InCharge™

Service Assurance Manager User's Guide for NetIQ AppManager Adapter

Version 1.0

smarts

Cisco Part Number: OL-6061-01

the technology of Crystal Decisions. The use of all Crystal Decisions software products is subject to a separate license agreement included with the Software electronically, in written materials, or both. YOU MAY NOT USE THE CRYSTAL DECISIONS SOFTWARE UNLESS AND UNTIL YOU READ, ACKNOWLEDGE AND ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE AGREEMENT. IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE, YOU MAY RETURN, WITHIN THIRTY (30) DAYS OF PURCHASE, THE MEDIA PACKAGE AND ALL ACCOMPANYING ITEMS (INCLUDING WRITTEN MATERIALS AND BINDERS OR OTHER CONTAINERS) RELATED TO THE CRYSTAL DECISIONS' TECHNOLOGY, TO SMARTS FOR A FULL REFUND; OR YOU MAY WRITE, CRYSTAL WARRANTIES, P.O. BOX 67427, SCOTTS VALLEY, CA 95067, U.S.A.

GNU eTeks PJA Toolkit

The eTeks PJA Toolkit is resident on the CD on which the Software was delivered to you. Additional information is available at eTEks' web site: http://www.eteks.com. The eTeks PJA Toolkit program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation; version 2 of the License. The full text of the applicable GNU GPL is available for viewing at http://www.gnu.org/copyleft/gpl.txt. You may also request a copy of the GPL from the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. The eTeks PJA Toolkit program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For a period of three years from the date of your license for the Software, you are entitled to receive under the terms of Sections 1 and 2 of the GPL, for a charge no more than SMARTS' cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code for the GNU eTeks PJA Toolkit provided to you hereunder by requesting such code from SMARTS in writing: Attn: Customer Support, SMARTS, 44 South Broadway, White Plains, New York 10601.

IBM Runtime for AIX

HP-UX Runtime Environment for the Java™ 2 Platform

The Software contains the HP-UX Runtime for the Java™ 2 Platform, distributed pursuant to and governed by Hewlett-Packard Co. ("HP") software license terms set forth in detail at: http://www.hp.com. Please check the Software to determine the version of Java runtime distributed to you.

DataDirect Technologies

NetBSD

RSA Data Security, Inc.

AES

License Terms:

The free distribution and use of this software in both source and binary form is allowed (with or without changes) provided that:

1. distributions of this source code include the above copyright notice, this list of conditions and the following disclaimer;
2. distributions in binary form include the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other associated materials;
3. the copyright holder's name is not used to endorse products built using this software without specific written permission.

ALTERNATIVELY, provided that this notice is retained in full, this product may be distributed under the terms of the GNU General Public License (GPL), in which case the provisions of the GPL apply INSTEAD OF those given above.

Disclaimer: This software is provided 'as is' with no explicit or implied warranties in respect of its properties, including, but not limited to, correctness and/or fitness for purpose. Issue Date: 26/08/2003

Contents

Preface

The InCharge NetIQ AppManager Adapter is a one-way notification import adapter that transfers NetIQ events into the InCharge Service Assurance Manager (SAM) Adapter Platform.

Purpose

The *InCharge Service Assurance Manager User's Guide for NetIQ AppManager Adapter* provides an overview of the InCharge NetIQ AppManager Adapter, detailed configuration information, and procedures for its operation.

Intended Audience

This document is intended for network operations personnel and integrators who set up and maintain the InCharge NetIQ AppManager Adapter.

Document Organization

This document consists of the following sections:

1. INTRODUCTION	Provides an overview of the InCharge NetIQ AppManager Adapter
2. NETIQ ADAPTER DATA MAPPING AND SYNCHRONIZING	Details how the InCharge NetIQ Adapter maps data from the AppManager
3. CONFIGURING THE NETIQ ADAPTER	Details how to configure the InCharge NetIQ Adapter
4. INSTALLING AND USING THE NETIQ ADAPTER	Provides installation requirements for the InCharge NetIQ Adapter, and provides instructions for starting and stopping the adapter
A. RECONCILING EVENT MODELS	Provides information about how the InCharge NetIQ Adapter reconciles event data
B. GENERIC EVENT CONVERSION	Provides information about the conversion of event data
C. ADAPTER CONF FILE	Provides a sample adapter.conf file
D. EVENT PROCESSING CONF FILE	Provides a sample event.conf file

Table 1: Document Organization

Documentation Conventions

Several conventions may be used in this document as shown in Table 2.

CONVENTION	EXPLANATION
`sample code`	Indicates code fragments and examples in Courier font
keyword	Indicates commands, keywords, literals, and operators in bold
%	Indicates C shell prompt
#	Indicates C shell superuser prompt
<parameter>	Indicates a user-supplied value or a list of non-terminal items in angle brackets
[option]	Indicates optional terms in brackets

CONVENTION	EXPLANATION
/InCharge	Indicates directory path names in italics
yourDomain	Indicates a user-specific or user-supplied value in bold, italics
File > Open	Indicates a menu path in italics
▼▲	Indicates a command that is formatted so that it wraps over one or more lines. The command must be typed as one line.

Table 2: Documentation Conventions

Directory path names are shown with forward slashes (/). Users of the Windows operating systems should substitute back slashes (\) for forward slashes.

Also, if there are figures illustrating consoles in this document, they represent the consoles as they appear in Windows. Under UNIX, the consoles appear with slight differences. For example, in views that display items in a tree hierarchy such as the Topology Browser, a plus sign displays for Windows and an open circle displays for UNIX.

Finally, unless otherwise specified, the term InCharge Manager is used to refer to InCharge programs such as Domain Managers, Global Managers, and adapters.

InCharge Installation Directory

In this document, the term ***BASEDIR*** represents the location where InCharge software is installed.

- For UNIX, this location is: */opt/InCharge<n>/<productsuite>*.
- For Windows, this location is: *C:\InCharge<n>\<productsuite>*.

The *<n>* represents the InCharge software platform version number. The *<productsuite>* represents the InCharge product suite that the product is part of.

Table 3 defines the *<productsuite>* directory for each InCharge product.

PRODUCT SUITE	INCLUDES THESE PRODUCTS	DIRECTORY
InCharge IP Management Suite	• IP Availability Manager • IP Performance Manager • IP Discovery Manager • InCharge Adapter for HP OpenView NNM • InCharge Adapter for IBM/Tivoli NetView	/IP
InCharge Service Assurance Management Suite	• Service Assurance Manager • Global Console • Business Dashboard • Business Impact Manager • Report Manager • SAM Failover System • Notification Adapters • Adapter Platform • SQL Data Interface Adapter • SNMP Trap Adapter • Syslog Adapter • XML Adapter • InCharge Adapter for Remedy • InCharge Adapter for TIBCO Rendezvous • InCharge Adapter for Concord eHealth • InCharge Adapter for InfoVista • InCharge Adapter for NetIQ AppManager	/SAM
InCharge Application Management Suite	• Application Services Manager • Beacon for WebSphere • Application Connectivity Monitor	/APP
InCharge Security Infrastructure Management Suite	• Security Infrastructure Manager • Firewall Performance Manager • InCharge Adapter for Check Point/Nokia • InCharge Adapter for Cisco Security	/SIM
InCharge Software Development Kit	• Software Development Kit	/SDK

Table 3: Product Suite Directory for InCharge Products

For example, on UNIX operating systems, InCharge IP Availability Manager is, by default, installed to */opt/InCharge6/IP/smarts*. This location is referred to as ***BASEDIR****/smarts*.

Optionally, you can specify the root of ***BASEDIR*** to be something other than */opt/InCharge6* (on UNIX) or *C:\InCharge6* (on Windows), but you cannot change the *<productsuite>* location under the root directory.

For more information about the directory structure of InCharge software, refer to the *InCharge System Administration Guide*.

Additional Resources

In addition to this manual, SMARTS provides the following resources.

InCharge Commands

Descriptions of InCharge commands are available as HTML pages. The *index.html* file, which provides an index to the various commands, is located in the ***BASEDIR****/smarts/doc/html/usage* directory.

Documentation

Readers of this manual may find other SMARTS documentation (also available in the ***BASEDIR****/smarts/doc/pdf* directory) helpful.

InCharge Documentation

The following SMARTS documents are product independent and thus relevant to users of all InCharge products:

- *InCharge Release Notes*
- *InCharge Documentation Roadmap*
- *InCharge System Administration Guide*
- *InCharge ICIM Reference*
- *InCharge ASL Reference Guide*
- *InCharge Perl Reference Guide*

InCharge Service Assurance Manager Documentation

The following SMARTS documents are relevant to users of the InCharge Service Assurance Management product suite.

- *InCharge Service Assurance Management Suite Installation Guide*
- *An Introduction to InCharge Service Assurance Manager*
- *InCharge Operator's Guide*
- *InCharge Service Assurance Manager Configuration Guide*

- *InCharge Service Assurance Manager Business Dashboard Configuration Guide*
- *InCharge Service Assurance Manager User's Guide for Business Impact Manager*
- *InCharge Service Assurance Manager User's Guide for Report Manager*
- *InCharge Service Assurance Manager Failover System User's Guide*

The following SMARTS documents are relevant to InCharge Service Assurance Manager adapters.

- *InCharge Service Assurance Manager Notification Adapters User's Guide*
- *InCharge Service Assurance Manager SQL Data Interface Adapter User's Guide*
- *InCharge Service Assurance Manager Adapter Platform User's Guide*
- *InCharge XML Adapter User's Guide*
- *InCharge Service Assurance Manager User's Guide for Remedy Adapter*
- *InCharge Service Assurance Manager User's Guide for Concord eHealth Adapter*
- *InCharge Service Assurance Manager User's Guide for InfoVista Adapter*

InCharge Application Services Manager Documentation

The following SMARTS documents are relevant to users of InCharge Application Service Manager.

- *InCharge Application Management Suite Installation Guide*
- *InCharge Application Services Manager User's Guide*
- *InCharge Application Services Manager Discovery Guide*
- *InCharge Application Connectivity Monitor User's Guide*

Technical Support

SMARTS provides technical support by e-mail or phone during normal business hours (8:00 A.M.—6:00 P.M. U.S. Eastern and Greenwich Mean Time). In addition, SMARTS offers the InCharge Express self-service web tool. The web tool allows customers to access a personalized web page and view, modify, or create help/trouble/support tickets. To access the self-service web tool, point your browser to:

https://websupport.smarts.com/SelfService/smarts/en-us

U.S.A Technical Support

E-Mail: *support@smarts.com*
Phone: +1.914.798.8600

EMEA Technical Support

E-Mail: *support-emea@smarts.com*
Phone: +44 (0) 1753.878140

Asia-Pac Technical Support

E-Mail: *support-asiapac@smarts.com*

You may also contact SMARTS at:

	U.S.A World Headquarters	United Kingdom
Address	SMARTS 44 South Broadway White Plains, New York 10601 U.S.A	SMARTS Gainsborough House 17-23 High Street Slough Berkshire SL1 1DY United Kingdom
Phone	+1.914.948.6200	+44 (0)1753.878110
Fax	+1.914.948.6270	+44 (0)1753.878111

For sales inquiries, contact SMARTS Sales at:
sales@smarts.com

SMARTS is on the World Wide Web at:
http://www.smarts.com

1

Introduction

This chapter provides a brief overview of the InCharge NetIQ AppManager Adapter.

The NetIQ AppManager provides various server and application monitoring capabilities. Originally developed for the Windows platform, it has been expanded to monitor UNIX platforms. Windows, however, remains the product focus.

NetIQ AppManager is designed to periodically execute Knowledge Scripts on managed hosts and create and forward event messages when problems or noteworthy changes in status are detected. The event messages are stored as records in an SQL database on a central server. Client interfaces can access the stored messages and generate reports.

InCharge NetIQ AppManager Adapter

The InCharge NetIQ AppManager Adapter operates in conjunction with NetIQ AppManager and provides:

- The importation of AppManager events into InCharge from the SQL database as simple notifications unassociated with existing InCharge topological elements.
- The importation of AppManager events into InCharge from the SQL database as notifications against the supporting host or application object (ASM).

- A configurable means of extending the core adapter to provide extended notifications and/or topology creation/maintenance through the addition of ASL hook scripts.

