------------------
logical.name
network.name
reference.no
cause.code
action
action2
action3
network.address
type
category
domain
objid
version
model
serial.no.
vendor
location
contact.name
contact.phone
resolution
assignee.name
priority.code
failing.component
system
ci.date.time
flow
server.id
system.state
units
value
port.index
severity.code
site.category
fix.type
```

```
resolution.code
subcategory
product.type
problem.type
adj.resolution.time
explanation1
class
--------------------------- end file-----------------------------
```

The map template file is not a required piece of the configuration; it is used to make it convenient and manageable to name the event fields in the ServiceCenter TCL event object. Without this file, however, you can still set the values using integer indexes.

OVO Message filtering the event.ini file

The `event.ini` file in the `<install dir>/EventMap/ToSC` directory is a means of configuring the scfromitoMSI, scfromitoMEI and scfromitoTTI components of the interface.

In the event.ini file, you can specify:

- How you want to filter MSI event messages
- How you want to filter MEI event messages
- Which TCL map files to execute when a message of type MSI, MEI and/or TTI is available

There are two main structures in the ini file, and this file closely follows the format of a Microsoft Windows ini file format.

Sections

Section header names are identified by enclosing them in square brackets

[....]. The brackets group sets of configurable parameters. There are three required section header names:

- MSI_CONFIG Section for configuring the MSI interface
- MEI_CONFIG Section for configuring the MEI interface
- TTI_CONFIG Section for configuring the TTI interface

Each section header name contains its own set of required parameters.

Section name	Required parameters
MSI_CONFIG	OPC_USER - This is the OVO user the MSI interface will connect as to retrieve message details. OPC_PASSWORD - This is the password for the OPC_USER. Optionally, you may specify the string literal USE_ETC_PASSWD in this field. This parameter suppresses the password check. MSI_OPCREG_MSGTYPE - This is an MSI registration attribute to filter on which OVO message type to notify. This is a String value of maximum 36 characters, no spaces. Consult your OVO Message Source Templates for available message types. If you specify the keyword "ALL" in this field, all messages copying/diverting to the Message Stream Interface will be notified. MSI_OPCREG_GROUP - This is an MSI registration attribute to filter on which OVO message group to notify. This is a String value of maximum 32 characters, no spaces. Consult your OVO Message Source Templates for available message groups. MSI_OPCREG_NODENAME - This is an MSI registration attribute to filter on which OVO node name to notify. This is a String value of maximum 254 characters, no spaces. MSI_OPCREG_OBJECT - This is an MSI registration attribute to filter on which OVO message object to notify. This is a String value of maximum 32 characters, no spaces. Consult your OVO Message Source Templates for available message objects.

Section name	Required parameters
	MSI_OPCREG_SEVERITY - This is an MSI registration attribute to filter on which OVO message severity to notify. This is a String value of 1 or more characters to denote ORing of values. Valid characters are: "c" - critical "w" - warning "n" - normal "u" - unknown "j" - major "m" - minor "cwj" - critical or warning or major. This is the only attribute you don't need the '|' to denote ORing. MSI_OPCREG_APPLICATION - This is an MSI registration attribute to filter on which OVO message application to notify. This is a String value of maximum 32 characters, no spaces. Consult your OVO Message Source Templates for available message applications. POLL_INTERVAL 0 means blocking read for opcif_read call. # means number of seconds to sleep before calling non-blocking opcif_read. MSI_EVENTMAPS - This parameter specifies which section header to fall into, to look for the attribute "mapname" for specifying which TCL script to execute for this interface. **Note:** Values with more than one attribute implicitly assume a logical ANDing of all non-null attributes. **Note:** Within each individual attribute, use the '|' character to logically OR more than one value.

Section name	Required parameters
MEI_CONFIG	OPC_USER - This is the OVO user the MEI interface will connect as to retrieve message details. OPC_PASSWORD - This is the password for the OPC_USER. Optionally, you may specify the string literal USE_ETC_PASSWD in this field. This parameter suppresses the password check. MEI_OPCREG_MSG_EVENT_MASK - This is an OVO Message Event Interface event mask to specify which type of change-in-message should be notified. Valid values are: ■ OPC_MSG_EVENT_OWN - when a message is owned ■ OPC_MSG_EVENT_DISOWN - when a message is disowned ■ OPC_MSG_EVENT_UNACK - when a message is unacknowledged ■ OPC_MSG_EVENT_ACK - when a message is acknowledged ■ OPC_MSG_EVENT_ANNO - when a message is annotated ■ OPC_MSG_EVENT_NO_ANNO - when there are no annotations POLL_INTERVAL: ■ 0 means blocking read for opcif_read call ■ # means seconds to sleep before calling non-blocking opcif_read MEI_EVENTMAPS - This parameter specifies which section header to fall into, to look for the attribute "mapname" for specifying which TCL script to execute for this interface. **Note:** For MEI_OPCREG_MSG_EVENT_MASK, use the '|' character to denote logical OR'ing of event types.
TTI_CONFIG	OPC_USER - This is the OVO user the MEI interface will connect as to retrieve message details. OPC_PASSWORD - This is the password for the OPC_USER. Optionally, you may specify the string literal USE_ETC_PASSWD in this field. This parameter suppresses the password check. TTI_EVENTMAPS - This parameter specifies which section header to fall into, to look for the attribute "mapname" for specifying which TCL script to execute for this interface.

Example event.ini file:

```
------------ begin file ----------------------------------------
[ MSI_CONFIG ]
OPC_USER=opc_adm
OPC_PASSWORD=USE_ETC_PASSWD
MSI_OPCREG_MSGTYPE=ALL
MSI_OPCREG_GROUP=
```

```
MSI_OPCREG_NODENAME=
MSI_OPCREG_OBJECT=
MSI_OPCREG_SEVERITY=
MSI_OPCREG_APPLICATION=
MSI_EVENTMAPS=MSI_EVENTMAPS
POLL_INTERVAL=0

[ MEI_CONFIG ]
OPC_USER=opc_adm
OPC_PASSWORD=USE_ETC_PASSWD
MEI_OPCREG_MSG_EVENT_MASK=OPC_MSG_EVENT_OWN|OPC_MSG_EVENT_DISOWN|O
PC_MSG_EVENT_UNACK|OPC_MSG_EVENT_ACK|
OPC_MSG_EVENT_ANNO|OPC_MSG_EVENT_NO_ANNO
MEI_EVENTMAPS=MEI_EVENTMAPS
POLL_INTERVAL=0

[ TTI_CONFIG ]
OPC_USER=opc_adm
OPC_PASSWORD=USE_ETC_PASSWD
TTI_EVENTMAPS=TTI_EVENTMAPS

[ MSI_EVENTMAPS ]
mapname=EventMap/ToSC/eventmapMSI.tcl

[ MEI_EVENTMAPS ]
mapname=EventMap/ToSC/eventmapMEI.tcl

[ TTI_EVENTMAPS ]
mapname=EventMap/ToSC/eventmapTTI.tcl

---------------- end file ----------------------------------------
```

Using TTI - trouble ticketing interface

To use the TTI:

1. Identify which message source you want to forward to the TTI in the Message Source Template and enable it.

2. In the Node Bank submap OVO, select the following menu commands, in order: Actions->Utilities->Trouble Ticket.

3. Enable the interface by clicking on the check box.

4. Enter in the Call of trouble ticket field <install dir>/TTI.sh. <install dir> is the SCAuto product install directory.

Default behavior

The scfromitoMSI process picks up all events that copy or divert to the Message Stream Interface. If the host name of the occurring event is different, ServiceCenter creates a new incident ticket. If the host name of the occurring event is the same, the same ticket is updated by default, and a new ServiceCenter OVO event is created with a ServiceCenter Group attribute. You can use the ServiceCenter Message Group in the OVO Message Group Bank to filter the messages.

The scfromitoMEI process only picks up events that were previously opened as ServiceCenter incident tickets or ServiceCenter event group events if the message for the incident or event has been modified by the OVO message browser. This modification can consist of an operator owning, acknowledging, or annotating a ServiceCenter event type or a OVO event that caused a incident ticket. When this type of modification occurs, a pmu is generated to modify the same incident ticket in ServiceCenter.

Example eventmapMSI.tcl Script

```
--------------------------- begin file ---------------------------
```

[comments removed]

```
# create only if its a new message from OPC
if { $SC_MSGTYPE != "New message" } {
    tcl_logprint "eventmapMSI.tcl: skipping event - $SC_MSGTYPE"
    return
}

# skip ServiceCenter events
if { $OPCDATA_GROUP == "ServiceCenter" } {
    tcl_logprint "eventmapMSI.tcl: skipping ServiceCenter event -
$SC_MSGTYPE"
    return
}

# skip Normal severity events
if { $OPCDATA_SEVERITY == "8" } {
    tcl_logprint "eventmapMSI.tcl: skipping \"Normal\" severity
event - ($OPCDATA_MSGID)"
    return
}

# OPCDATA_CREATION_TIME processing
# save the incoming (seconds) value in new variable
# format as previous release
```

```
# create value that SC will like for a date/time field
set OPCDATA_CREATION_TIME_SECONDS $OPCDATA_CREATION_TIME
set OPCDATA_CREATION_TIME [clock format
$OPCDATA_CREATION_TIME_SECONDS -format "%a %b %d %T %Y"]
set OPCDATA_CREATION_TIME_SC [clock format
$OPCDATA_CREATION_TIME_SECONDS -format "%m/%d/%Y %T" -gmt true]

# do the same for OPCDATA_RECEIVE_TIME
set OPCDATA_RECEIVE_TIME_SECONDS $OPCDATA_RECEIVE_TIME
set OPCDATA_RECEIVE_TIME [clock format
$OPCDATA_RECEIVE_TIME_SECONDS -format "%a %b %d %T %Y"]
set OPCDATA_RECEIVE_TIME_SC [clock format
$OPCDATA_RECEIVE_TIME_SECONDS -format "%m/%d/%Y %T" -gmt true]

# create the event object
create_sc_event eventObject

#set evuser to this hostname
#eventObject set_evuser "SCITO-[lindex [split [info hostname] .]
0]"

# set the event type using a template to define field names
# * if you don't use a template, you can use integer
# * indexes into the evfield array.
eventObject set_evtype pmo use_template "EventMap/ToSC/pmo.map"

# start mapping field names to ITO values
eventObject set_evfield logical.name $OPCDATA_NODENAME
eventObject set_evfield network.name $OPCDATA_NODENAME
eventObject set_evfield reference.no $OPCDATA_MSGID
eventObject set_evfield cause.code $OPCDATA_MSGSRC
eventObject set_evfield action [ format "%s on %s (Original
message: %s) %s\n<ITOMSGID:%s,%s>\n" $OPCDATA_MSGTEXT
$OPCDATA_CREATION_
TIME $OPCDATA_ORIGMSGTEXT $OPCDATA_LAST_ANNOTATION $OPCDATA_MSGID
$SC_MSGTYPE ]
eventObject set_evfield network.address $OPCDATA_NODENAME
eventObject set_evfield type ITOEvent
eventObject set_evfield category example

# convert ITO severity to ServiceCenter severity code
# proc is defined in util.tcl
source "$INSTALLDIR./EventMap/ToSC/util.tcl"
set SC_SEVERITY [ ConvertSeverity "$OPCDATA_SEVERITY"]
#tcl_logprint "eventmapMEI.tcl: Converted ITO severity
$OPCDATA_SEVERITY to SC severity.code $SC_SEVERITY"

eventObject set_evfield severity.code $SC_SEVERITY
eventObject set_evfield priority.code $SC_SEVERITY
```

```
eventObject set_evfield failing.component [ concat $OPCDATA_GROUP
"," $OPCDATA_OBJECT "," $OPCDATA_APPLICATION ]
eventObject set_evfield system $OPCDATA_MSGSRC