Functional Overview

The NetIQ AppManager Adapter is a one-way notification import adapter that transfers NetIQ events into the InCharge Service Assurance Manager (SAM) Adapter Platform. The NetIQ AppManager is viewed as having the authoritative event view and it is the task of the NetIQ AppManager Adapter to ensure that view is replicated in SAM.

Beyond host inventory, NetIQ AppManager does not support meaningful topological data from an InCharge perspective. Although other topological information may be available, the only reliable and supported managed objects are the hosts on which AppManager agents run. However, since the SAM Adapter Platform commonly receives information from one or more InCharge Availability Managers (AM), the NetIQ Adapter considers the InCharge topological host view as authoritative. As configured by default, no host managed objects are imported from AppManager, and no events are imported that AppManager has derived from hosts unknown to InCharge. A configuration option allows the NetIQ Adapter to construct SAM Adapter Platform host objects based on information in AppManager Events, but this is intended only for installations where AM topology is not available.

The InCharge NetIQ Adapter handles imported events by expanding configured notification templates. Proper specification of the notification *ClassName*, *InstanceName*, and *EventName* allows the adapter to produce notifications which:

- Are Display Only: events are converted to SAM Adapter Platform notifications and are displayed in the Global Console; there is no reflection into the SAM Adapter Platform topology.
- Provide meaningful symptomatic information of SAM Adapter Platform objects for subsequent correlation by Application Services Manager (ASM).

More complex notification and topology processing is possible through the use of ASL hook scripts.

Architectural Overview

InCharge communicates directly with one or more NetIQ AppManager Adapters, each of which communicates with a single AppManager Repository (see Figure 1). The Repositories are implemented as SQL Server 2000 databases and provide access to event and other data through a suite of predefined stored procedures supplied by NetIQ. No asynchronous triggers are available and data changes of any type are derived by polling the database. Multiple adapters may be configured to connect to a single SAM Adapter Platform to import information from multiple AppManager repositories. As with other Smarts Adapters, notifications are related by the notification source to the generating adapter.

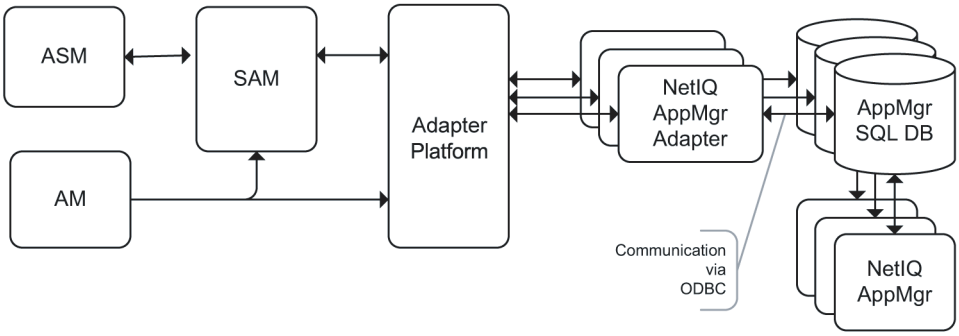

Figure 1: InCharge-AppManager Architecture

The NetIQ AppManager Adapter is implemented as a multi-threaded ASL adapter that performs the following:

- On startup, the *adapter.conf* and *event.conf* configuration files are processed.
- A dedicated thread is launched to monitor SAM Adapter Platform connectivity.
- On initial or re-connection to the SAM Adapter Platform, any pre-existing SAM Adapter Platform notification state is synchronized with that of AppManager by a comparison of state between the two servers.
- A single thread is launched to poll the NetIQ database to detect changes in event status and propagate those changes as InCharge Notifications. The thread terminates on either loss of the SAM Adapter Platform connection, or a SQL Server error. It restarts whenever the SAM Adapter Platform connection is re-established.

- The NetIQ AppManager Adapter process terminates when it loses the connection to the SQL Server.

The NetIQ AppManager Adapter supports a configuration file, *event.conf,* that determines how Knowledge Script events are to be translated into notifications using notification templates (see Figure 2) and adapter hook scripts. These custom hook scripts may additionally manipulate the InCharge topology.

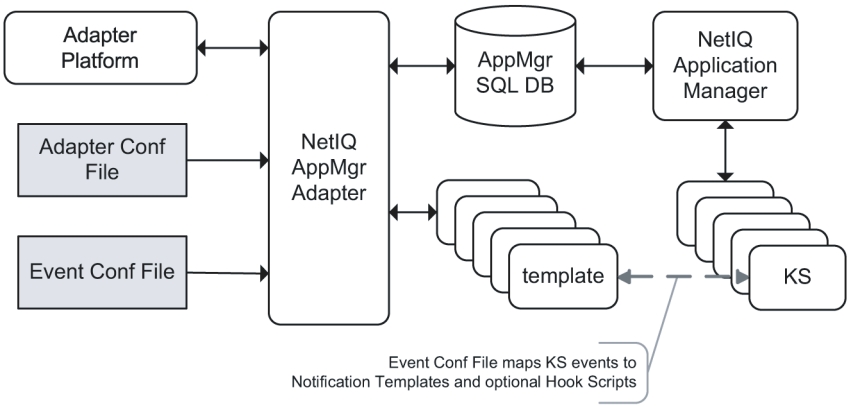

Figure 2: InCharge NetIQ Adapter and Notification Templates

The Adapter maps references to AppManager *Machine Names* into InCharge *Unitary Computer System* (UCS) names using SAM Adapter Platform name and IP address information supplied by InCharge AMs that discover hosts supporting AppManager agents.

2

NetIQ Adapter Data Mapping and Synchronizing

This chapter explains data mapping and synchronizing. It also disucsses importing event data.

Topology and Event Mapping

Topological information is of significance to the NetIQ AppManager Adapter in two areas:

- The identification and importation of externally known managed objects as topology
- The identification of the particular topological object against which an event condition is reported

NetIQ maintains an object table in the SQL database that identifies hosts and other managed objects by a unique number (Object ID); this is specific to a particular SQL database. The host is the foundation object which AppManager monitors and upon which applications run. Only host objects commonly have valid entries in the object table. In other words, a monitored web server application may or may not have an Object ID in the table, but the host on which it runs will always be represented in the table.

Although the NetIQ Adapter does not import or export topology, it must be able to relate hostnames between InCharge and AppManager. It accomplishes this by mapping AppManager *Machine Names* into InCharge *Unitary Computer System* (UCS) names using the SAM Adapter Platform Name and IP address information that is supplied by AMs when they discover hosts supporting AppManager agents.

The free-form nature of the AppManager event data generally precludes fixed automated analysis. Additionally, there is no reliable generic means of determining the application instance that caused the event. However, since AppManager associates hosts with specific Knowledge Scripts, it is possible to identify the supporting InCharge host object associated with a given event. Because the NetIQ Adapter's event handling is driven by user-defined Notification Templates, it can be configured to apply this information using different strategies as appropriate.

Host-Centered Status

The simplest configuration approach is to status the host object rather than the supported application objects. This can be accomplished even with the limited topological information available from AppManager. In this case, notifications are always created against the InCharge host representing the AppManager host on which the monitoring application runs. This solution can be applied to any Knowledge Script since all events from all Knowledge Scripts provide the event *Machine Name*. Explicit topology mapping between InCharge and AppManager internal representations is effectively avoided.

Application-Centered Status

Since InCharge Application Services Manager (ASM) can correlate symptomatic notifications of specific application objects, a more refined notification design that statuses individual applications may be desirable. In this case, the lack of generically available managed object instance identification is problematic. Although there is no general solution, either of two strategies may be applied to cover the majority of cases:

- Host-based instance identification for singleton applications
- Custom parsing of event messages to extract instance identification

Cases not addressable by these strategies are restricted to generate only notifications with no topological impact.

Using Singleton Applications to Simplify Instance Identification

Many applications may be termed Singleton Applications. Because of operational policy or application restrictions, only one singleton application of a given type may run on a given host; for example, an Exchange mail server. This affords an opportunity to differentiate Singleton Application instances only by the name of the supporting host. Templates are arranged to generate notifications against appropriate application classes with instance names differing only by hostname component, such as *APP-ExchangeServer/<hostname>*. Explicit topology mapping between InCharge and AppManager internal representations is effectively avoided.

Custom Parsing Using Hook Scripts

Custom prehook scripts may be associated with particular Knowledge Scripts or Notification Templates. These scripts may parse the event message text to identify the particular application instance causing the event. The resulting instance name can be employed as required in the Notification Template. The means of establishing correspondence between AppManager and InCharge application objects is completely ad hoc. This solution requires more effort than other strategies and is more fragile because of the ability of customers to modify the event message text. However, it can provide an effective means of supporting non-singleton applications.

Events

The following sections discuss AppManager and Service Assurance Manager Adapter Platform event attributes.

AppManager Event Attributes

AppManager events are produced by Knowledge Scripts running on agents located on monitored Hosts. These events are ultimately stored in several tables of the repository. Of these, the Event Table stores basic event information as shown below. The rows marked with an * show column data available for Notification Template and hook script use.

NAME	TYPE	DESCRIPTION
EventID *	integer	Identity field; unique event identifier
ParentEventID *	integer, Null	Null if parent event, else EventID of parent event
JobID *	integer, Null	Job ID of job that created the event
MachineObjID	integer	Machine object ID of machine in Object Table
MachineName *	varChar(128)	Full machine name as it appears in TreeView
Status *	integer	Openstatus &0xFF=0 Acknowledgedstatus &0xFF=1 Closedstatus &0xFF=2 Deletedstatus &0x80000000=0x80000000
Last Occurrence	integer	UTC of last event occurrence
ObjID	integer, Null	DB Table index of affected object. If <NULL> then the host is assumed to be the target object. May refer to a parent object of the actual affected object depending on script. Actually used only to blink item in TreeView
KPName *	varChar(40), Null	Name of Knowledge Script. The prefix is separated from the base script name by "_" and indicates the Application Module to which the KS belongs. This prefix may be redefined by the user by copying and renaming the Application module.
Severity *	integer	Severity from 1-40. Generally mapped into displayable severities: Severe= 01-10 Warning= 11-20 Informational= 21-30 Diagnostic= 31-40
EventMsg *	varChar(255), Null	Free form message describing Event
Occurence	integer	Number of times event occurred
IconID	integer, Null	ID of Icon used by Console
ModificationTime	integer	UTC of last event modification
ChildComment	integer, Null	Comment added by user
RootSrvObjID	integer	Server Group of the Machine
Value	float, Null	Reserved
QDW_Flag	tinyint, Null	Reserved - used by Analysis Center

Table 4: AppManager Event Table Description

Other pertinent columns, *Agent Long Message*, *Agent Short Message*, and *First Occur Time* are held in the Event Detail Table. The NetIQ Adapter obtains repository information by way of a fixed suite of NetIQ-supplied stored procedures making the exact database schema irrelevant.

NAME	TYPE	DESCRIPTION
EventID *	integer	Identity Field; unique event identifier
DropKpObjID	integer, Null	
FirstOccurTime *	integer	UTC of first event notification
UserStatusID	integer	
TypeObjName	varChar(169), Null	
Comment	varChar(255), Null	
AgentMsgshort *	varChar(255), Null	Text string provided by Knowledge Script
AgentMsglong *	text, Null	Text string provided by Knowledge Script
AckTime	integer, Null	UTC when event is ACKED
AckUserID	varbinary(85), Null	User ID of event ACKer
AJobID	integer, Null	
ActionIdxString	varChar(255), Null	
ReservedInteger1	integer	Reserved
ReservedInteger1	integer	Reserved
ReservedVarchar1	varChar(255), Null	Reserved

Table 5: AppManager Event Detail Table Description

Service Assurance Adapter Platform Notification Attributes

AppManager event attributes are mapped to Service Assurance Adapter Platform notification attributes according to Notification Templates defined in the *event.conf* file. As with the Trap and BMC Patrol Adapters, both simple and aggregated notifications are supported. Several sample templates are provided, two of which are Notify-Only and Host (see *Event Processing Conf File* on page 51). The former creates notifications that appear in the SAM Global Console, while the latter causes statusing of the supporting host object. As an example, Table 6 shows the notification property values provided by these two default Templates.

PROPERTY NAME	DESCRIPTION	FLAGS	NETIQ VALUE
Acknowledged	Indicates if this event has been acknowledged	Read-only Computed (with expression) BOOLEAN	Derive from Event[Status]
Active	Indicates if this event currently active	Read-only Computed (with expression) BOOLEAN	Derive from Event[Status]
Category	Category of this event. The event category represents a broad categorization of the event, e.g. availability vs. performance	Stored STRING	Performance
Certainty	The certainty of this event	Stored FLOAT Range: 0 .. 100	100
ClassDisplayName	Display name for the event class	Stored STRING	same as ClassName
ClassName	Class name of the object where this event occurred. This attribute along with InstanceName and EventName uniquely identify this event.	Stored STRING	NOTIFY-ONLY: NIQAM Host: UnitaryComputerSystem
DisplayName	The string shown in the GUI when this object is displayed	Stored STRING	Host: Event[EventMsg]
ElementClassName	The class name of the topology element associated with the event in the repository where this event resides. This may or may not have the same value as ClassName.	Read-only STRING	Host
ElementName	The name of the topology element associated with the event in the repository where this event resides. This may or may not have the same value as InstanceName. The string is empty if there is no related element.	Read-only STRING	Event[MachineName]
EventDisplayName	Display name for the event Name	Stored STRING	same as EventName

PROPERTY NAME	DESCRIPTION	FLAGS	NETIQ VALUE
EventName	Name of the event. This attribute along with ClassName and InstanceName uniquely identify this event.	Stored STRING	NOTIFY-ONLY: ALERT Host: DegradedSymptom
EventText	The textual represenation of the event	Stored STRING	Event[EventMsg]
EventType	Indicates the nature of the event. A MOMENTARY event has no duration. An authentication failure is a good example. A DURABLE event has a period during which the event is active and after which the event is no longer active. An example of a durable event is a link failure. {DURABLE | MOMENTARY}	Enumerated Stored STRING	DURABLE
InstanceDisplayName	Display name for the event instance	Stored STRING	same as InstanceName
InstanceName	Instance name of the object where this event occurred. This attribute along with ClassName and EventName uniquely identify this event.	Stored STRING	Hostname mapped from Event[MachineName]
IsRoot	Is this a root notification?	Read-only BOOLEAN	TRUE
Name	Name of object	Read-only Stored STRING Required	Composite string of: <notificationtag>_<EventID> (notificationTag from adapter.conf, typically: NETIQ_<odbcDns>)
Severity	An enumerated value that describes the severity of the event from the notifier's point of view: 1 - Critical is used to indicate action is needed NOW and the scope is broad, e.g. an outage to a critical resource. 2 - Major is used to indicate action is needed NOW. 3 - Minor should be used to indicate action is needed, but the situation is not serious at this time. 4 - Unknown indicates that the element is unreachable, disconnected or in an otherwise unknown state. 5 - Normal is used when an event is purely informational.	Stored UNSIGNED Range:1-5	Map from Event[Status] according to configured mapping rules

PROPERTY NAME	DESCRIPTION	FLAGS	NETIQ VALUE
UserDefined1	User defined field 1		Event[MachineName]
UserDefined2	User defined field 2	Stored STRING	Event[Status] = (1-40)
UserDefined3	User defined field 3	Stored STRING	Event[EventID]
UserDefined4	User defined field 4	Stored STRING	Event[ParentEventID]
UserDefined5	User defined field 5	Stored STRING	Event[JobID]
UserDefined6	User defined field 6	Stored STRING	Event[AgentMsg Short] or Event[AgentMsg Long]

Table 6: Notification Properties for Notify-Only and Host Sample Templates

Event Severity

The NetIQ Adapter provides two means of setting the severity of imported AppManager events:

- Mapped AppManager severity levels
- Explicit severity specification independent of AppManager severity

AppManager Event Severity Mapping

NetIQ event severity is represented as a continuum of values from 1-40. AppManager is configured to partition these into four groups: severe, warning, informational, and diagnostic. These groupings, however, are largely for display purposes; severity is maintained internally as the 1-40 integer. The NetIQ Adapter provides a configurable mapping to transform the AppManager integer values into the standard SAM notification severities (see *Adapter Conf File* on page 41). The InCharge Unknown notification state has no counterpart in AppManager and cannot be mapped from imported data.

AppMgr Severity	Default AppMgr Severity Values	Default SAM Severity Values	SAM Severity	SAM Description
Severe	1-10	1-5	Critical	action is needed NOW and the scope is broad, for example, an outage to a critical resource
		6-10	Major	action is needed NOW
Warning	11-20	11-20	Minor	action is needed, but the situation is not serious at this time
		N/A	Unknown	the element is unreachable, disconnected or in an otherwise unknown state
Informational	21-30	21-40	Normal	the event is purely informational
Diagnostic	31-40			

Table 7: AppManager to SAM Severity Mapping

Explicit Severity Specification

The reported severities for third party applications may not be appropriate from the SAM global perspective. Notification Templates provide a means of specifying fixed SAM notification severity in lieu of the mapped AppManager value. This is typically used to demote external severities into an InCharge context.

Event State Mapping

In general, it is not possible to completely reconcile the AppManager and InCharge event models. The NetIQ Adapter typically uses multiple AppManager events to determine the lifetime of imported notifications (see *Reconciling Event Models* on page 31). The active state of imported notifications accurately reflects the state of the real world monitored condition, however, reverse mapping of the state of an imported notification to the state of a single AppManager event is not possible. Similarly, there is no accurate correspondence between the InCharge and AppManager acknowledgement models.

Hostname Mapping

The NetIQ Adapter uses a built-in SAM Adapter Platform algorithm to transform the AppManager *Machine Name* (simple basename) to the InCharge representation (basename, DNS name, or IP address). In order to guarantee that the required IP and naming information is available in the Adapter Platform hosting the NetIQ Adapter, that Adapter Platform must also host the AM used to discover the hosts supporting AppManager agents. Because AppManager uses true hostnames and the AM and AppManager agents are supported by the same DNS, correspondence between the AppManager *Machine Name* and AM discovered hosts can be achieved using available SAM Adapter Platform data. Both the *Machine Name* and mapped InCharge *UCS instance name* are placed in a table of Computed Adapter Values and are available for use in notification templates as desired.

Synchronizing Data

Explicit synchronization occurs whenever a connection is established with the Service Assurance Manager Adapter Platform and with the AppManager repository. Once synchronized, normal processing proceeds in such a fashion that synchronization is maintained and further explicit synchronization operations are not required. Each such sequence of explicit synchronization and subsequent ongoing event processing is termed an *epoch*. SQL server error or SAM Adapter Platform connection loss terminates the current epoch. SQL server connection loss terminates the NetIQ Adapter process.

Event Synchronization

Event synchronization occurs at the beginning of every epoch.

The purpose of synchronization is three-fold:

- Introduce into the SAM Adapter Platform new notifications that do not previously exist
- Correct the status of notifications which do pre-exist
- Delete pre-existing notifications which are no longer relevant

Synchronization uses a variation of the standard discovery strategy.

- The SAM Adapter Platform is queried for all pre-existing notifications originally produced by that particular Adapter and a table of potentially stale notifications is created.
- PresyncHookScript is invoked, if defined
- The AppManager repository is queried to obtain all non-deleted events and the SAM Adapter Platform is updated as required to reflect these new events. Each corresponding entry in the stale notification table is deleted as events are imported. Events are otherwise processed normally using the mechanism outlined in *Generic Event Conversion* on page 39.
- PostsyncHookScript is invoked, if defined
- At the conclusion of import, all SAM Adapter Platform notifications corresponding to the remaining entries in the stale notification table are deleted.

Importing Data

Data import begins after notification synchronization has been completed and is abandoned on SQL server error or loss of the SAM Adapter Platform connection. The data import process makes no attempt to recover interrupted or errored server accesses (either the SAM Adapter Platform or SQL Server) beyond low-level I/O or ODBC retries; a new epoch must begin and resynchronization is required before import processing can resume anew.

Importing Topology

The AppManager host inventory is well defined but is normally not imported to InCharge. However, the NetIQ Adapter provides a configuration option *acceptEventsFromUnknownHosts* which allows the adapter to incrementally add to the SAM Adapter Platform topology specific host objects referenced in events by *Machine Name*. This operational mode is designed solely for use in environments where AM discovery is not possible; normally the core NetIQ Adapter imports no topology.

Custom post-hook scripts may also create topology (for example, Application objects) based on information parsed from specific event messages. Custom pre- and post-synchronization hook scripts may be configured to assist in maintenance of such topology.

Importing Event Data

The NetIQ Adapter accesses the SQL database by way of stored procedures supplied by NetIQ. No asynchronous triggers are available to determine changes to the event tables, so the database is periodically polled to detect new or changed events.

Each event is associated in the SQL database with the particular Knowledge Script that generated it. AppManager groups Knowledge Scripts into categories. By convention the application category name is presented as the prefix of each Knowledge Script separated by an underscore:

```
<applicationCategory>_<knowledgeScriptBasename>
```

The *event.conf* configuration file guides event processing by associating the name of the generating Knowledge Script with a particular notification template. Each notification template generates a single notification and an optional aggregating parent notification. Events generated by Knowledge Scripts not defined in the configuration file are ignored. The Event processing strategy is outlined in greater detail in *Generic Event Conversion* on page 39.

Exporting Data

Events are not exported. However, the NetIQ Adapter may be configured to automatically CLOSE selected events in AppManager after they have been processed by the adapter. This is not strictly an export operation but does reflect an imposed change of state. These operations may be selected to minimize AppManager housecleaning operations and to minimize renotification by NOTIFY_WO_CLEAR events during epoch synchronization

3

Configuring the NetIQ Adapter

The InCharge NetIQ AppManager Adapter needs to be configured before it can operate in conjunction with the Service Assurance Manager Adapter Platform and the SQL server.

This chapter discusses the configuration requirements for the NetIQ AppManager Adapter.

NetIQ Adapter Configuration Requirements

The NetIQ Adapter uses two configuration files and a number of supporting hook scripts. The configuration files are processed and validated on startup prior to connection to the Service Assurance Manager Adapter Platform or the SQL Server.

Note: The NetIQ Adapter must be stopped and restarted to process configuration changes.

In addition to the configuration files, the NetIQ Adapter command-line utility, *ic-niqam-adapter.cmd*, must be configured before it can be used.

Table 8 lists the configuration files and scripts that need to be configured.

FILES REQUIRING CONFIGURATION	DESCRIPTION
BASEDIR/local/script/niqam/ic-niqam-adapter.cmd	Adapter Service Installation and start/stop utility.
BASEDIR/local/conf/ics/niqam/adapter.conf	Primary adapter configuration file.
BASEDIR/local/conf/ics/niqam/event.conf	Configuration file defining processing of each imported event. Describes Knowledge Script to Notification Template mapping.
BASEDIR/local/rules/niqam/hook/*.asl	Adapter hook scripts invoked by Notification Templates

Table 8: NetIQ Adapter Configuration Files and Scripts

The adapter.conf File

This file provides application-level configuration and is in .sh format that allows only assignments of the form *varName=value*. Standard .sh comment format is allowed. See *Adapter Conf File* on page 41 for a description and example of the file. The available options are accompanied by a description which includes the default value that is used if the associated parameter is undefined (commented out).

The event.conf File

This file defines notification templates and associates those templates with particular Knowledge Scripts. See *Event Processing Conf File* on page 51 for a description and example of the file. Notification templates may inherit from other templates allowing common configuration schemas to be isolated in one or more base templates. Derived templates may override properties of the base templates.

Each template supports one of four notification modes:

- AUTO
- NOTIFY
- CLEAR
- NOTIFY_WO_CLEAR

NOTIFY and CLEAR present the familiar two basic notification operations. NOTIFY_WO_CLEAR is similar to NOTIFY but directs the adapter not to expect a matching event to CLEAR the generated notification. AUTO is the most prevalent mode and directs the adapter to NOTIFY on the initial event and CLEAR on the associated State Change event (See *Reconciling Event Models* on page 31).

The NetIQ Adapter includes a sample *event.conf* file that defines two basic sample templates to generate Notify-Only and Host Notifications. Notifications generated with the former appear in the SAM Global Console, while the latter causes statusing of the supporting SAM Adapter Platform host. Several other sample templates are included to illustrate other notification situations including ClearOnAcknowledge Expiration and Error Event handling.