# print out a debug of event created
#eventObject print

# send the event to queue
tcl_logprint "eventmapMSI.tcl: queueing new ticket to be opened.
($OPCDATA_MSGID)"
eventObject send
---------------------------- end file ----------------------------
```

TCL Event Mapping from ServiceCenter to OVO

The mapping mechanism to map ServiceCenter events into OVO messages is done by invoking a TCL script. ServiceCenter event fields are made into TCL_GLOBAL variables that can be accessed in the targeted script with the $VARIABLE syntax. Various OVO programming APIs are also made into TCL_Command commands callable from within the same script, thus effectively bridging the two domains.

Configuration overview

By default, the mapping files reside in an EventMap directory within the product installation directory. The hierarchy and function of the EventMap directory structure follows:

- EventMap
- EventMap/FromSC/ - files to configure events from ServiceCenter to OVO.
- EventMap/ToSC/ - files to configure events from OVO to ServiceCenter

The following files are located in the EventMap/FromSC/ directory:

event.ini	The configuration file that specifies OVO user and password, and TCL scripts to invoke based on event types.
pmo.tcl	The event map script to invoke when there is a pmo (Incident Opened).
pmu.tcl	The event map script to invoke when there is a pmu (Incident Updated).

pmc.tcl	The event map script to invoke when there is a pmc (Incident Closed).
util.tcl	Utility TCL script that contains command to convert ServiceCenter priority codes to OVO Severity values.
default.tcl	A default TCL script to invoke when an event type is received and there is no equivalent TCL script.
showEvent.tcl	A debug script that will display all event field values from a ServiceCenter event.

The `event.ini` file configures the OVO user name and password, along with the TCL scripts that are invoked based on the evtype field of the incoming ServiceCenter event.

ServiceCenter TCL variables

The available variables depend on the evtype type field and are in accordance with the event maps from Events Services in ServiceCenter for that type. These variables are made available as TCL_GLOBAL and will be accessible via the $VARIABLE syntax globally from within the called TCL script. See the Event Services User's Guide for more detail about event maps.

Variable name	Short description
INSTALLDIR	The directory path where the product was installed. This is not part of an event field and is added for convenience in locating helper scripts.
SCEVTYPE	ServiceCenter registration name for the event, for example, pmo.
SCEVTIME	Date and time event occurred.
SCEVSYSSEQ	ServiceCenter eventout queue sequence number. Used as checkpoint in sync file.
SCEVUSRSEQ	A user-assigned sequence number used to trace an event through ServiceCenter.
SCEVSYSOPT	A code to identify system options.
SCSCEVUSER	The event user name; if passed, it is used as the operator name.
SCEVPSWD	The event user's password.
SCEVSEPCHAR	The character used to separate fields in the $SCEVFIELDS variable. Default is "^".

Variable name	Short description
SCEVFIELDS	The data describing the event, with fields separated by the $SCEVSEPCHAR character.
SCEVFIELDS_1..2..3 ...*	The event fields from the $SCEVFIELDS string parsed out as individual field values using the $SCEVSEPCHAR separator value.

Note: The specific field definition depends on the evtype coming back and is documented in the <evtype>.tcl scripts themselves. It is up to the user to define types that do not have a corresponding TCL script.

OVO programming APIs as TCL commands

LEGEND:

```
< >   required parameter
[ ]   optional parameter
```

These are OVO Programming APIs that have been wrapped in TCL commands and made available during the script invocation. Specifically, the available APIs consists of opcif_* types for writing to the Message Stream Interface, and opcmsg_* types for retrieving message details from the database.

Summary of OVO TCL commands

- `opcif_write <Message Text> <Application> <Message Group> <Message Type> <Node Name> <Object> <Severity>`
- `opcmsg_annotation_add <Message Id> <Annotation Text>`
- `opcmsg_ack <Message Id>`
- `opcmsg_unack <Message Id>`
- `opcmsg_own <Message Id>`
- `opcmsg_disown <Message Id>`
- `opcmsg_escalate <Message Id>`
- `opcmsg_op_action_start <Message Id>`
- `tcl_logprint`

opcif_write

```
opcif_write <Message Text> <Application> <Message Group> <Message
Type> <Node Name> <Object> <Severity>
```

The opcif_write command creates a new OVO message. To pass a null value, use "" for an empty string instead. There are seven required arguments to this command:

Message Text	The message text of the newly created OVO message.
Application	The application that this message belongs to. Check your Message Source Templates for valid application names.
Message Group	The group that this message belongs to. Check your Message Source Templates for valid application names.
Message Type	The type that this message belongs to. Check your Message Source Templates for valid application names.
Node Name	The OVO node name that this message is created for.
Object	The object that this message belongs to. Check your Message Source Templates for valid application names.
Severity	The OVO severity value for this message. Check the utility script `EventMap/FromSC/util.tcl` for the conversion utility to convert from ServiceCenter priority codes. Valid values are: c - Critical j - Major m - Minor w - Warning n - Normal

Example:

```
# source in the util.tcl script to get the ConvertSeverity command
source "$INSTALLDIR./EventMap/FromSC/util.tcl"

# construct a formatted message text to be passed to opcif_write
set OPC_Message_Text "Incident ticket $SCEVFIELDS_2 opened on
$SCEVFIELDS_4 by $SCEVFIELDS_5 for Message ID $SCEVFIELDS_14\n
Category: $SCEVFIELDS_3\n   Assigned to: $SCEVFIELDS_9\n
Severity: $SCEVFIELDS_8\n   CauseCode: $SCEVFIELDS_17\n
Hostname: $SCEVFIELDS_18\n   Group: $SCEVFIELDS_19\n   Location:
$SCEVFIELDS_21\n   Action: $SCEVFIELDS_26\n   Contact:
$SCEVFIELDS_32 $SCEVFIELDS_34\n   NetworkName: $SCEVFIELDS_35\n
Resolution: $SCEVFIELDS_37\n   UpdateAction: $SCEVFIELDS_38"

# convert ServiceCenter severity code to ITO Severity indicator
set SCSEVERITY [ ConvertSeverity "$SCEVFIELDS_8" ]

# create the new ITO message
opcif_write "$OPC_Message_Text" "ServiceCenter_pmo"
"ServiceCenter" "TroubleTicket" "$SCEVFIELDS_18" "pmo"
"$SCSEVERITY"
```

In this example, a new OVO message is created with a formatted message text from ServiceCenter event fields of Application="ServiceCenter_pmo", Group="ServiceCenter",Type="TroubleTicket", node name from ServiceCenter event logical.name field, Object="pmo" and a converted Severity that is equivalent to the ServiceCenter priority code.

opcmsg_annotation_add

```
opcmsg_annotation_add <Message Id> <Annotation Text>
```

This command adds an annotation to an existing OVO message. This command requires two arguments, consisting of a OVO message ID and free-form annotation text.

Example:

```
# pre-format an annotation text to use from ServiceCenter field
values
set OPC_Annotate_Text "Incident ticket $SCEVFIELDS_2 opened on
$SCEVFIELDS_4 by $SCEVFIELDS_5\n   Category: $SCEVFIELDS_3\n
Assigned to: $SCEVFIELDS_9\n   Severity: $SCEVFIELDS_8"

# annotate and existing ITO message
opcmsg_annotation_add "$SCEVFIELDS_14" "$OPC_Annotate_Text"
```

The previous example pre-formats annotation text to be used to annotate an existing OVO message identified by the message ID stored in event field index 17 of the evfields string.

opcmsg_ack

```
opcmsg_ack <Message Id>
```

Given the message ID of an existing OVO message, this command will acknowledge it as the user defined in the `event.ini` file.

opcmsg_unack

```
opcmsg_unack <Message Id>
```

Given the message ID of an existing OVO message, this command will unacknowledge it as the user defined in the event.ini file.

opcmsg_own

```
opcmsg_own <Message Id>
```

Given the message ID of an existing OVO message, this command will own it as the user defined in the event.ini file.

opcmsg_disown

```
opcmsg_disown <Message Id>
```

Given the message ID of an existing OVO message, this command will disown it as the user defined in the `event.ini` file.

opcmsg_escalate

```
opcmsg_escalate <Message Id>
```

Given the message ID of an existing OVO message, this command will escalate it as the user defined in the `event.ini` file.

opcmsg_op_action_start

```
opcmsg_op_action_start <Message Id>
```

Given the message ID of an existing OVO message, this command will start the pre-defined operator action that the user defined in the `event.ini` file.

tcl_logprint

```
tcl_logprint “Message”
```

This is used to write messages from the tcl script to the SCAuto for OVO log file (`scito.log`).

Example:

```
tcl_logprint “pmo.tcl: ITO message created for pmo. $SCEVFIELDS_2"
```

Event configuration file

The `event.ini` file in the `<install dir>/EventMap/FromSC` directory is a means of configuring the sctoito component of the interface. In here, you may specify:

- The user name/password to connect to OVO to retrieve message details from its database.
- The TCL map script to invoke when a certain event type is read from ServiceCenter.

Sections

Section header names are identified by square brackets "[....]". The brackets group sets of configurable parameters. There are two required section header names:

CONFIG	Section for specifying OVO username/password to connect as.
EVENT	Section for configuring event types to the TCL map script to execute.

Each section header name has its own set of required parameters.

Section name	Required parameters
CONFIG	OPC_USER - The OVO user name. OPC_PASSWORD - This is the password for the OPC_USER. Optionally, you may specify the string literal USE_ETC_PASSWD in this field. This parameter suppresses the password check.
EVENT	<event type>=<TCL map script to execute> For example: pmo=EventMap/FromSC/pmo.tcl This will tell the interface to execute the script <install dir>/EventMap/FromSC/pmo.tcl when a "pmo" event is received from ServiceCenter.

Default `event.ini` file.

```
---------------------------- begin file --------------------------
[ CONFIG ]
OPC_USER=opc_adm
OPC_PASSWORD=USE_ETC_PASSWD

[ EVENT ]
pmo=EventMap/FromSC/pmo.tcl
pmu=EventMap/FromSC/pmu.tcl
pmc=EventMap/FromSC/pmc.tcl
---------------------------- end file ----------------------------
```

Default behavior

When there is a pmo in the eventout table (such as when a ticket is newly created), sctoito treats it as an acknowledgment of the new incident ticket and annotates the originating event. sctito also creates a new ServiceCenter opened event with the IM number being the Object.

When there is a pmu in eventout (such as when its created by a pmu in eventin, or when a ServiceCenter operator modifies the ticket), if the actor or user field is not SCITO, sctoito annotates the originating event and creates a new ServiceCenter update event with the IM number being the Object. This prevents automatic annotations from OVO from going into an infinite loop, updating ServiceCenter and vice-versa.

Similarly, when eventout has a pmc that is not from user SCITO, the originating event will be annotated and a new ServiceCenter closed OVO event will be generated.

Default pmo.tcl script

```
[comments deleted]
#*****************************************************************
# BEGIN data block
# Define data value mapping between ServiceCenter event variables
# to OVO variables to be used by OPC commands in action block.
#*****************************************************************

set OPC_Annotate_Text "Incident ticket $SCEVFIELDS_2 opened on
$SCEVFIELDS_4 by $SCEVFIELDS_5\n   Category: $SCEVFIELDS_3\n
Assign
ed to: $SCEVFIELDS_9\n   Severity: $SCEVFIELDS_8"

set OPC_Message_Text "Incident ticket $SCEVFIELDS_2 opened on
$SCEVFIELDS_4 by $SCEVFIELDS_5 for Message ID $SCEVFIELDS_14\n
Categ
ory: $SCEVFIELDS_3\n   Assigned to: $SCEVFIELDS_9\n   Severity:
$SCEVFIELDS_8\n   CauseCode: $SCEVFIELDS_17\n   Hostname:
$SCEVFIELD
S_18\n   Group: $SCEVFIELDS_19\n   Location: $SCEVFIELDS_21\n
Action: $SCEVFIELDS_26\n   Contact: $SCEVFIELDS_32
$SCEVFIELDS_34\n
  NetworkName: $SCEVFIELDS_35\n   Resolution: $SCEVFIELDS_37\n
UpdateAction: $SCEVFIELDS_38"

set OPC_Application "ServiceCenter_pmo"

set OPC_Message_Group "ServiceCenter"

set OPC_Message_Type "TroubleTicket"

set OPC_Node_Name "$SCEVFIELDS_18"

set OPC_Object "$SCEVFIELDS_2"