The latter portion of the *event.conf* file lists 1800 Knowledge Scripts with sample template associations. These are provided only as a starting point for configuration. In general, the supplied *event.conf* file will require customization for proper operation with a NetIQ AppManager installation.

SQL Server Configuration Requirements

The following sections describe the configuration requirements for the SQL server.

SQL Server Access Method

The NetIQ Adapter uses ODBC to access the database of the NetIQ AppManager. An appropriate ODBC data source name (DSN) must be configured in the *adapter.conf* file to direct the NetIQ Adapter to the proper database. Separate *adapter.conf* files and separate adapter instances are required to access multiple AppManager databases.

SQL Server Authorization And Access Privileges

The NetIQ Adapter interface consists of the execution of SQL stored procedures to AppManager. SQL Server access must be granted to the adapter using the preferred form of Authorization by the particular installation. Configuration of an appropriate *username* and *password* may be required in the *adapter.conf* file.

The basic operations of the adapter require Read access to execute the stored procedures and export event data. However, the configurable adapter options to automatically CLOSE selected AppManager events require Write access as well.

NetIQ AppManager Configuration Requirements

For proper InCharge management of NetIQ events, individual Knowledge Scripts must be configured to generate a new event when the original event condition no longer exists. This is the State Change event. As shown below, the option to automatically close the original event may also be selected to simplify housekeeping tasks in AppManager. This second configuration is recommended but not necessary.

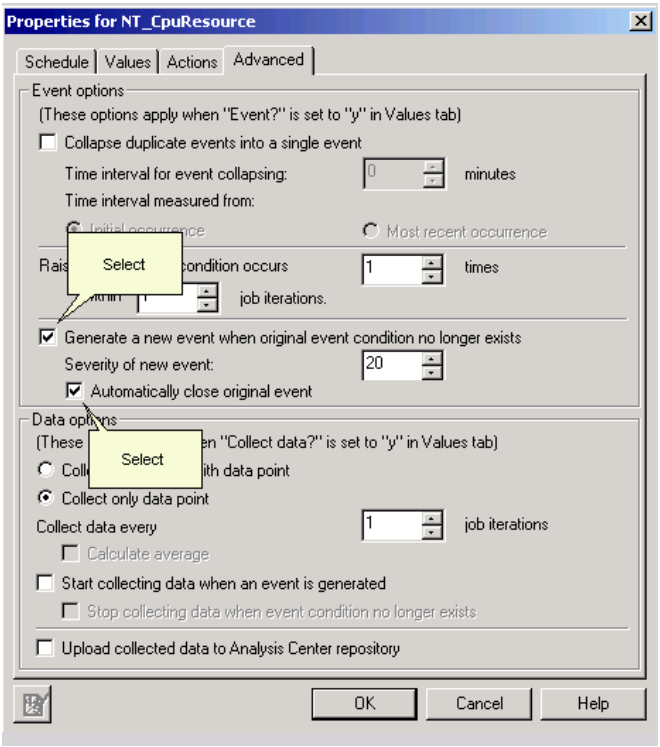

Figure 3: AppManager Properties

Because the NetIQ Adapter obtains event information by polling the AppManager database, care must be exercised in configuring other available AppManager options to ensure that no event (in particular, the State Change event) is automatically and immediately closed or archived by AppManager before it can be polled by the Adapter.

ASL Hook Script Requirements

Adapter Scripting Language (ASL) hook scripts must be implemented to customize event processing beyond that afforded by the basic adapter or the supporting configuration files. These hook scripts extend functionality without altering the core of the adapter.

The NetIQ Adapter supports the following hook scripts:

- pre- and post-synchronization scripts
- prehook scripts
- posthook scripts

All hook scripts are executed by way of GA_PersistentDrivers and receive all -D variables passed to the adapter at startup. This allows the user to define additional, non-conflicting -D variables which may be supplied when the adapter is invoked, and which will then be passed to the hook scripts without requiring alteration of the adapter.

Pre-/Post-Synchronization Hook Scripts

Two optional hook scripts may be defined in the adapter configuration file:

- presyncHookScript
- postsyncHookScript

These scripts are invoked as indicated in the previous section to support special-purpose event and topology synchronization processing required by pre- and post-hook scripts and not provided by the basic adapter.

Prehook Scripts

The standard event processing of the NetIQ Adapter may be extended by implementing prehook scripts. The hook scripts have access via global tables to event and configuration data (from the *adapter.conf* file), as well as precomputed values developed by the adapter (see *Generic Event Conversion* on page 39). A prehook script typically parses/processes Knowledge Script-specific raw event data, and posts results in a global table of Hook Script Results where they become candidates for substitution into the Notification Template.

Prehook script processing will typically be required for each Knowledge Script to:

- Select specific event messages to convert into notifications
- Select specific event error messages to import as non-clearing events

As previously discussed, the Knowledge Script name is used to select the specific Notification Template to expand. However, Knowledge Scripts may generate more than one type of event message, and may also generate error events which must be handled differently than the expected events. This is particularly problematic when the expected event should be notified in AUTO mode so that a State Change event is expected to CLEAR it, but an Error Event is received instead for which no corresponding Event will occur to affect the CLEAR. Prehook scripts address these situations by performing glob-style matching of event messages to cause different adapter processing for different event messages.

Prehook scripts can control the operation of the adapter to accept or reject notification import, or to force the use of an alternate Notification Template other than that configured in the *event.conf* file. Prehook scripts may be written to accept parameters. This can minimize the number of required hook scripts by allowing implementation of more generic scripts which accept pattern and control information via parameters as defined in the *event.conf* file. Several samples of such parameterized prehook scripts are provided along with invocation examples in the sample file (see *Event Processing Conf File* on page 51). The samples also show the available global data and global control variables, and provide several common utility functions useful for debugging.

Posthook Scripts

The standard event processing of the NetIQ Adapter may be extended by implementing posthook scripts. The hook scripts have access to the same data as prehook scripts as well as the template-expanded Proposed Notification (see *Generic Event Conversion* on page 39). A posthook script typically uses the Proposed Notification to guide manipulation or creation of topology in the host SAM Adapter Platform server. Using the Proposed Notification as a source of driving parameters may allow some degree of generic processing, reducing the number of required unique posthook scripts.

Epoch Synchronization

Each hook script has a unique supporting GA_PersistentDriver which is reset once at the beginning of a new epoch. The associated thread may access a Boolean global IS_SYNC_PHASE, which, if TRUE, indicates that the adapter is in 'epoch synchronization mode'. In this mode, the hook script is invoked to process pre-existing events in AppManager in much the same manner it processes new or updated events in normal operation. Interpretation of the special mode flag is hook script-specific but may typically signal additional manipulation of tables initialized by a pre-synchronization hook script that is to drive housekeeping functions by a post-synchronization hook script.

4

Installing and Using the NetIQ Adapter

This chapter includes the following sections:

- Installation requirements
- Deployment considerations
- Using the NetIQ Adapter

See the *InCharge Service Assurance Management Suite Installation Guide* for detailed information about installing the NetIQ AppManager Adapter.

Installation Requirements

The following lists the hardware and software requirements for the NetIQ AppManager Adapter.

Supported Platforms

The InCharge NetIQ AppManager Adapter is supported on:

- Windows 2000, Server and Advanced Server, with SP4
- Windows 2003

Hardware Requirements

The InCharge NetIQ AppManager Adapter requires:

- 20MB Minimum memory (RAM)
- 1MB Minimum application disk space

Required InCharge Software

The following InCharge components are required for the NetIQ AppManger Adapter:

- InCharge Service Assurance Manager 6.2
- InCharge Service Assurance Manager Adapter Platform 6.2
- Global Console 6.2
- InCharge Availability Manager 6.2
- InCharge Application Services Manager 1.1

Required Third-Party Software

The following third-party software components are required for the NetIQ AppManager Adapter:

- NetIQ AppManager 5.0.1
- MS SQL Server 2000

The NetIQ Adapter also requires access to an installed ODBC driver.

Deployment Considerations

The following considerations apply when deploying the NetIQ AppManager Adapter.

Co-residency Requirements

The NetIQ AppManager Adapter does not need to be co-resident with any particular NetIQ AppManager or InCharge product component.

Network and Domain Considerations

The Availability Manager responsible for monitoring hosts supporting AppManager agents must feed the same SAM Adapter Platform as does the NetIQ AppManager Adapter.

ODBC DSN Configuration

The Windows host on which the NetIQ Adapter is deployed must make available an ODBC DSN referencing the target NetIQ SQL database. This name and the configured Username and Password must be configured into ***BASEDIR****/smarts/local/conf/ics/niqam/adapter.conf*. The configured User must minimally have Read Access to the NetIQ SQL database. The User must also be granted Write Access if Adapter options are selected to close AppManager events.

The NetIQ Adapter Software

The NetIQ Adapter should be installed according to the specific documentation packaged with the adapter software. This is either a README file located in Point Patches, or an InstallShield selection in the case of normal General Access CD Releases. The former installs the software under ***BASEDIR****/smarts/local,* while the latter places the product directly under ***BASEDIR****/smarts/*. Relevant Adapter components are located in subdirectories named *niqam* (NetIQ AppManager).

Only a single software installation is necessary to run multiple instances of the NetIQ Adapter.

Modifying the NetIQ Adapter Files

Use the sm_edit utility to modify the NetIQ Adapter files. See *InCharge System Administration Guide* for additional information about modifying InCharge files.

- Copy the directory ***BASEDIR****/smarts/conf/niqam* and its contents as ***BASEDIR****/smarts/local/conf/niqam*
- Modify the following files to reflect your environment.
  - ic-niqam-adapter
  - ic-niqam-adapter.cmd

Special Considerations to Support Multiple AppManagers

Each supported AppManager SQL database will require a separate instance of the NetIQ Adapter. Each Adapter will require:

- A unique InCharge Domain Name
- Unique customized command scripts
- A unique customized *adapter.conf* file.

The *event.conf* file will typically be shared by all NetIQ Adapters, although separate versions may be configured if desired. To simplify administration, SMARTS recommends using some common naming element in the Domain Name, command script, and *adapter.conf* filename. For example, to support an AppManager on both the East and West coasts, the following choices might be appropriate:

DEFAULT	APPMANAGER EAST	APPMANAGER WEST
Domain Name	NETIQ_EAST	NETIQ_WEST
ic-niqam-adapter.cmd	smarts/local/script/ic-niqam-east.cmd	smarts/local/script/ic-niqam-west.cmd
ic-niqam-adapter	smarts/local/script/ic-niqam-east	smarts/local/script/ic-niqam-west
adapter.conf	smarts/local/conf/adapter-east.conf	smarts/local/conf/adapter-west.conf

Table 9: Multiple Files for a Multiple AppManager Installation

Using the NetIQ AppManager Adapter

The NetIQ Adapter runs as a server on sm_adapter, and will accept an optional parameter that may be used to specify an alternate configuration filename. The main adapter script is ***BASEDIR****/smarts/rules/ics/niqam/main.asl* and is minimally invoked as:

```
sm_adapter \
        --name=NETIQ_AM \
      [ -D "confFilename=adapter.conf" \]
        -Msm_system \
        -Msm_sdi\
       niqam\main.asl
```

Command Line Utility

The utility ***BASEDIR****/local/script/ic-niqam-adapter.cmd* can be used to install or remove the NetIQ Adapter as a Windows Service, or to issue commands to manually start and stop that service. Open a command window in ***BASEDIR****/local/script/* and run the following command to see the available options:

```
ic-niqam-adapter.cmd --help
```

During installation the leading portion of the utility must be manually edited to define the available options as defined in the associated comments.

As the -help option reveals, the utility offers several options to install and remove the adapter as a service. For example, the following command installs the adapter as a service that starts automatically at Windows startup:

```
ic-niqam-adapter.cmd install
```

The path to the InCharge NetIQ Adapter Service in the Windows Registry is as follows:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<SERVICE-NAME>

Starting and Stopping the Adapter

Once installed as a service, the NetIQ Adapter may be manually controlled using the normal Windows GUI for that purpose, or by way of the *ic-niqam-adapter.cmd* utility using options as shown below:

```
ic-niqam-adapter.cmd start
ic-niqam-adapter.cmd stop
ic-niqam-adapter.cmd bounce
```

Managing Hostname Mapping

By default, the NetIQ Adapter rejects events associated with hosts for which there are no corresponding host objects in the SAM Adapter Platform topology. Rejected hostnames can be configured to appear in the adapter log as warnings. Whenever a new host is managed by NetIQ, some convenient Knowledge Script should be run (for example, NT_ProcessUp) to create an event. The SAM Global Console should then be inspected to verify import of the event as a notification. If the notification is missing, the adapter log should be searched to locate a rejected hostname mapping.