# convert ServiceCenter severity code to ITO severity
# proc is defined in util.tcl
set rc [catch {set SCSEVERITY [ ConvertSeverity "$SCEVFIELDS_8"
]}]

set OPC_Severity "$SCSEVERITY"

#*****************************************************************
```

```
# BEGIN Action block
# Define ITO message updating/creating logic using variables
# mapped in data block.
# See GUIDE.txt for available OPC Tcl commands and their
# parameters.
**********************************************************************
#
# first try to annotate original message, if message doesnt exist,
# create a new one.
#
# also put in special identifying text **** WARNING **** do not
# modify set rc [catch [opcmsg_annotation_add "$SCEVFIELDS_14"
# "$OPC_Annotate_Text \n\n<scito:$SCEVFIELDS_2,pmo>\n" ] ]

if {$rc == 0} {
tcl_logprint "pmo.tcl: ITO message annotated for pmo.
$SCEVFIELDS_2"
}

#
# lets make a new ServiceCenter message.
#
set rc [catch [opcif_write "$OPC_Message_Text" "$OPC_Application"
"$OPC_Message_Group" "$OPC_Message_Type" "$OPC_Node_Name" "$OPC_Ob
ject" "$OPC_Severity" ] ]

if {$rc == 0} {
tcl_logprint "pmo.tcl: ITO message created for pmo. $SCEVFIELDS_2"
} else {
tcl_logprint "pmo.tcl: After running opcif_write pmo,
$SCEVFIELDS_2, ret = ($rc) $errorCode\n$errorInfo."
}