A

Reconciling Event Models

This appendix discusses the reconciling of the InCharge and AppManager event models.

InCharge Notification Model and AppManager Events

The InCharge Notification model is outlined in Figure 4; it uses a simplified three state representation. The notification is a direct representation of the monitored condition, transitioning between active and inactive as a direct mirror of the real world condition. Acknowledgement simply indicates that the notification is being handled and has no effect on the primary state.

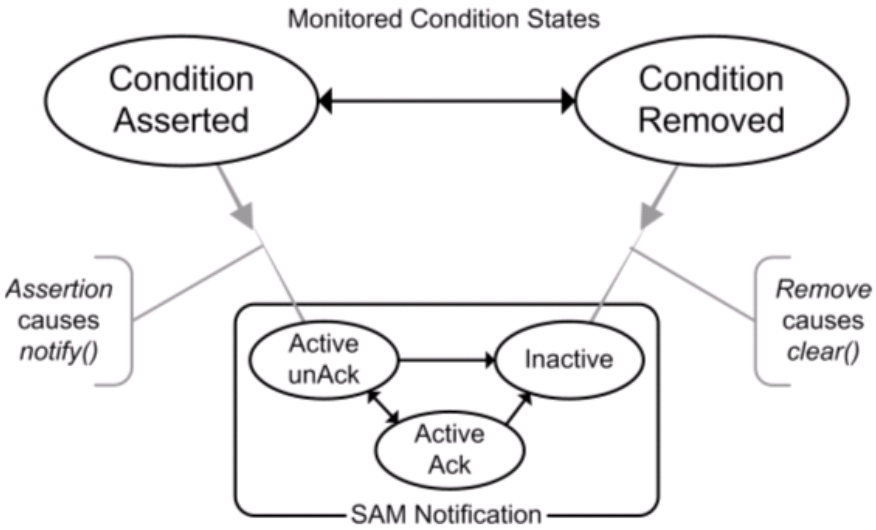

Figure 4: InCharge Notification Model

NetIQ AppManager events follow the transition table of Figure 5. Transitions between states may be manual or automatic. The transition to an ACKed state is a manual operation. However, unlike the InCharge Acknowledgement concept, the AppManager model interprets Acknowledgment as an indication that the system should do no further status updates to the event since it is being resolved by an operator. This includes preventing AppManager AutoClear of the event should the original monitored condition be removed; ACKed events must therefore be manually cleared.

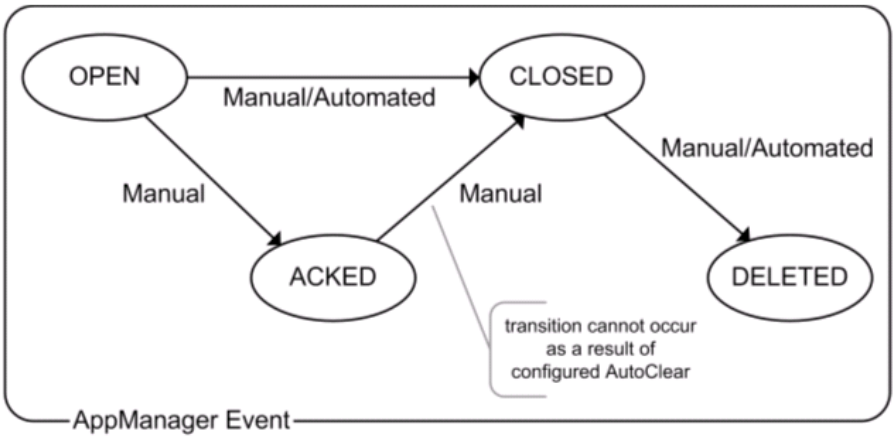

Figure 5: AppManager Event Transitions

NetIQ AppManager Event Model

Unlike InCharge notifications, AppManager events do not directly represent the state of the monitored condition; rather, they are messages generated by monitoring agents as a result of polling and have been filtered or processed by the central manager. As illustrated in Figure 6, there may potentially be many simultaneous events related to a single monitored condition, both for the ASSERT and REMOVE states. Each of these events has its own suite of states that generally are unrelated to one another. From this perspective, AppManager events resemble trouble tickets, each ticket being a report of the state at a point in time, and each possessing its own ticket state which is largely unrelated to the actual state of the Monitored Condition.

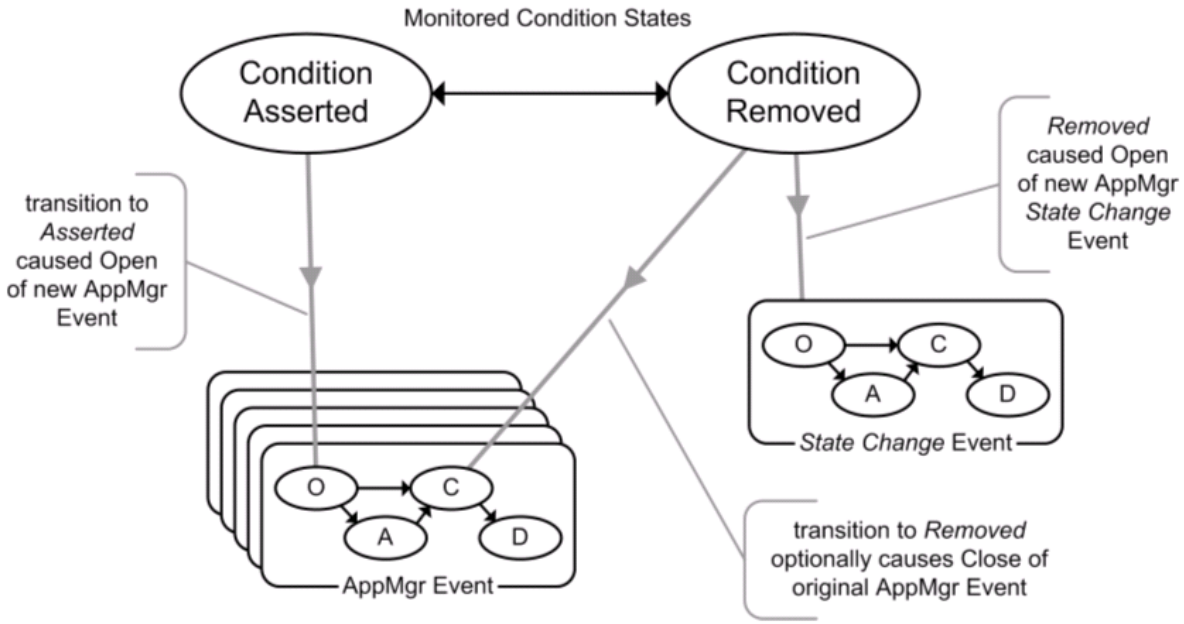

Figure 6: AppManager Multiple Events

Reconciling the Models

Although it is not possible to completely reconcile these two divergent event models, it is possible to correctly derive a valid InCharge notification that accurately reflects the state of the Monitored Condition. Figure 7 illustrates a possible technique which relies on expanding the (Adapter) view to encompass multiple events while ignoring the CLOSE state of all events.

For monitored conditions of interest, AppManager is configured to automatically generate a Status Change event when the Asserted Condition is removed. As shown in the illustration, notify() and clear() invocations are derived from the OPEN of different events which exactly bracket the Assertion period.

The required status change configuration setting may be set as a global default but is required only on events forwarded to InCharge. For completely automated (headless) AppManager operation, it remains but to automatically clear all involved events at the earliest opportunity. This may be accomplished by configuring AutoClear of the Assertion events and arranging timer-based aging of the Removal events. Again, these AppManager settings may be arranged as global defaults to simplify administration.

This approach gives a simplified, one-way data flow with AppManager as the Authoritative event source, allows headless AppManager operation, and provides complete management of derived notifications from the SAM Global Console using existing InCharge operational models. The AppManager console is used to configure and administer the monitoring agents and may continue to be used to handle events not forwarded to InCharge.

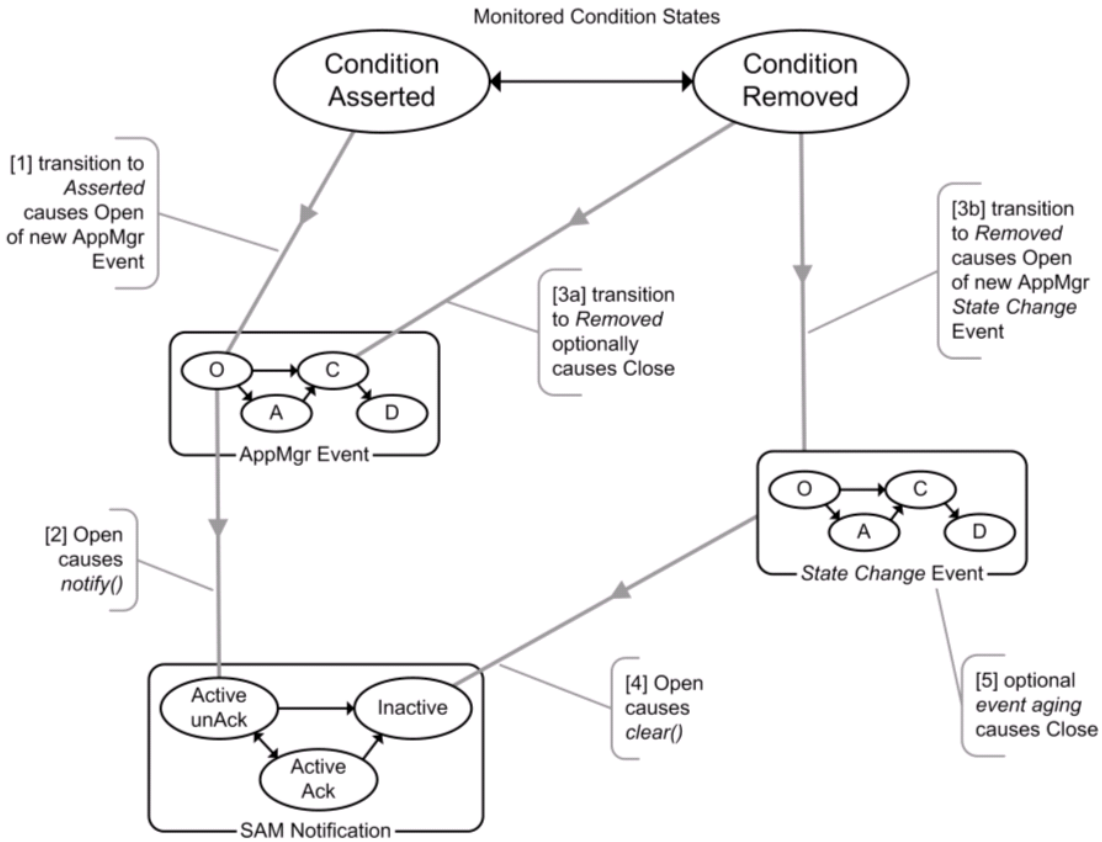

Figure 7: Deriving InCharge Notifications from AppManager Events

The availability of a persistent event state in the SQL database and the polled nature of AppManager events make it possible to accurately derive the Monitored Condition state during synchronization. Fundamental problems of correctly maintaining the state using deltas can be overcome by adopting an aggressive policy that resorts to the guaranteed accuracy of complete resynchronization whenever a connection loss or SQL errors occur.

Special Situations

The following sections describe several special situations.

Separate Knowledge Scripts Generating NOTIFY/CLEAR

There may be situations where the notify() and clear() operations may be generated by different Knowledge scripts (e.g. NT_ProcessUp and NT_ProcessDown). This is generally inadvisable since message timing from different asynchronous periodic pollers can lead to erroneous states. That said, a simple translation of such event pairs into notifications is possible. As in the previous discussion, the events must be configured to generate a 'State Change' event. The OPEN Event of one Knowledge Script is mapped to a NOTIFY, and the OPEN Event of another Knowledge Script is mapped to a CLEAR using directives available in the *event.conf* file. The NetIQ Adapter discards the associated State Change events.

The State Change configuration requirement ensures that the relative timing of condition assertion and removal can be accurately reproduced by ordering Events by ID. This is necessary to derive the proper notification state during the synchronization phase.

Error Events and other NOTIFY without CLEAR Events

There are other classes of events in AppManager which cannot be configured to produce an accompanying State Change event. Of particular importance are single Error Events which may be generated instead of the expected pair of original/state change events discussed above. This may occur because a Knowledge Script detects and publishes an error condition. As with many traps, these events must generate a notification with an Expiration Time and possibly a Clear On Acknowledge capability.

Synchronization of such events is problematic; issues of re-notification of timer or ACK-cleared events arise. For cases where the Expiration is set, the NetIQ Adapter attempts to minimize spurious re-notifications by not notifying events that would have already expired. In addition, a configuration option is available to allow the NetIQ Adapter to ignore all CLOSED non-clearing Events. Here the assumption is that the event can only be CLOSED because of explicit operator action on the AppManager console, because of a timer expiration in AppManager, or because of a timer expiration in the Adapter. In any case, the event situation has already been addressed and further Notification is not warranted.

B

Generic Event Conversion

The NetIQ AppManager Adapter uses user-defined notification templates to direct the conversion of imported events into InCharge notifications. Templates specify the values to be provided for notification properties and allow complete control of the statused class, object, event, user fields, and other details. Templates specify notification property values as strings composed of both literal substrings and other data elements from four other tabular sources:

- Configuration data from the *adapter.conf* file
- Event data
- Generic adapter-computed results (address data, generic event field manipulation)
- Optional prehook script analysis results.

In addition, a form of static inheritance is available to allow templates to be derived from other templates. This provides a convenient means of localizing common configuration options such as customer-specific user field definitions in select common base templates.

Configuration information in the *event.conf* file associates a particular Notification Template with each Knowledge Script of interest. The event indicates the generating Knowledge Script which, in turn, allows the adapter to identify the proper template and process the event data as shown below. The optional posthook script may create or manipulate InCharge topological elements according to the four data sources previously mentioned or, more generically, the proposed notification.

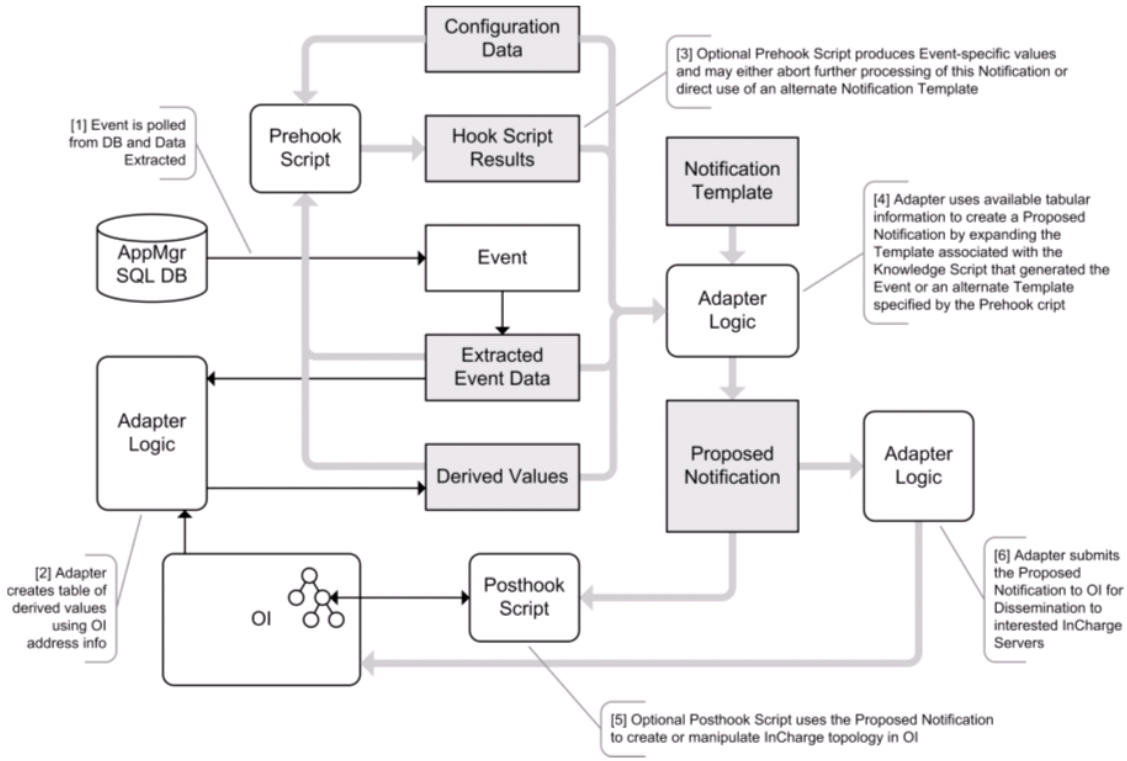

Figure 8: Event to Notification Conversion with Templates

C

Adapter Conf File

This appendix includes a sample *adapter.conf* file. The sample file is for illustrative purposes only.

```
#~ adapter.conf - NetIq AppManager event configuration
#settings
#
# Copyright (c) 2004 System Management ARTS (SMARTS)
# All Rights Reserved
#
# RCS $Id: adapter.conf,v 1.6 2004/03/31 23:04:26 fortiea Exp
#$
# $Source: /src/MASTER/smarts/niqam/conf/adapter.conf,v $
#
#  This conf file is in a format acceptable to /bin/sh.
#Summary:
#-only blank lines, sh-style comments, and simple
#assignments to prescribed variables are allowed;
#expressions, line continuations, etc are not allowed.
#-as with sh, there must be no whitespace surrounding the "="
#-all rvals (string after '=') must be surrounded by quotes
#("") or ('')
#-if double quotes are used, symbol substitution occurs in the
#value string. That is, any existing variable name can be used
#in the construction of another assignment by using standard
#sh curly brace variable notation: ${lval}.  Multiple ${lval}
#substitutions can be used in a single assignment. Some vars
#are predefined and may be similarly used. See log for such
available 'builtin' vars. Of note:
#       ${smHomeDir} = InCharge home (typically /opt/smarts)
#       ${hostname}  = name of this host
#-the default value is used if an explicit assignment is not
#made for any particular variable
```

```
#
#Use care in formatting the values. Parsing errors are logged
#to the adapter log which should be verified after every conf
#change.
#


# Domain Name of InCharge OI Server
#
#The server name may be specified using either of the two
#conventional InCharge notations: "domainName" or
#"hostname:port/domanName" where hostname is either a
#resolvable long or short network name or an IP address.
#
#   If unspecified, default = 'INCHARGE-OI'
#
#oiDmName='INCHARGE-OI'


# ODBC DSN referencing target AppManager Repository "QDB"
#database
#
#By convention the name is a hyphenated catenation of hostname
#and RDB name but may be any string.
#
#   If unspecified, default = "${hostname}-QDB"
#
odbcDsn='NETIQ-QDB'


# Username and password to access AppManager Repository
#database
#
#If the DSN is configured to use windows authorization, these
#values are ignored.
#
#   If unspecified, default = 'sa'
#   If unspecified, default = 'password'
#
#sqlUsername='sa'
#sqlPassword='password'


# Time in seconds to wait between AppManager event polls
#
#Determines approximate periodicity of AppManager SQL
#repository queries to import event states into SAM.
#
#   If unspecified, default = '30'
#
```

```
#eventPollDelay='30'


# Severity Mapping Thresholds
#
#AppManager reports severity using integer values in the
#range of 01-40. The following thresholds partition these
#values into one of four SAM severity categories. The SAM
#'unknown' severity has no AppMgr counterpart. Values greater
#than severityMaxValue_minor are mapped to SAM severity =
#'normal'.
#
#   If unspecified, defaults =
#
#          severityMaxValue_critical='5'   ## 1-5   = critical
#          severityMaxValue_major='10'    ## 6-10  = major
#          severityMaxValue_minor='20'    ## 11-20 = minor
#                                           ## 21-40 = normal
#
#severityMaxValue_critical='5'
#severityMaxValue_major='10'
#severityMaxValue_minor='20'


# Hostname inclusion/exclusion glob expressions
#
#AppManager MachineNames are mapped into internal Smarts
#hostnames for processing. These expressions are applied to
#all raw unmapped AppManager MachineNames prior to mapping.
#MachineNames must match the 'include' expr and must NOT match
#the 'exclude' expr for the event to be processed further.
#
#   If unspecified, include default = '*'    -matches all
#   If unspecified, exclude default = ''    -matches nothing
#
#includedHostGlobExpr='*'
#excludedHostGlobExpr=''


# Synchronization Time period
#
#The Adapter synchronizes InCharge notifications to match
#monitored AppMgr conditions on startup and on re-connection
#to OI. This involves reprocessing historical AppMgr Events
#and can be very time-consuming for large SQL repositories.
#The next settings are combined to determine the time in the
#past from which to begin synchronization. Use these settings
#to limit the historical view and reduce the synchronization
#time. Existing Notifications for conditions occuring prior
```

```
#to the computed sync start time will be cleared. Specify 0
#for both values to synchronize all Events reqardless of age.
#
#   If unspecified, 'Days' default = '14'-ignore Events > 2
#weeks old
#   If unspecified, 'Hours' default = '0'
#
#syncTimePeriodDays='14'
#syncTimePeriodHours='0'


# Accept Events from Unknown Hosts
#
#Boolean. If FALSE, all Events from Hosts unknown to InCharge
#are rejected.
#
#If TRUE, Events relating to Hosts not Discovered by AM nor
#known to OI will still be accepted. The AppManager
#MachineName from the Event Detail will be used in the Adapter
#in place of the resolved Unitary Computer System Name
#normally obtained from OI for discovered Hosts. The Adapter
#creates the target Host object in OI.
#
#   If unspecified, default = 'FALSE'
#
#acceptEventsFromUnknownHosts='TRUE'


# Event processing mapping file
#
#This string is the name of a file that defines Notification
#Templates and maps Knowledge Script names to these
#templates. The filespec is relative to smarts/conf or
#smarts/local/conf.
#
#   If unspecified, default = 'niqam/event.conf'
#
#eventConfFile='niqam/event.conf'


# Pre-synchronization hook script
#
#If non-null, this string is the name of an optional
#hookscript to perform synchronization tasks before
#outstanding AppManager events are processed for
#synchronization. The internal adapter event tables have been
#initialized before this hook is invoked. The filespec is
#relative to smarts/rules or smarts/local/rules.
#
#   If unspecified, default = 'niqam/event.conf'
```

```
#
#   If unspecified, default = ''
#
#presyncHookScript='niqam/hook/presyncHookScriptSample.asl'


# Post-synchronization hook script
#
#If non-null, this string is the name of an optional
#hookscript to perform synchronization tasks after
#outstanding AppManager events are processed for
#synchronization. The internal adapter stale event tables
#have not yet been processed when this hook is invoked. The
#filespec is relative to smarts/rules or smarts/local/rules.
#
#   If unspecified, default = ''
#
#postsyncHookScript='niqam/hook/postsyncHookScriptSample.asl'


# Time in seconds to delay between server reconnection
#attempts
#
#Determines approximate periodicity of SQL and IC server
#reconnection checks to allow resynchronization and
#recommencement of processing. This delay only affects
#retries after connection loss or other unrecoverable program
#errors; it does not cause initial startup latency.
#
#   If unspecified, default = '60'
#
#epochRetryDelay='60'


# Enable Auto-Close of 'Status Change' Events in AppMgr
#
#Boolean. If TRUE, causes Adapter to CLOSE all Status Change
#Events in AppMgr. These signal that the monitored condition
#has been removed or unasserted. This serves no purpose other
#than to minimize housekeeping tasks in AppMgr.
#
#   If unspecified, default = 'TRUE'
#
#enableAutoCloseStatusChangeEvents='FALSE'


# Time in seconds to delay before auto-closing 'Status Change'
#Events in AppMgr
#
```

```
#Determines minimum time to delay before auto-closing 'Status
#Change' Events in AppMgr.  This should be set to some non-
#zero value for redundant configurations to allow sufficient
#opportunity for all Adapters to process the raw Event.
#Requires AutoClose of Status Change Events to be enabled.
#
#   If unspecified, default = '0'
#
#statusChangeEventAutoCloseDelay='0'


# Enable AutoClose Directives
#
#Boolean. If TRUE, causes Adapter to CLOSE events targetted by
#either "CloseSourceEvent:" (event.conf) or "AUTO_CLOSE"
#(hook script) directives.
#
#   If unspecified, default = 'TRUE'
#
#enableAutoCloseDirectives='FALSE'


# Time in seconds to delay before auto-closing non-clearing
#Events in AppMgr
#
#Determines minimum time to delay before auto-closing Events
#in AppMgr which generate notifications which are cleared by
#either "CloseSourceEvent:" (event.conf) or "AUTO_CLOSE"
#(hook script) directives. This should be set to some non-zero
#value for redundant configurations to allow sufficient
#opportunity for all Adapters to process the raw Event. Large
#values may cause previously cleared notifications to re-
#occur during Adapter synchronization because the generating
#Event persists in an OPEN state.
#
#   If unspecified, default = '0'
#
#nonClearingEventAutoCloseDelay='0'


# Enable Ignore Closed NOTIFY_WO_CLEAR Events During Sync
#
#Boolean. If TRUE, CLOSED Events causing NOTIFY_WO_CLEAR
#Notifications are ignored during sync phase. This prevents
#renotification by Events which are likely to have been
#cleared by ClearOnACK or which have been resolved and closed
#in AppMgr. When combined with the 'ClearSourceEvent:'
#Template directive, this option can minimize spurious
#Notifications.
#
```

```
#   If unspecified, default = 'TRUE'
#
#enableSyncIgnoreClosedNotifyWoClearEvents='FALSE'


#=========[ Options Useful for Problem Resolution ]=========


# Enable Missing UCS Event Logging
#
#Boolean. If TRUE, Adapter logs events skipped because no
#corresponding UCS exists in OI or the Machine Name does not
#pass the glob expressions above. This option is meaningless
#if acceptEventsFromUnknownHosts='TRUE' above.
#
#   If unspecified, default = 'FALSE'
#
enableMissingUcsLogging='TRUE'


# Enable Undefined Template Logging
#
#Boolean. If TRUE, Adapter logs events skipped because no
#Template is defined for the associated Knowledge Script
#
#   If unspecified, default = 'FALSE'
#
enableUndefinedTemplateLogging='TRUE'


# Enable Template logging
#
#Boolean. If TRUE, template definitions are logged as
#event.conf is parsed. Provided as a debugging tool for
#Template Developers.
#
#   If unspecified, default = 'FALSE'
#
#enableTemplateLogging='TRUE'


# Enable Event Logging
#
#Boolean. If TRUE, Events imported from AppManager are
#logged.
#
#   If unspecified, default = 'TRUE'
#
#enableEventLogging='FALSE'
```

```
# Enable Event Detail Logging
#
#Boolean. If TRUE, details of Event import are logged.
#Requires event logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
#enableEventDetailLogging='TRUE'


# Enable Poll Time Logging
#
#Boolean. If TRUE, each poll is logged with the current time
#in UTC of the SQL Server (not Adapter local time). Requires
#event logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
enablePollTimeLogging='TRUE'


# Enable Proposed Notification Logging
#
#Boolean. If TRUE, the proposed Notification is logged prior
#to post-hook script execution and submission to OI. Requires
#event logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
#enableProposedNotifLogging='TRUE'


# Enable Rejected Notification Logging
#
#Boolean. If TRUE, warning are logged whenever a hook script
#rejects submission of a proposed notification.  Requires
#event logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
#enableRejectedNotifLogging='TRUE'


# Enable Active Notification Logging
#
#Boolean. If TRUE, the internal Adapter table of Active
#Notifications is logged for every significant processed
#Event. This can generate very large logs and is intended as a
#problem resolution tool to assist in diagnosing issues of
```

```
#missed or improperly handled events. Use cautiously.
#Requires event logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
#enableActiveNotificationLogging='TRUE'


# Enable Aggregate Notification Logging
#
#Boolean. If TRUE, NOTIFY operations on Aggregate
#Notifications are logged.
#Since these parallel NOTIFY of the child notification, they
#are not generally interesting. This option is provided as a
#debugging aid for Template Developers.  Requires event
#logging to be enabled.
#
#   If unspecified, default = 'FALSE'
#
#enableAggregateLogging='TRUE'


# Maximum number of 'new' Events logged during Sync Phase
#
#If Event Logging is enabled, each new or changed event is
#first logged as "new (evt,sts) = ....". During
#synchronizations ALL events in the SQL DB are considered new.
#This parameter may be used to limit logging of these old
#events. This has no effect on logging of the *processing* of
#these events, just their initial enumeration. A value of '0'
#logs all events.
#
#   If unspecified, default = '500'
#
maxSyncPhaseEventsLogged='1'

#### eof ####
vim:ts=8:sw=4:sts=4:tw=79:fo=tcqrnol:noet:
```