#
# now lets try to own the message.
#
#set rc [catch [opcmsg_own "$SCEVFIELDS_14" ] ]
#
#if {$rc == 0} {
#tcl_logprint "pmo.tcl: ITO message owned. $SCEVFIELDS_2"
#} else {
#tcl_logprint "pmo.tcl: ITO message unable to own. $SCEVFIELDS_2"
#}
```

OVO business logic configuration

This section contains a step-by-step process for configuring OVO to enable automatic event message integration with ServiceCenter. This process is part of the normal configuration process upon installation of SCAuto for OVO.

General process

Requirements analysis

To take full advantage of OVO and ServiceCenter integration, you need a high-level plan for application integration. While the integration can automatically create incident tickets from any events managed by OVO, a defined set of application requirements enables you to focus this general capability into specific measurable results.

For example, you can choose to have all kernel alarms open incident tickets automatically, but this process will create hundreds of incident tickets. If the design is reviewed from a requirements standpoint, where the service desk is pursuing operations in accordance with a service level management (SLM) agreement , perhaps it can be refined to open tickets only on critical events from a small set of critical hosts. In this case, defining the requirements helps meet the specific goals of the SLM agreement by notifying the service desk of potentially huge impacts on meeting the contract.

The process of defining requirements precedes the configuration of any software. It consists of analyzing your business and application and service desk requirements, and organizing these requirements to suit your needs. It is the first step in the Analyze, Design, and Implement process described in this section.

Design phase

After you have defined the requirements of your consolidated service desk environment, you can determine the specific monitored sources help needed to meet these requirements. This is the Design step of the process. In OVO, this amounts to identifying what Message Sources are relevant. This may be done for individual Message Sources or for groups. Beyond noting which Message Sources are relevant, the next step is to define the parameterization and configuration of each relevant Message Source.

Implementation phase

With a coherent design that meets the requirements, it is then possible to modify the OVO Message Source Templates to meet this design specification. This is the Implement step of the process.

After the modified templates are saved, they need to be pushed to the managed nodes to which they apply. This step, which is the final configuration action, acknowledges that the modified templates are controlled centrally from OVO but are distributed broadly to the nodes or groups that are specified as the appropriate recipients of the policy.

Design considerations

When designing your system, consider these guidelines:

- If you expect to receive many of the same types of messages that will cause OVO to ServiceCenter interaction, use the Message Stream Interface (MSI). SCAuto for OVO support of MSI involves a daemon process that is constantly waiting for new messages in the message stream. This is not processor intensive, and it is implemented to be a lightweight background process for performing regular, repeated event message processing.
- If you expect to receive fewer and less frequent, and consequently more important messages, use the Trouble Ticket Interface (TTI). SCAuto for OVO support of TTI involves a single thread of execution that is started upon receipt of the event message. This amounts to a processing intensive approach that offers a direct message delivery.

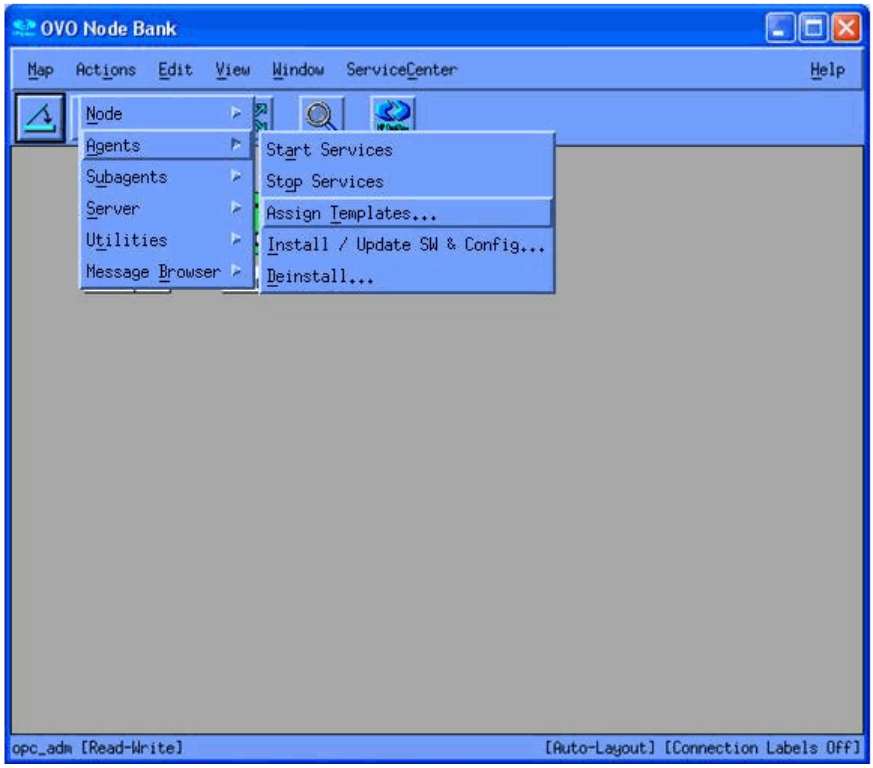
Implementation steps

The following steps describe how to modify a Message Source Template to activate automated event messages from OVO to ServiceCenter. These steps are modeled from a process that implemented a high-level goal, stated as:

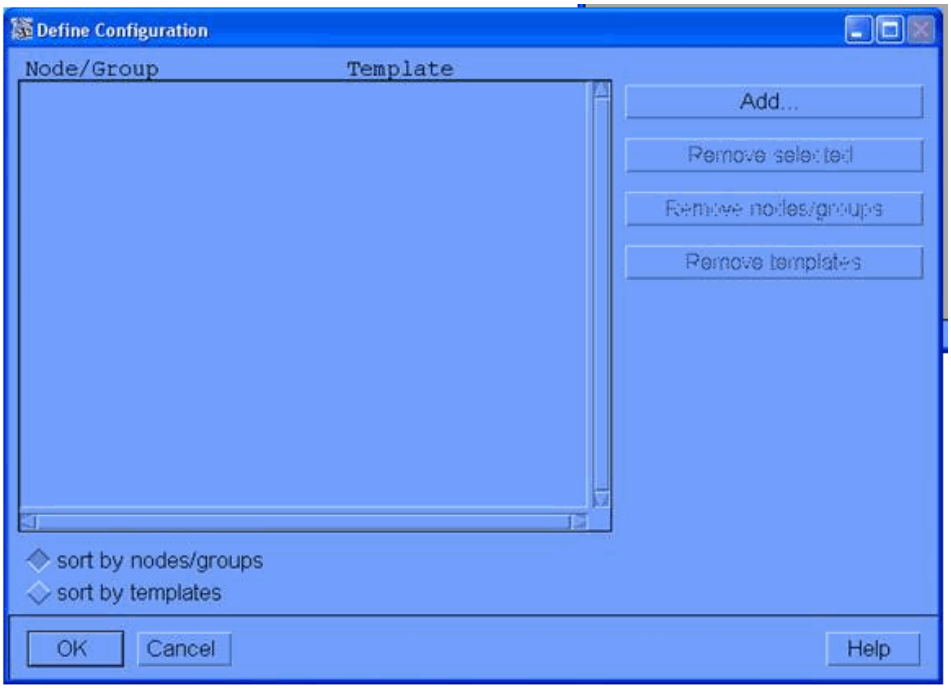
```
Integrate Security alerts into Incident Tickets where any failed
switch user (su) command causes OVO alerts.
```

Step 1 Review templates assigned to a node.

From the OVO Node Bank window, go to the Actions menu, click Agents, then click Assign Templates.



The Define Configuration window opens.

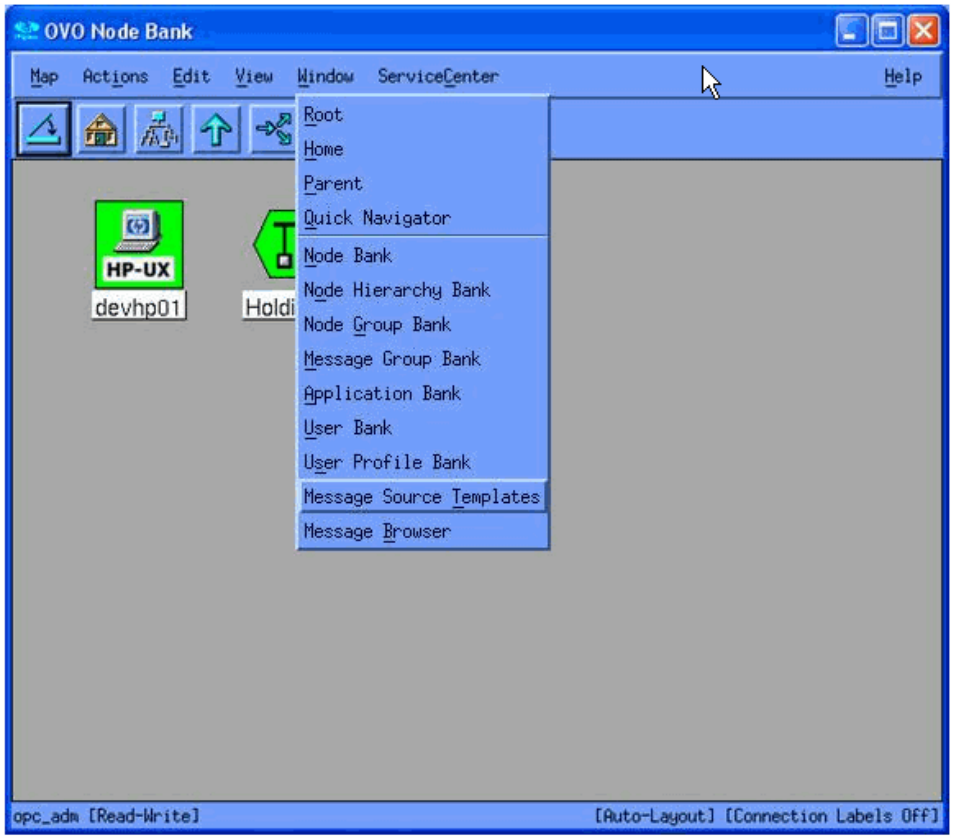


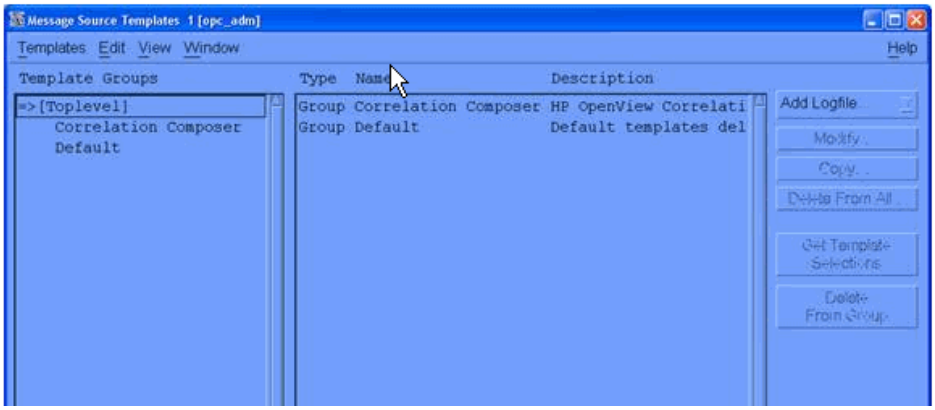
In this window, identify which templates have been assigned to specific nodes or groups. If the template that you want is already assigned, you need to push it out again to its subscribers once you complete your modifications. Plan to do this in addition to, or along with, the template that you push out in Step 7.

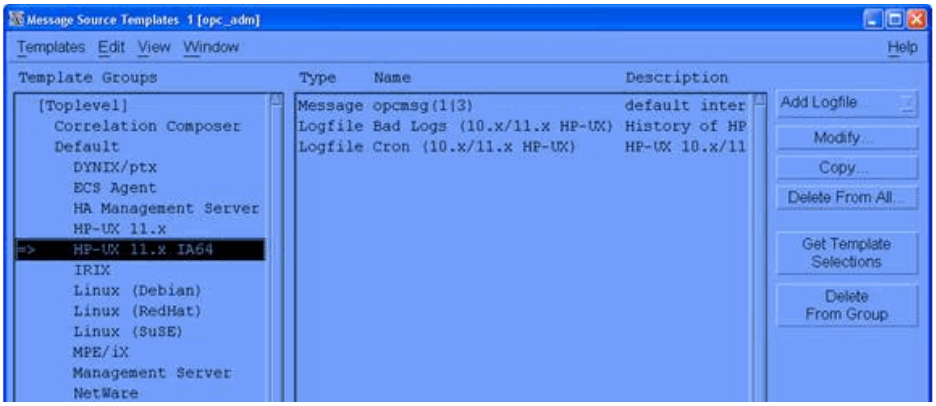
Step 2 Begin modifying templates (to assign to a node)

In the OVO Node Bank window, go to the Window menu and click Message Source Templates. The Message Source Templates window is displayed. This window contains two panes:

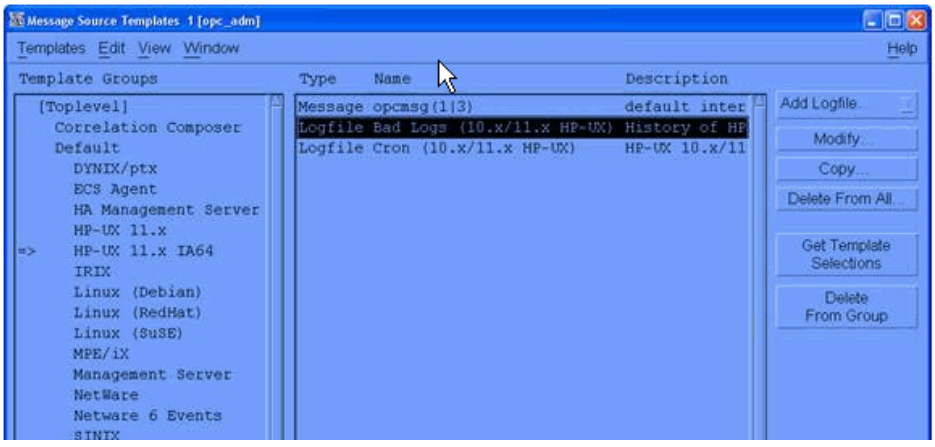
- The left pane contains the message source groups available for you to select.
- The right pane displays the unique Message Source Templates in a selected group. Navigate the groups by clicking on items to show values in the right pane or double-clicking to expand items in the left pane.



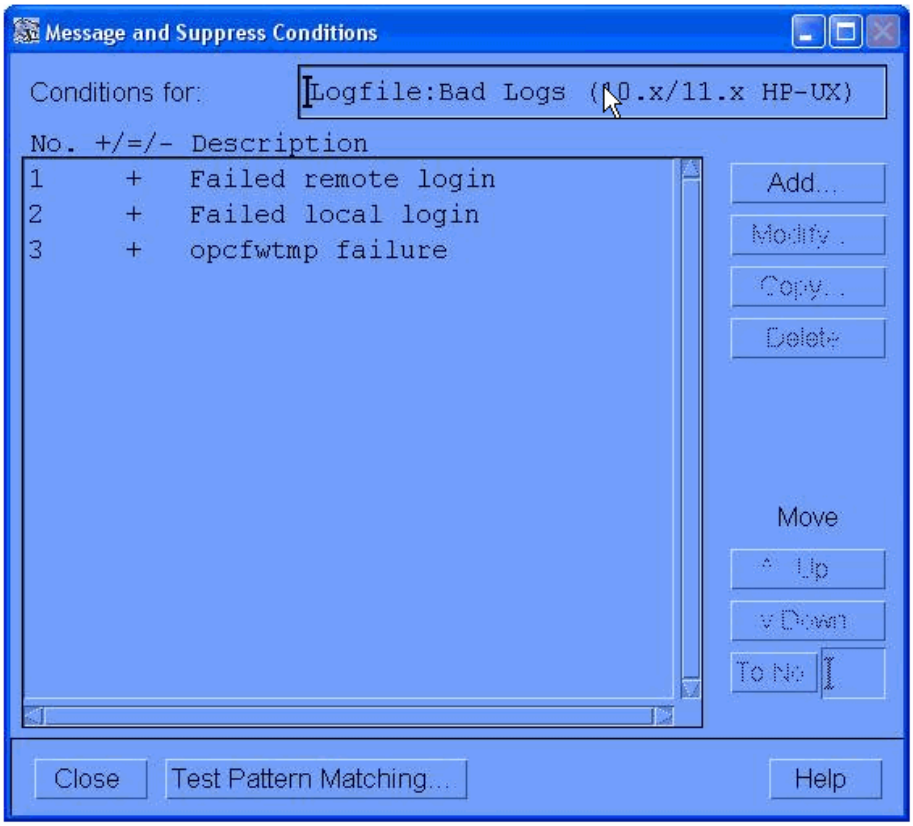




When you find the group you are interested in, select it in the left pane. The HP-UX 11.x IA64 group is used in this example.



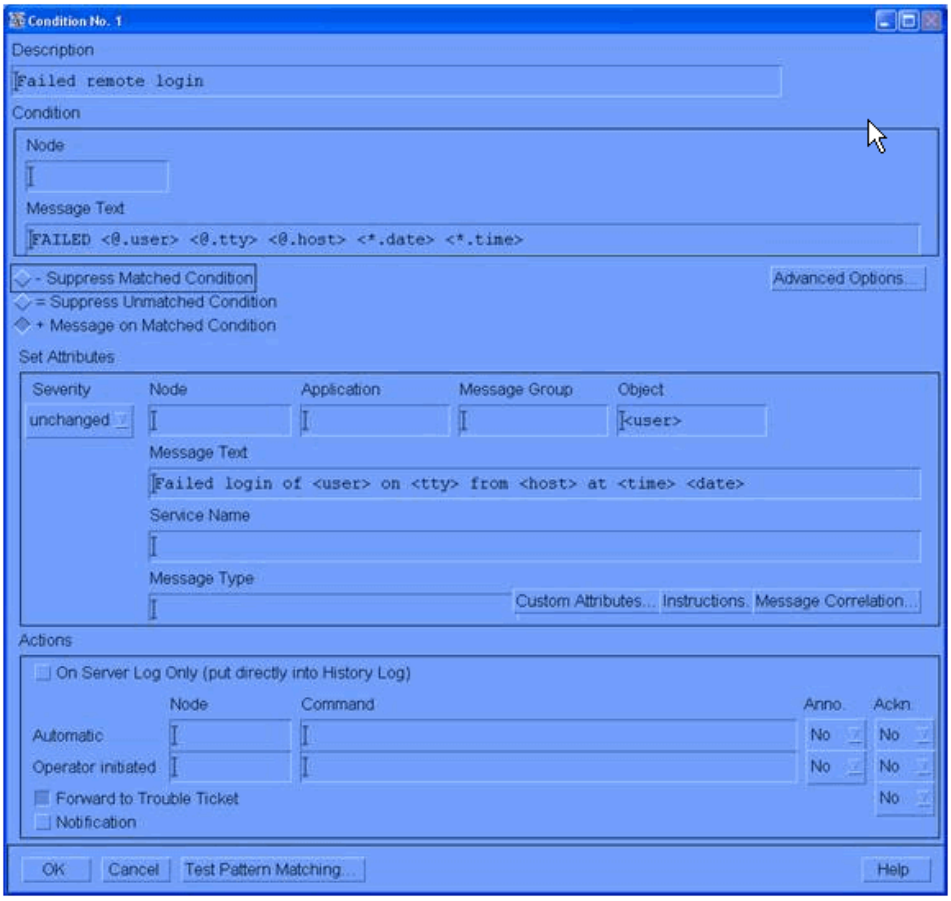
The right pane contains the members of the selected group. For the HP-UX 11.x IA64 group, there is a Logfile: Bad Logs item. When this item is selected, it activates the Conditions button. Press this button to edit the configuration of the Logfile Bad Logs template. This displays the Message & Suppress Conditions window.



Step 3 Modify templates (to assign to a node)

From the Message & Suppress Conditions window, ensure that the conditions are set as you require. For example, to suppress a condition, be sure that it has a minus sign (-) in the window. To activate a condition, it needs a plus sign (+). To change the setting, select the condition and press the Modify button. This

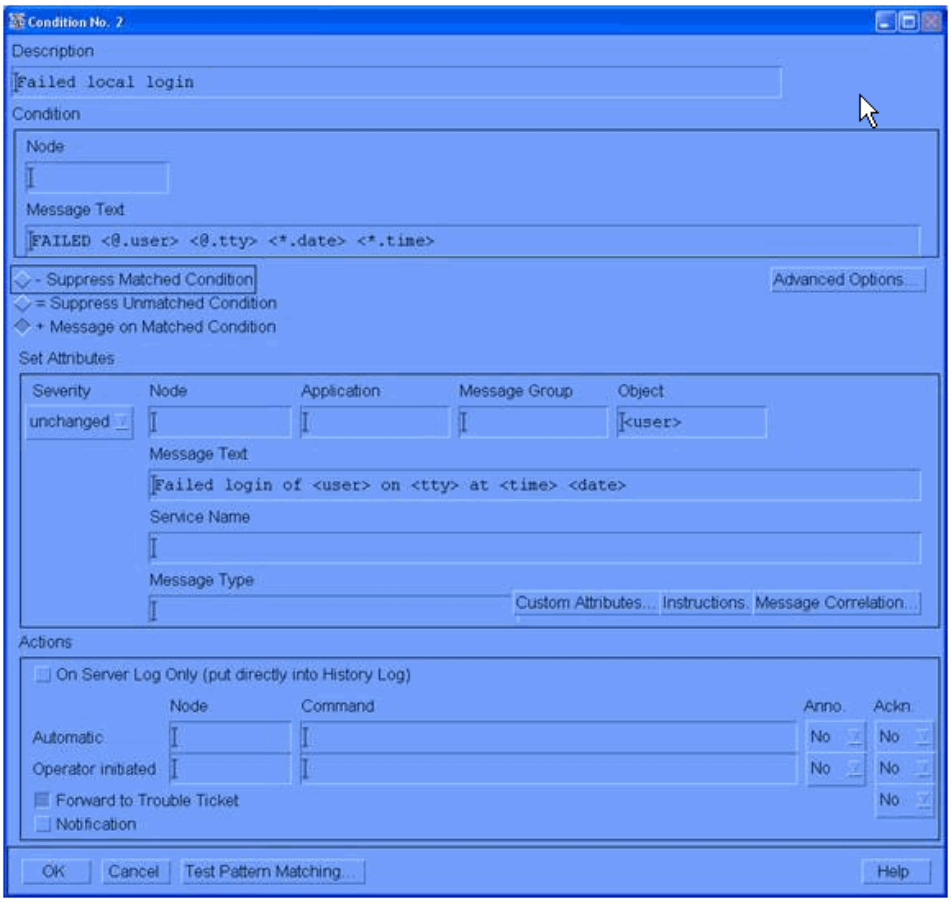
displays the Condition No. <N> window (where N is the number of the condition in the list, according to the condition you select).



Step 4 Modify template conditions to forward to trouble ticket

To invoke the TTI interface, select the Forward to Trouble Ticket check box on the Condition No. <N> window. This action completes the Message Source Template configuration. Save your edits by clicking the OK button (closes the Condition No. <N> window), and proceed to Step 7 - Install Templates to Selected Nodes.

Note: If you wish to use the MSI interface, proceed to the next step without configuring the Trouble Ticket Interface.



This form is the same place where you would set up automatic actions, such as executing a command on the node that generated this event message. Certain attributes can also be set here. These attributes can then be used for processing actions in SCAuto for OVO maps (scripts) and can later be used within ServiceCenter. The attribute settings will apply regardless of the choice of MSI or TTI setting.

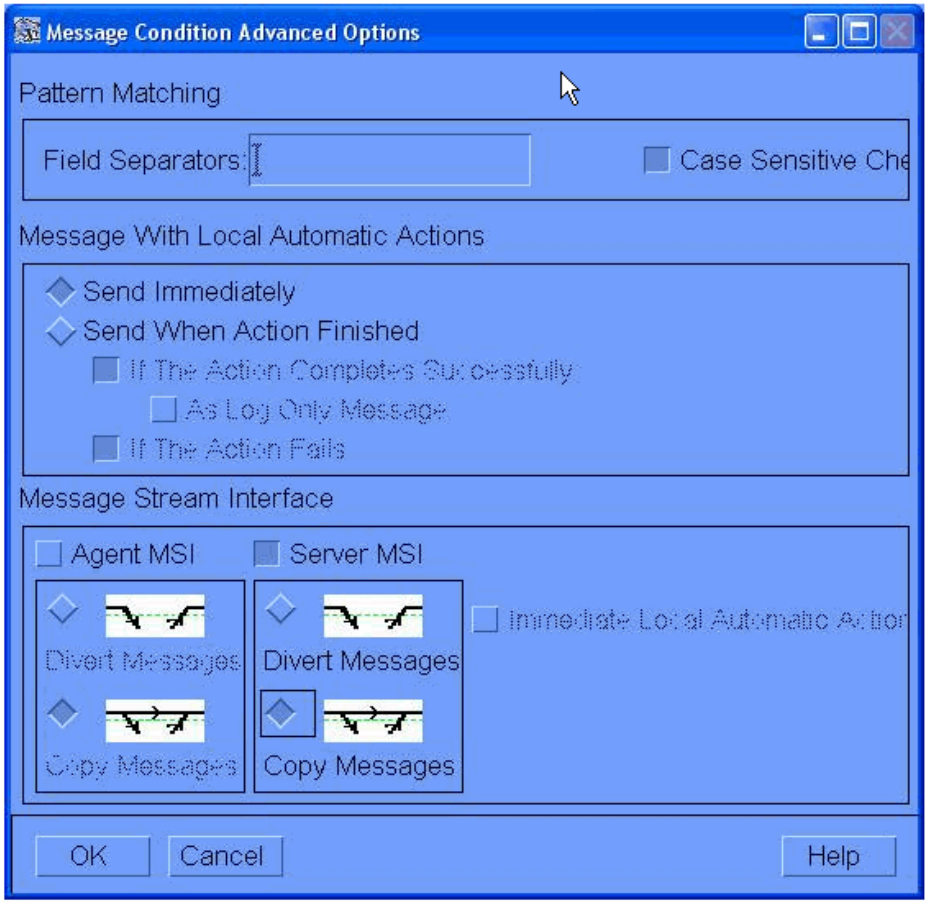
For example, the Message Group attribute in the center of this form can be set to variable values or literal values (constants). This attribute value is passed to SCAuto for OVO in an environment variable named $OPCDATA_GROUP. For example, the Message Group attribute can be given a value of TEST in the form.

Then, the SCAuto for OVO TCL script can use the $OPCDATA_GROUP as the name of the category variable used in the ServiceCenter event message. This implies that a new ticket will be created in the TEST incident ticket category, if it exists.

Furthermore, if you create OVO Message Groups that match ServiceCenter incident ticket categories, there will be both a high-level logical relationship between your OVO implementation and your ServiceCenter implementation, and you will also have a detailed connection through this use of message source template attributes.

Step 5 Modify template conditions to use message stream interface.

To use the MSI interface, click the Advanced Options button in the Condition No. <N> window. The Message Conditions Advanced Options window opens.



In the Message Conditions Advanced Options window, there is a section labeled Message Stream Interface. Within this section, there are two areas: Agent MSI and Server MSI. Activate just the Server MSI interface by checking the box. Then, select the Copy Message button once the Server MSI interface is activated.

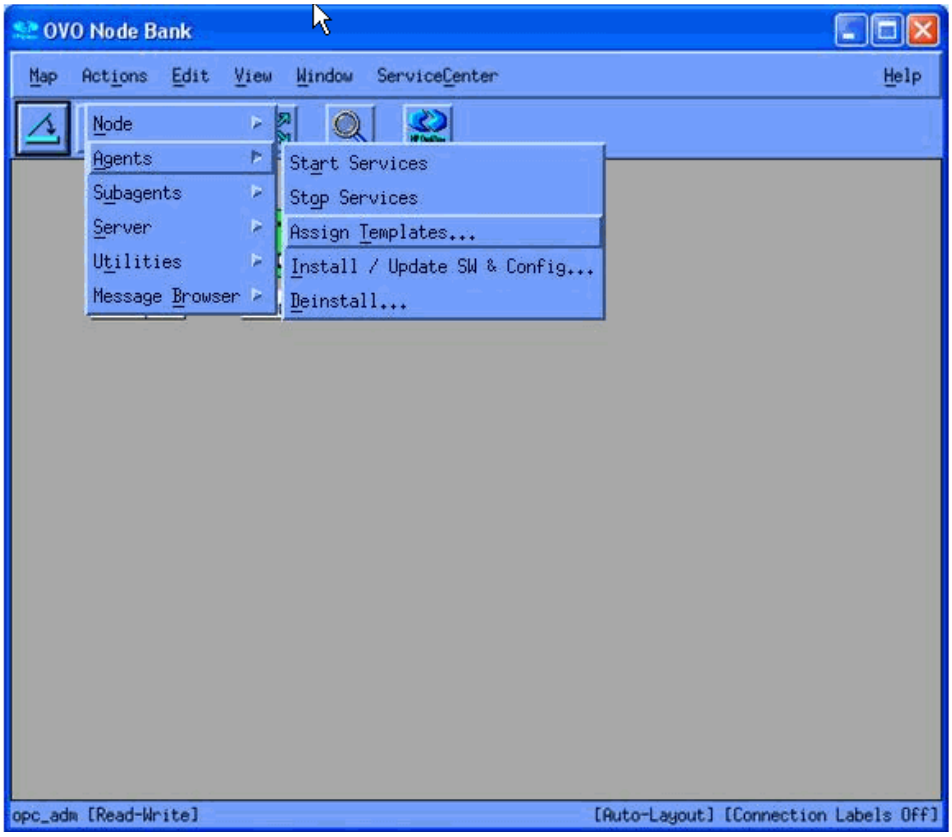
Save your edits by clicking the OK button. This closes the Message Conditions Advanced Options window. Continue to click the OK or Close buttons in the remaining open windows until you return to the Message Source Templates window.

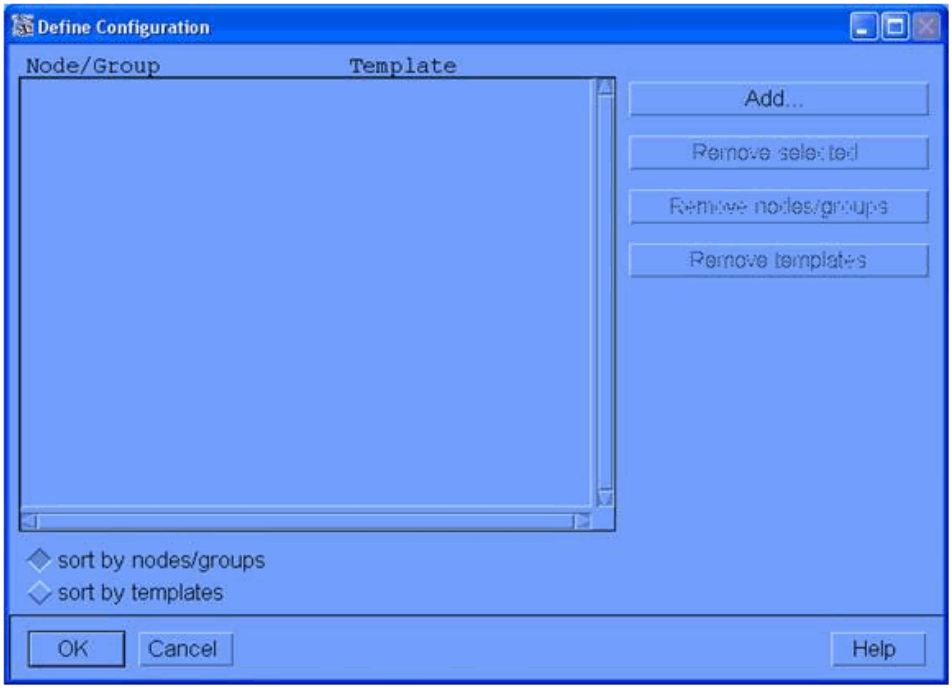
Using this form, you can also set up various Message Stream parameters, such as suppressing duplicate event messages.

Note: If you leave the Message Source Template window open, with the selected (now modified) template, this streamlines step 6.

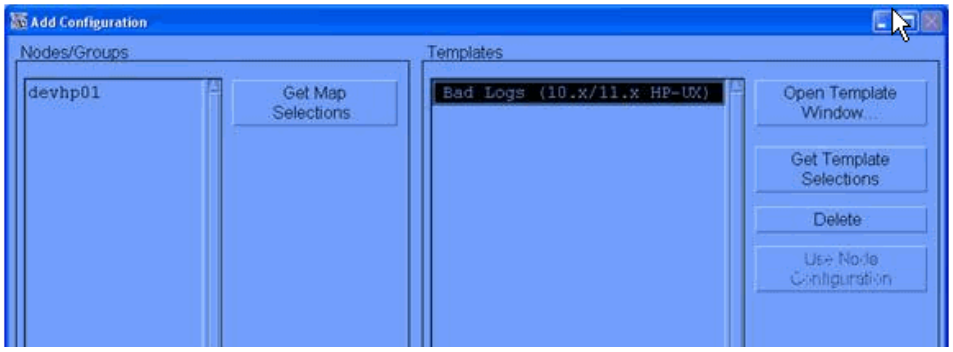
Step 6 Add modified template to configuration.

From the OVO Node Bank window, go to the Actions menu and click Agents. Then, click Assign Templates to display the Define Configuration window. This process repeats the commands and actions of step 1.





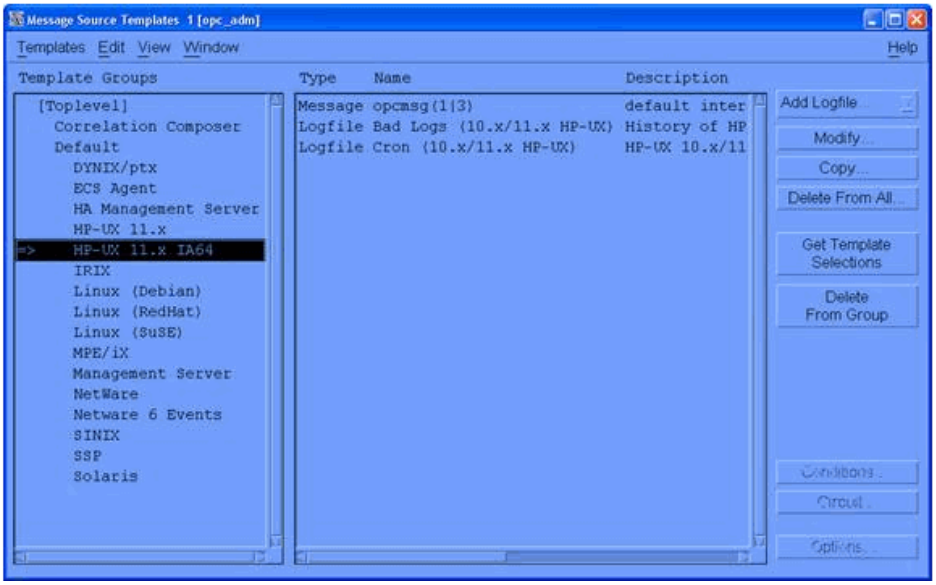
In the Define Configurations window, click the Add button to launch an Add Configurations window.

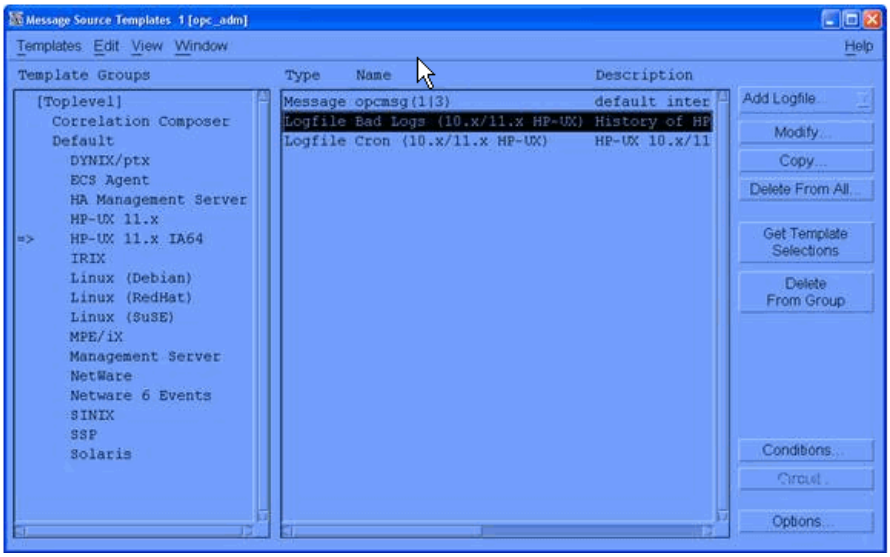


This window contains a Get Template Selections button. This is used, for example, when you have an unusual workflow. In this case, you would open a new window, select your modified template in the new window, and then return to the first window and click the Get Template Selections button to import your selection.

If your Message Source Template window is still open from the last step, and contains a selected template, you can click the Get Template Selections button to complete this step. Otherwise, continue with the instructions here.

To open the window where you can select a template, click the Open Template Window button in the Add Configurations window. This displays the Message Source Templates window.