D

Event Processing Conf File

This appendix includes a sample *event.conf* file. The sample file is for illustrative purposes only.

```
#~ event.conf - map of Knowledge Scripts to Event processing
#Scripts
#
# Copyright (c) 2004 System Management ARTS (SMARTS)
# All Rights Reserved
#
# RCS $Id: event.conf,v 1.5 2004/03/31 23:05:49 fortiea Exp $
# $Source: /src/MASTER/smarts/niqam/conf/event.conf,v $
#
#
# This configuration file may contain empty lines or sh-style
#comments. All other lines are considered part of
#definitions. The following discussion uses this notation:
#
#<description>= <> description of a parameter; the "<>" are
#                notational metacharacters and should not be
#                included in actaul use.
#{foo|bar}= {||} a choice selection of *only* the values
#shown
#   [<anything>]= []optional elements
#
#
#  There are 2 definition types:
#
#   (Type 1) Notification Template
#
#   TEMPLATE <templateName> [inherits from
#<baseTemplateName>]
#   ClassName:   <defining value>    ## required
```

```
#   InstanceName:<defining value>    ## required
#   EventName:   <defining value>    ## required
#
#   [<modifier>]
#   ...
#   [<modifier>]
#
#   where a <modifier> is one of:
#
#       <propertyDefinition>
#       <aggregateDefinition>
#       <hookScriptSpecifier>
#       <controlSpecifier>
#
#
#   --><propertyDefinition> is one of the following:
#
#       ClassDisplayName: <value>  # defaults by IC to
#ClassName
#       EventDisplayName: <value>  # defaults by IC to
#EventName
#       InstanceDisplayName: <value>  # defaults by IC to
#InstanceName
#
#       Category:{'Performance'|'Availability'|'Error'|etc}
#       Certainty:'100'
#       InMaintenance:{'TRUE'|'FALSE'}
#       ElementClassName: <value>
#       ElementName:<value>
#       EventText:<value>
#       EventType:{'DURABLE'|'MOMENTARY'}
#       Severity:"${A_Severity}" or
#                '{critical|major|minor|normal}'
#UserDefined1:<value>
#       UserDefined2:<value>
#       UserDefined3:<value>
#       UserDefined4:<value>
#       UserDefined5:<value>
#       UserDefined6:<value>
#       UserDefined7:<value>
#       UserDefined8:<value>
#       UserDefined9:<value>
#       UserDefined10:<value>
#
#The ClassName, InstanceName, EventName specifiers are
#required.
#The remaining properties are set to "" if unspecified and
#will thus receive the system defaults. If the optional
#'inherits from' modifier is used, the current template is
#cloned from the referenced base template instead of being
```

```
#being initialized with cleared property values. Any
#<modifier> definitions in a derived template override those
#in the base template allowing minor customizations w/o
#building a new template.
#
#
#   --><aggregateDefinition> is defined as:
#
#       aggregate:{
#                EventName:  <defining value>## required
#                [<propertyDefinition>]
#                ...
#                [<propertyDefinition>]
#                }
#
#If aggregation is desired, an EventName *must* be specified;
#all other properties are optional. For simplicity, when
#Aggregation is used, the subordinate Notifications should
#NOT specify EventNames which can be correlated. For example,
#such subordinate Templates might be derived from the default
#NOTIFY_ONLY Template (which specifies an EventName
#meaningless to ASM) while specifying 'DegradedSymptom' for
#the aggregate EventName.
#
#
#   --><hookScriptSpecifier> is a record of either form:
#
#      PREHOOK({SELF|OI}):  <aslFileNameToProcessEvent>
#[args]
#      POSTHOOK({SELF|OI}): <aslFileNameToProcessEvent>
#[args]
#
#Hook scripts are passed the specified args as list elements
#of the global HOOK_PARMS. Hook script filenames are relative
#to .../rules or .../local/rules. Only one prehook script and
#one posthook script is allowed per template (the last
#mentioned applies). The {SELF|OI} parm determines where
#actions are sent. SELF indicates no remote server accesses
#are required, while OI indicates that object references
#should manipulate the remote OI server.
#
#The prehook script is invoked after the adapter has
#calculated the ${A_*} variables but before the template is
#applied. The script may compute additional candidates for
#template substitution as ${H_<varname>}. Prehook scripts
#usually specify SELF.
#
#The posthook script is invoked after the template has been
#applied but before the proposed Notification is created.
```

```
#This allows values from the proposed Notification to be used
#to drive the script. Posthook scripts usually specify OI.
#
#   --><controlSpecifier> is one of:
#
#      State:    {'AUTO'|'CLEAR'|'NOTIFY'|'NOTIFY_WO_CLEAR'}
#      Expiration:'<seconds until Notification autoClear>'
#      ClearOnAcknowledge: {'TRUE'|'FALSE'}
#      CloseSourceEvent:   {'TRUE'|'FALSE'}
#
#The 'State:' specifier describes the action to be applied to
#the Notification. 'AUTO' indicates that the Adapter should
#NOTIFY on the initial Event and CLEAR on the associated
#'State Change' Event. The Adapter discards 'State Change'
#Events for all non-AUTO settings. 'NOTIFY_WO_CLEAR'
#indicates the Adapter should NOTIFY on the initial Event but
#should maintain no state nor expect an associated CLEAR from
#any Event for the Notification. Such notifications should
#specify ClearOnAcknowledge/Expiration. The 'NOTIFY' and
#'CLEAR' settings do as the name suggests.
#
#The 'CloseSourceEvent:' specifier determines if the Adapter
#will close the Event in AppManager. This feature may be used
#to minimize AppMgr housekeeping tasks.
#
#
#   (Type 2) Knowledge Script Template Processing directive
#
#   KS <ksName> <templateName>
#       [<modifier>]
#       ...
#       [<modifier>]
#
#Any <modifier> defined in a Template may be overridden in a
#KS directive by re-specifying the <modifier> with the new
#value.
#
#
==============================================================
#
#Notification Templates must be defined in the file before
#they are referenced in other definitions.
#
#Values assigned in <propertyDefinition>s be literal text or
#may optionally contain parameter names of the form:
#${varName}. The parameter values are substituted literally
#without further processing.
#
#
#   --> These raw Event values are available for substitution:
```

```
#
#   ${E_EventID}    -unique numeric event identifier
#   ${E_ParentEventID} -event ID of parent event
#   ${E_JobID}-numeric Job ID of job that created the event
#   ${E_MachineName} -full Machine Name as it appears in
#TreeView
#   ${E_Status}    -numeric event status
#                           Openstatus = 0
#                       Acknowledgedstatus = 1
#                             Closedstatus = 2
#                            Deletedstatus = 3
#   ${E_KPName}    -name of Knowledge Script
#   ${E_Severity}    -numeric everity (1-40) (see
#adpater.conf)
#   ${E_EventMsg}    -free form message describing Event
#   ${E_FirstOccurTime} -UTC of first event notification
#   ${E_AgentMsgShort}  -text string provided by Knowledge
#Script (0-255)
#   ${E_AgentMsgLong}   -text string provided by Knowledge
#Script (0, 255+)
#   Note- AgentMsgShort and AgentMsgLong are
#   exclusive; neither present if no message, Short
#   used up to 255 chars, Long used for 255+
#
#
#   --> These Adapter-generated values are available for
substitution:
#
#   ${A_StatusAsText}   = ${E_Status} mapped to text
#   ${A_AgentMsg} = ${E_AgentMsgShort} or ${E_AgentMsgLong}
#   ${A_UcsName} = ${E_MachineName} resolved to IC UCS name
#   ${A_UcsClass} = ${E_MachineName} resolved to IC
#Creation Class Name
#   ${A_Severity} = ${E_Severity} mapped according to
#adapter.conf
#   ${A_SeverityAsText} = ${A_Severity} in text format
#
#   E_KPName is of form:[<KSG>:]<AppGroup>_<RootName>
#
#   ${A_KS_Group} = ${E_KPName} group prefix (nnnn before ":")
#or ""
#   ${A_KS_GroupC} = ${E_KPName} group prefix w/colon (nnnn:)
#or ""
#   ${A_KS_Name} = ${E_KPName}  w/o any "<KSG>:" prefix
#   ${A_KS_AppGroup} = ${A_KS_Name} prefix (before "_")
#   ${A_KS_RootName} = ${A_KS_Name} suffix (after  "_")
#
#
#   --> These CONF parameters (from adapter.conf) are
#available for substitution:
```

```
#
#   ${C_odbcDsn} -ODBC DSN referencing AppManager Repository
#   ${C_sqlUsername}    -username to access SQL DB
#   ${C_adapterName}    -domain name of this adapter
#   ${C_hostname}    -name of host on which adapter is running
#   ${C_serverName} -domain name of server to which adapter
#connects
#
#
#   --> These Hookscript results are available for
#substitution:
#
#   ${H_<varname>} -varname is hookscript-dependent as the
#index of the global Table HOOK_DATA
#
#
#Enclose constant values in single quotes ('') and values
#containing ${...} in double quotes ("""). Variable
#substitution is not performed within single quoted strings.
#Single character escaping is not supported (e.g. \").
#
#
#   An Important Note on Event Filtering:
#
#In the examples below, a sample prehook script is used in
#every definition to filter Events for processing. The sample
#rejects all Events which do not match the glob expression
#specified as a parameter to the hook script. Use of this
#sample script or other custom hook scripts to achieve this
#same end is required to avoid producing Notifications from
#unexpected Events (e.g. Error Events, etc). Failure to
#follow this policy may result in creation of meaningless
#Notifications which will persist in InCharge since no
#corresponding CLOSE Event is likely to occur.
#
#
## Default Base Template from which to derive other Templates
#
#This template is not used directly by KS specifiers but may