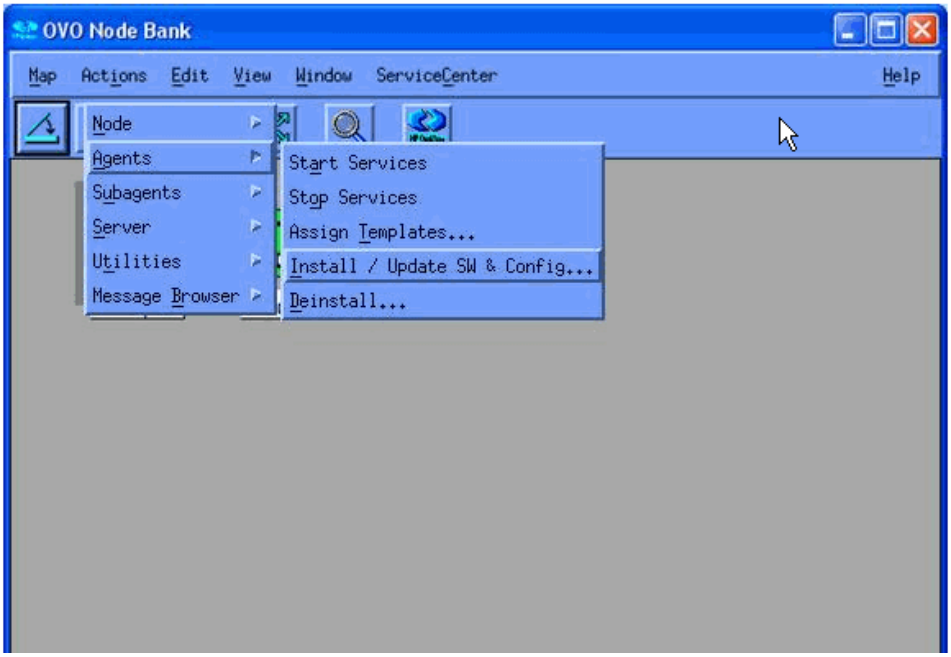
When you finish assigning your modified templates, click the OK button to save your edits. This returns you to the Define Configuration window. Verify that your newly assigned template is displayed in this window. Press the OK button until you return to the OVO Node Bank window. This completes the steps to define and add the desired templates.

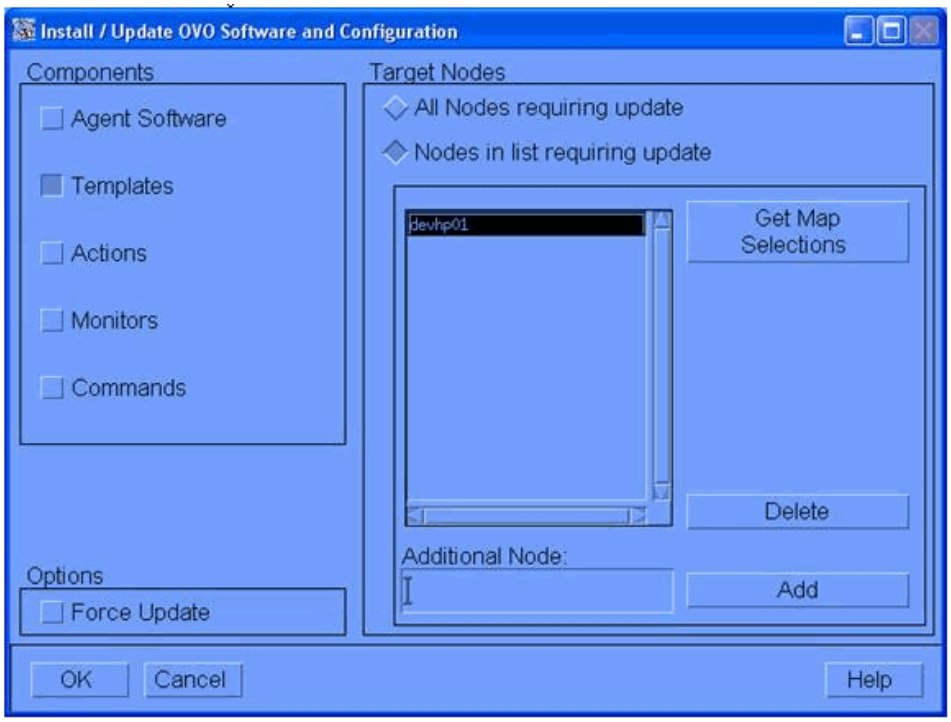
The Message Source Templates window may still be open from step 5. You can close it now as well.

Step 7 Install Templates to Selected Nodes

This step involves "pushing" the configuration to the endpoints. Once the message source templates are installed at the endpoints, any activity that triggers the template at the nodes will cause OVO event messages and subsequent activation of SCAuto for OVO.

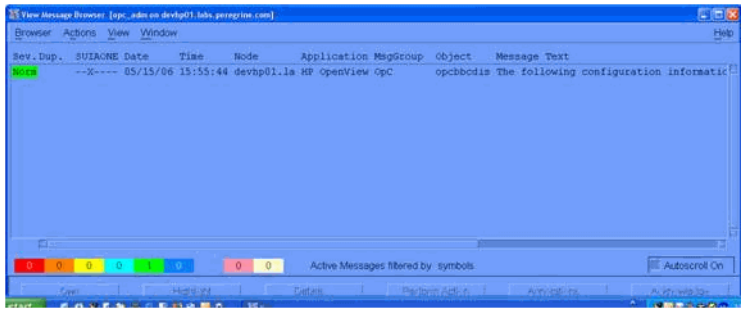
From the OVO Node Bank window, highlight one or more nodes to receive the configuration. On the Actions menu, click Agents, then click Install / Upgrade S/W Configuration. This displays the Install/Upgrade OVO Software and Configuration window.





Verify that your node or group is in the list of target nodes. You must also verify that the Templates checkbox is selected in the Components area in the left part of the window. Your message source may also require Actions, Monitors, and Commands.

When you click OK, the template configuration is sent to the node and it replaces any existing configuration at the node. If you have an OVO message browser open, you will see an opcdista type of event message that confirms that the configuration has been received by the endpoint .



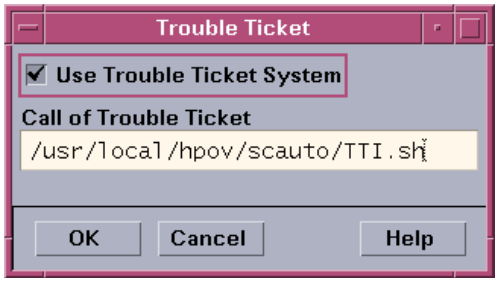
Step 8 Final configuration for use of trouble ticket interface

If you chose to use the TTI interface, there is one additional configuration step. You need to specify which program OVO should run whenever the TTI function is invoked.

From the OVO Node Bank window, go to the Actions menu and click Utilities. Then, click Trouble Ticket to display the Trouble Ticket dialog box.

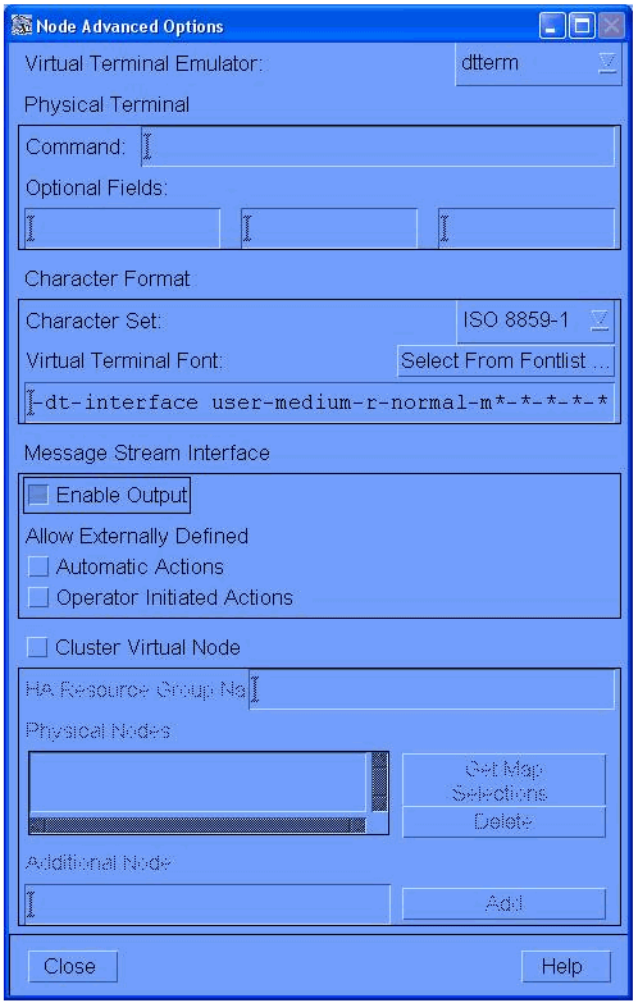
In the Trouble Ticket dialog box, select the Use Trouble Ticket System checkbox. In the Call of Trouble Ticket field, enter a full path name and script name to invoke the SCAuto for TTI program. (The program's name is `TTI.sh`. The default location for this is `/opt/OV/scauto/TTI.sh`; however, your installation may have placed this program in a different location.)

When you click OK, OVO will look for and validate that the program is available and executable.



Step 9 Final configuration for use of message stream interface.

From the OVO Node Bank window, select Actions > Server > Configure. In the Configure Management Server window under Message Stream Interface, select Enable Output.

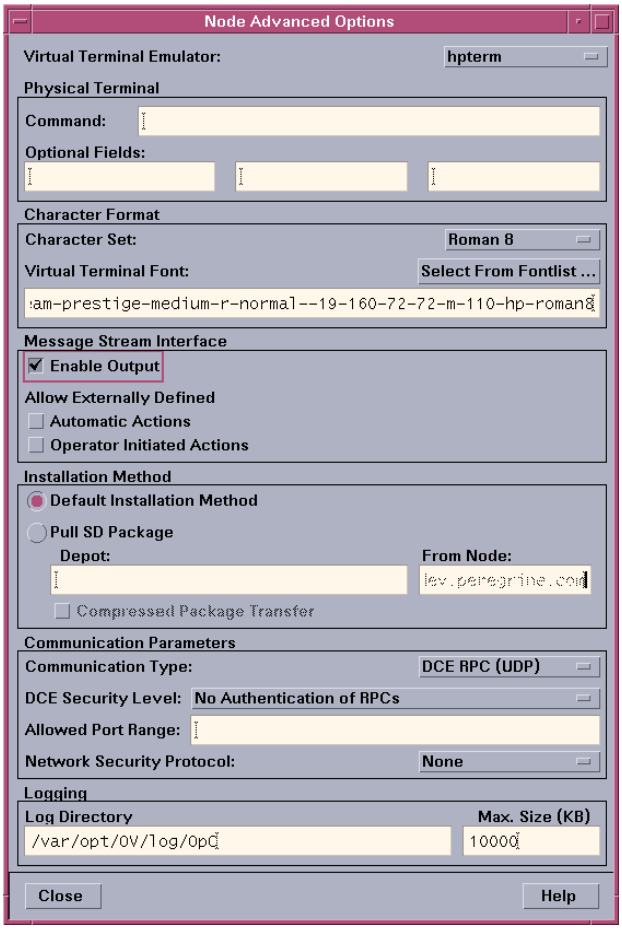


There is one more parameter that must be verified before the configuration is complete. To allow specific nodes to use the MSI or not, OVO has a parameter related to enabling outputs from nodes to the MSI.

From the OVO Node Bank window, select one or more nodes. Right-click the mouse, and click the Modify command. This opens the Modify Node window. Click on the Advanced Options button, and the Node Advanced Options

window appears. In the section of this window labeled Message Stream Interface, click on the Enable Output checkbox. This is also the place to enable automatic actions and operator initiated actions.

To save your edits, click OK until you get back to the OVO Node Bank window. You are now fully configured for automated event integration between OVO and ServiceCenter.

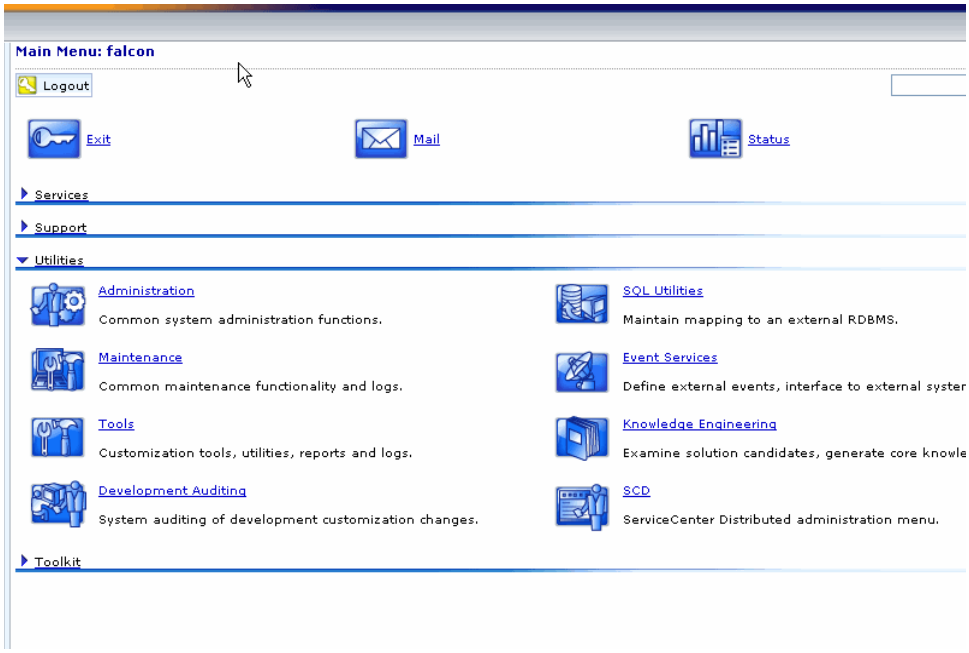


This completes the configuration of the OVO business logic inherent in the Message Source Templates.

ServiceCenter business logic configuration

ServiceCenter Event Services

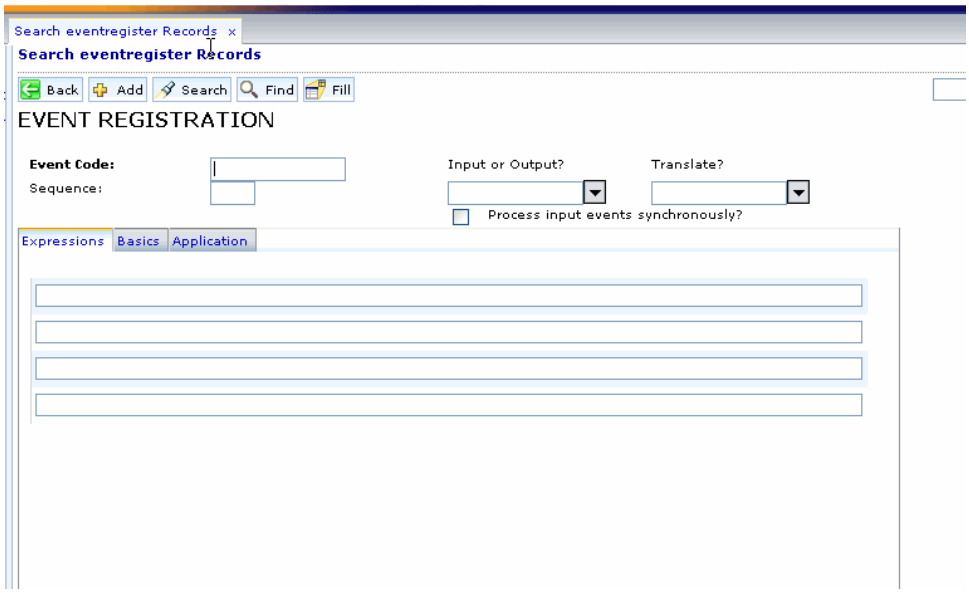
To access Event Services, click the Event Services button on the Main ServiceCenter menu. Select the Administration tab to reach Event Registration, Filters, and Maps.



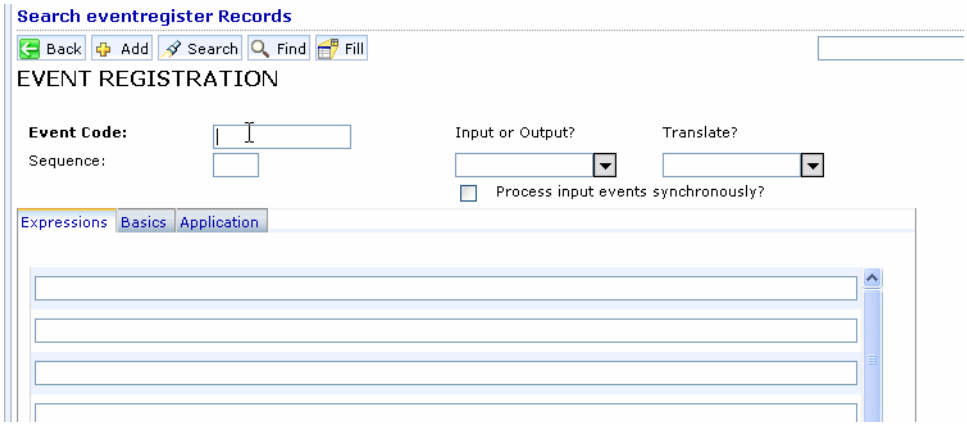
Event registration

On the Administration tab, click the Registration button to display the Event Registration form, which contains the definitions for events processed by the

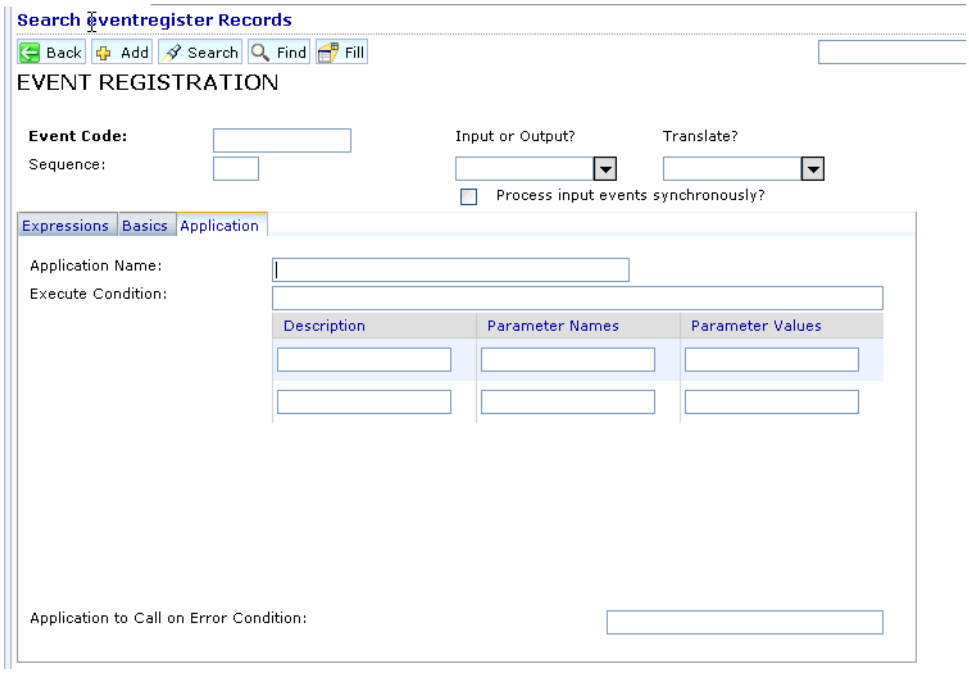
system. The Event Registration form contains three tabs: Expressions, Basics, and Application.



The Expressions tab displays the processing logic associated with the event type (for example, pmo).

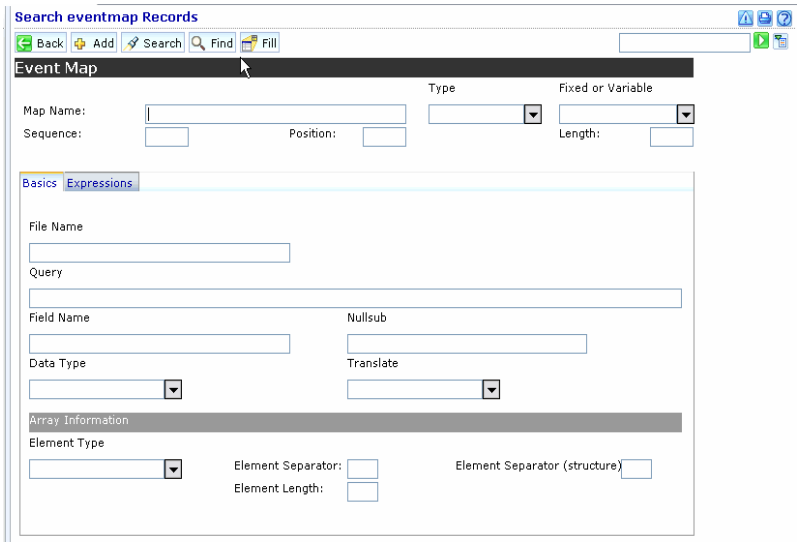


The Applications tab displays the RAD application that is associated with the event, as well as the parameters that are used when running the application upon processing of the event.

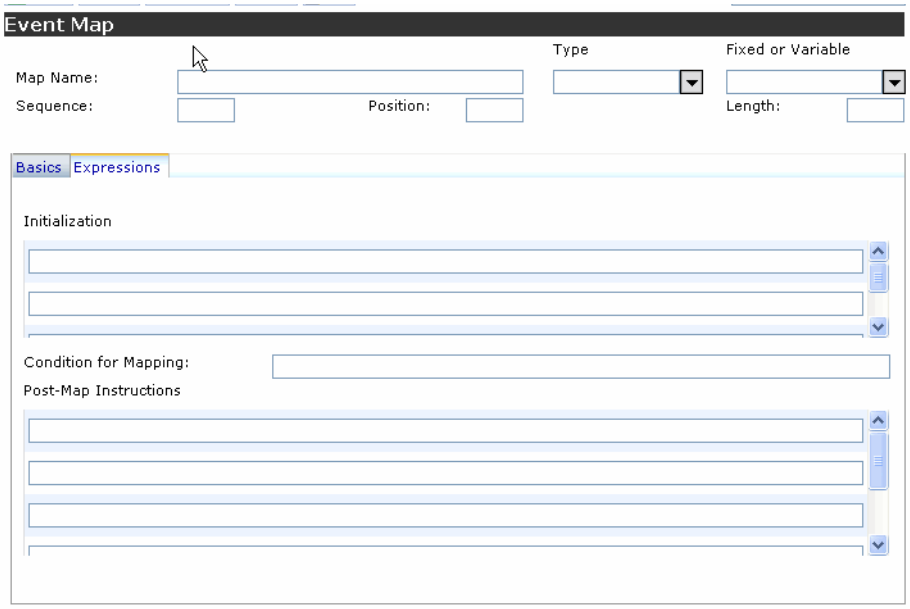


Event maps

Event Maps are the guides to recording and processing events, including data type and the name of the file where event data is stored.



The Expressions tab contains additional instructions for event processing.

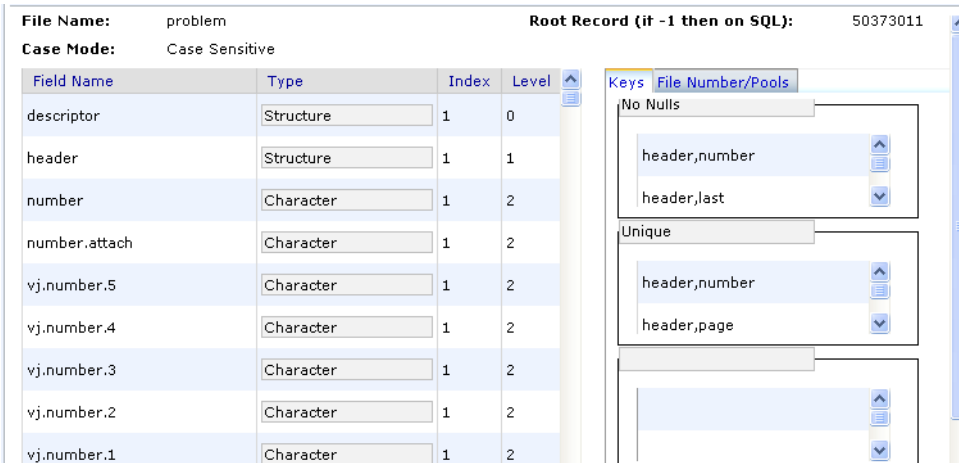


All Event Services features are addressed in greater detail in the *Event Services User's Guide* or ServiceCenter online help. See this guide if you plan to make modifications discussed in this section.

ServiceCenter Incident Management

You may want to customize and configure the definition of incident ticket formats and management in ServiceCenter. Based on your needs, you may need to create additional incident categories to reflect the automatic incident ticket processing supported by the SCAuto for OVO product. You can use the incident category named example as a guide or template to build new automated ticket categories and their associated forms and database dictionary definitions.

The following figure shows a portion of the database dictionary structure for the incident table. This can be used as reference for custom categories based on the probsummary table. For more detailed information, refer to the "Database Dictionary" section in the Base Utilities Guide.



Event data is channeled and presented in ServiceCenter according to controls defined in the Format Control records associated with specific incident category formats (display forms). Refer to the Format Control Guide for complete information on Format Control in ServiceCenter.

Additionally, the Forms Designer tool is used to create custom forms for added incident categories and customizations to accommodate the automated incident ticket generation from SCAuto for OVO events. Refer to the *Forms*

Designer Guide or ServiceCenter online help for more details on form (format) development and customization.

6 Scenarios

CHAPTER

This chapter provides scenarios of different high-level configurations that can be accomplished using ServiceCenter Automation (SCAuto) for OpenView Operations (OVO). Depending on your business needs, the following examples may be implemented individually or in combination to achieve your business goals.

Uni-directional automatic incident ticket creation (Mode 1)

This implementation is described in Chapter 1, "Introduction," as Mode 1 of the Operational Concepts.

In this mode, OVO events drive ServiceCenter tickets. This occurs through the registered connection from OVO to ServiceCenter via the MSI API. This mode requires the following components of SCAuto for OVO:

- scfromitoMSI and scevmon processes
- ToSC Queue file
- Event Maps (event.ini, TCL scripts, and maps referenced in event.ini)

You may start the adapter processes using the HP OpenView "ovstart <process name>" facility. Since by default, all the other SCAuto for OVO components are installed and configured to execute, using the integrated GUI menu options in the Root/Node Bank Windows will start all the adapters.

You may be able to customize and remove the unwanted components from the menu by doing the following:

- Modify these files:
  - `$OV_CONF/OpC/mgmt_sv/ui/registration/C/opc_adm/scauto`
  - `$OV_CONF/OpC/mgmt_sv/ui/registration/C/opc_adm/scauto`
- Under the Action “startALL”, “stopAll”, and “statusAll” sections, remove the components that you do not want to start/stop.

Since only the scfromitoMSI and scevmon processes are required in this implementation, you can use the LRF files supplied in the `<installed directory>/lrf_files` directory and execute `ovdelobj scfromitoMEI.lrf`, and ovdelobj sctoito to remove the registration of these components.

In addition, you can modify the LRF for scfromitoMSI and scevmon and change the first parameter to OVs_YES_START, and then execute `ovaddobj <filename>`. This causes the HP OVO ovstart facility to start the process by default.

For more information, refer to the *HP OpenView Administrator's Guide*, or the man pages for “ovaddobj” and “ovdelobj” commands as well as the man pages for "lrf".

Bi-directional incident ticket/OVO message creation/update/close (Mode 2)

This implementation is described in Chapter 1, “Introduction,” as Mode 2 of the Operational Concepts. This is the out-of-box default behavior.

In this mode, OVO events drive ServiceCenter tickets, and the ServiceCenter tickets control OVO events. This amounts to a partnership of managing the events, where each application provides a significant contribution to the event and incident management process. This mode is constructed through the

registered connection from OVO to ServiceCenter via the MSI API, as well as the MEI API. This mode requires the following components of SCAuto for OVO:

- scfromitoMSI, scfromitoMEI, sctoito, and scevmon processes
- ToSC and FromSC Queues
- Event Maps (event.ini, TCL scripts, and maps referenced in event.ini)

You can start or stop the adapter processes using the OVO ovstart <process name> facility or the integrated GUI menu option from the Root or Node Bank windows.

In addition, you can modify the LRF (in the `<installed directory>/lrf_files` directory) for these components and change the first parameter to OVs_YES_START, and execute ovaddobj <filename>. This causes the HP OpenView ovstart facility to start the process by default.

Refer to the *HP OpenView Administrator's Guide* or the man pages for ovaddobj and ovdelobj commands as well as the man pages for "lrf".

Creating incident tickets with trouble ticket interface (Mode 3)

This implementation is described in Chapter 1, "Introduction", as Mode 3 of the Operational Concepts.

This mode is an alternative configuration of Mode 1 (Uni-directional) or Mode 2 (Bi-directional). It uses the OVO TTI API instead of the MSI API. This allows for a more focused configuration of OVO Message Source Templates, freeing the MSI API to be used for other integration efforts, if desired. MEI usage with this mode is identical to the previous modes.

This mode requires the following components of SCAuto for OVO:

- scfromitoTTI and SCEVMON processes
- ToSC and FromSC Queues

- Event Maps (event.ini, TCL scripts, and maps referenced in event.ini)
- IT/O configurations for TTI to execute <installed directory>/TTI.sh, as well as source templates to Forward message to Trouble Ticket Interface.

Because this implementation does not require the execution of any of the other OVO interfaces, you can remove them from the GUI integration as well as the SPMD (ovstart) profile, if desired.

You can start or stop the scevmon process using the OVO ovstart scevmon facility. Since by default, all the other SCAuto for OVO components are installed and configured to execute, using the integrated GUI menu options in the Root/Node Bank Windows will start all the adapters.

You may be able to customize and remove the unwanted components from the menu by doing the following:

- Modifying these files:
  - `$OV_CONF/OpC/mgmt_sv/ui/registration/C/opc_adm/scauto`
  - `$OV_CONF/OpC/mgmt_sv/ui/registration/C/opc_adm/scauto`
- Under the Action startALL, stopAll, and statusAll sections, remove the components that you do not want to start or stop.

Because only the scevmon process is required in this implementation, you can use the LRF files supplied in the `<installed directory>/lrf_files` directory and execute `ovdelobj <component file name>` to remove the registration of these components.

In addition, you can modify the LRF scevmon and change the first parameter to OVs_YES_START, and then execute `ovaddobj <scevmon LRF filename>`. This causes the OVO ovstart facility to start the process by default.

Refer to the *HP OpenView Administrator's Guide*, or the man pages for ovaddobj and ovdelobj commands as well as the man pages for lrf.

Node-based incident tickets

By default, after installing the product, the To ServiceCenter TCL scripts are configured to generate pmo events with the category of example. Also, the default behavior for the example category in ServiceCenter is to match incident tickets based on the logical.name field in the pmo event. If this is the desired effect, you do not have to customize the product.

If you want the pmo to go into a specific ServiceCenter category, you must take the following steps:

- Make sure that the targeted category contains format control logic to handle a pmo the same way as the example category.
- Customize the TCL scripts to generate the desired category in the resulting pmo event.

Refer to Chapter 5, "Configuration," for more information about these steps.

Cause-based incident tickets per node

This implementation is primarily designed for a typical OVO configuration. It essentially takes the node-based concept of OpenView NNM to another level, to an Application-based or Cause-based event per node in the managed environment.

For example, in a node-based system, every event that is configured to be captured by OVO on the same node will funnel into the same incident ticket, regardless of the nature or cause of the message. In contrast, a Cause-based or Application-based system makes a distinction, for example, between a Security event and an OS event by opening different incident tickets for these events. By default, ServiceCenter and SCAuto for OVO are node-based.

To configure for a Cause-based or Application-based system, the default ServiceCenter Event Registration expression must be modified in ServiceCenter to match pmo events with existing incident tickets based on the logical.name, category, and cause.code fields in the pmo event.

OVO message group into ServiceCenter category

This implementation facilitates the management of large numbers of OVO event messages by funneling them into different ServiceCenter categories based on the OVO Message Group name. This enables the OVO Administrator to control which OVO message opens a incident ticket in which category. This grouping of incident tickets by OVO Message Groups is a logical approach to eventually assigning incident tickets to the same group of technical support staff based on their expertise.

To configure for incident tickets to be opened in categories specified by the Message Group of the OVO Message:

- In ServiceCenter, create categories matching the OVO Message Group names. This will later enable the SCAuto for OVO adapter to assign events to these specific categories based on the OVO Message Group of the interested event. Also, make sure that the correct event registration and format control logic are built into these new categories.
- In SCAuto for OVO, modify the default "ToSC" TCL scripts to generate specific category and cause.code in the resulting pmo. By default, the category field is hard-wired to the literal example. If desired, you can use the O message group value to populate this field by changing the line:

```
eventObject set_evfield category example

to

eventObject set_evfield category $OPCDATA_GROUP
```

6/22/06