#be used as a base Template from which to derive other
#Templates that will share these common property settings.
#
#This is for illustrative puposes only. If desired, Templates
#may be defined individually without derivation from any base
#Template(s). The Adapter does not require definition of any
#Templates with any particular names nor does it require a
#common base Template. Reproducing the Class/Instance/Event
#names in derived Templates, though unnecessary, may improve
#readibilty.
#
```

```
TEMPLATE DEFAULT
    ClassName:    '<REQUIRES_OVERRIDE>'
    InstanceName:    "${C_odbcDsn}"
    EventName:    "${E_KPName}-${E_EventID}"
    #ClassDisplayName:    ## default = ClassName
    #EventDisplayName:    ## default = EventName
    #InstanceDisplayName:    ## default = InstanceName
    Category:    'Performance'
    #Certainty:    '100'    ## Default = 100
    #InMaintenance:    'TRUE'    ## Default = FALSE
    #ElementClassName:'Host'    ## Default = ${A_UcsClass}
    #ElementName:    'myHost'    ## Default = ${A_UcsName}
    EventText:    "${E_EventMsg}"
    EventType:    'DURABLE'
    Severity:    "${A_Severity}" ##  or
{critical|major|minor|normal}
    UserDefined1:    "Machine Name = ${E_MachineName}"
    UserDefined2:    "NetIQ Status = ${E_Status}"
    UserDefined3:    "EventID = ${E_EventID}"
    UserDefined4:    "ParentEventID = ${E_ParentEventID}"
    UserDefined5:    "JobID = ${E_JobID}"
    UserDefined6:    "UCS Class = ${A_UcsClass}"
    UserDefined7:    "${A_AgentMsg}"
    #UserDefined8:    ''
    #UserDefined9:    ''
    #UserDefined10:    ''
    #State:    'AUTO'## Default = 'AUTO'
    #Expiration:    '0'## Default = 0
    #ClearOnAcknowledge:    'FALSE'## Default = FALSE
    #CloseSourceEvent:    'FALSE'## Default = FALSE


## Default Template to notify of non-clearing Events to be
#cleared by ACK
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#Notifications generated by this Template will CLEAR only by
#Expiration. This Template may be specified for Knowledge
#scripts that generate events for which no matching Event is
#generated of the "Status Change" type. Other Templates may be
#derived from this with ClassName adjusted to more meaningful
#values.
#
TEMPLATE EXPIRE inherits from DEFAULT
    ClassName:    'NETIQ-EXPIRE'
    State: "NOTIFY_WO_CLEAR"## there is no matching CLEAR
    Expiration:    '7200            '## clear notifs from OI
    CloseSourceEvent:'TRUE    '## clear Event in AppMgr
```

```
## Default Template to notify of non-clearing Events to be
#cleared by ACK
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#Notifications generated by this Template will CLEAR by
#Expiration or Acknowledgement.  This Template may be
#specified for Knowledge scripts that generate events for
#which no matching Event is generated of the "Status Change"
#type. Other Templates may be derived from this with ClassName
#adjusted to more meaningful values.
#
TEMPLATE CLEAR_ON_ACK inherits from DEFAULT
    ClassName:    'NETIQ-CLR-ON-ACK'
    State:"NOTIFY_WO_CLEAR"## there is no matching CLEAR
    Expiration:    '28800     '## clear notifs from OI
    CloseSourceEvent:'TRUE    '## clear Event in AppMgr
    ClearOnAcknowledge:'TRUE  '## allow quick clear


## Default General Error Template to notify of non-clearing
#Error Events
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#This Template may be specified by prehook scripts in the
#ERROR_TEMPLATE parameter when parsed Event messages reveal
#errors have occurred.
#
TEMPLATE GENERAL_ERROR inherits from CLEAR_ON_ACK
    ClassName:    'NETIQ-Error'
    Category:    'Error'


## Default "AGENT MSG" Error Template to notify of non-
#clearing Error Events
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#This Template assumes that the interesting error string is
#located in the Agent Message instead of the EventMsg and so
#interchanges these fields in the generated Notification.
#Error Events in this format often accompany Events in the
#GENERAL_ERROR format that contain the *same* error
#information in an alternate field.
```

```
#
TEMPLATE AGENT_MSG_ERROR inherits from GENERAL_ERROR
   EventText:"${A_AgentMsg}" ## switch properties where these
two
    UserDefined7:"${E_EventMsg}" ## pieces of Event info
appear in NOTIF


## Default Notification Template to create a simple
#aggregating AppMgr alert
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#This is a basic Template to use for Events which generate an
#accompanying CLEAR Event of the "State Change" type.
#
TEMPLATE NOTIFY_ONLY inherits from DEFAULT
    ClassName:    'NETIQ'
    InstanceName:    "NOTIFY-${A_KS_AppGroup}/${A_UcsName}"
    EventName:    "${E_KPName}"
    #ClassDisplayName:    ## default = ClassName
    #EventDisplayName:    ## default = EventName
    #InstanceDisplayName:    ## default = InstanceName
    Category:    'Performance'
    #ElementClassName:    'Host'    ## Default =${A_UcsClass}
    #ElementName:    'myHost' ## Default = ${A_UcsName}
    Aggregate: {
       #ClassName:            ## default = Notif ClassName
       #InstanceName:         ## default = Notif InstanceName
       EventName:             'Notify-Only'
       #ClassDisplayName:## default = Aggregate ClassName
       #InstanceDisplayName:## default = Aggregate
InstanceName
       #EventDisplayName:## default = Aggregate EventName
       Category:              'Aggregate'
    }
    State:    "AUTO"  ## NOTIFY/CLEAR on orig/StateChange
Events
#Expiration:            ''
    #ClearOnAcknowledge:    'FALSE'
    #CloseSourceEvent:    'TRUE'  ## ignored for "AUTO" events


## Default Notification Template to status Host
#
#  The specified Class and/or Event are known to ASM.
#  The generated notifications can be correlated.
#
```

```
#This template specifies valid Class and Instance but an Event
#unknown to ASM. However, it specifies valid Class, Instance,
#and Event known to ASM for an aggregating parent
#notification.
#
TEMPLATE Host inherits from NOTIFY_ONLY
    ClassName:   'Host'
    InstanceName:    "${A_UcsName}"
    EventName:    "${E_KPName}"
    Aggregate: {
       EventName:    'DegradedSymptom'
    }


## Default Notification Template to status Server
#
#  The specified Class and/or Event are known to ASM.
#  The generated notifications can be correlated.
#
#Based on the Host template, this template specifies
#DisplayClassName to make the Global Console display "Server"
#instead of the *true* ClassName which is "Host".
#
TEMPLATE Server inherits from Host
    ClassDisplayName:    'Server'
    Aggregate: {
       ClassDisplayName:   'Server'
    }


## Other Sample Server Templates
#
TEMPLATE NT inherits from Host
    ClassDisplayName:'NT Server'
    Aggregate:{ ClassDisplayName: 'NT Server' }

TEMPLATE UNIX inherits from Host
    ClassDisplayName:'UNIX Server'
    Aggregate:{ ClassDisplayName: 'UNIX Server' }

TEMPLATE DELL_SERVER inherits from Host
    ClassDisplayName:'DELL Server'
    Aggregate:{ ClassDisplayName: 'DELL Server' }

TEMPLATE HP_SERVER inherits from Host
    ClassDisplayName:'HP Server'
    Aggregate:{ ClassDisplayName: 'HP Server' }

TEMPLATE IBM_SERVER inherits from Host
    ClassDisplayName:'IBM Server'
```

```
    Aggregate:{ ClassDisplayName: 'IBM Server' }

TEMPLATE SIEMENS inherits from Host
    ClassDisplayName:'Siemens Server'
Aggregate:{ ClassDisplayName: 'Siemens Server' }

TEMPLATE WIN2000 inherits from Host
    ClassDisplayName:'WIN2000'
    Aggregate:{ ClassDisplayName: 'WIN2000' }

TEMPLATE WIN2003 inherits from Host
    ClassDisplayName:'WIN2003'
    Aggregate:{ ClassDisplayName: 'WIN2003' }


## Sample Template for pre-existing Singleton Application
#topological objects
#
#  The specified Class and/or Event are known to ASM.
#  The generated notifications can be correlated.
#
#Use/modify this Template to status General or specific
#application objects already existing in the topology. Adding
#appropriate hook scripts can allow the objects to be
#dynamically created as Events are encountered.
#
#   By convention Applications have InstanceNames of:
#   APP-<SpecificApplicationInstanceName>/<Hostname>
#
#A 'singleton application' is an application for which there
#can be only one instance per host. This may be an
#administration policy or an application restriction.
#InCharge topology administration is simplified for such
#applications because other distinguishing name elements are
#not required to disambiguate multiple applications per host.
#
#For Singleton applications with no known
#<SpecifiApplicationInstanceName>, we can differente
#Applications by the host on which they run. The next Template
#achieves this by using the NetIQ Application Group and the
#discovered OI Hostname to create a host-unique InstanceName
#(e.g. "APP-EXCHANGE/bigServer.com").
#
#   By convention Applications have InstanceNames of:
#   APP-<SpecificApplicationInstanceName>/<Hostname>
#
TEMPLATE APPLICATION inherits from DEFAULT
    ClassName:'Application'
    InstanceName:"APP-${A_KS_AppGroup}/${A_UcsName}"
    EventName:"${E_KPName}"
```

```
    Aggregate: {
       EventName:'DegradedSymptom'
       Category: 'Aggregate'
    }


## Sample KS Template for monitoring App health via
#ProcessDown
#
#  The specified Class and/or Event are known to ASM.
#  The generated notifications can be correlated.
#
#Use/modify this Template to status Application objects. If
#the Application does not exist it is created with the
#application name parsed from the Event message. If the same
#applications are monitored and statused by other Knowledge
#scripts then the InstanceNames must be reconciled to
#identify the same topological object.
#
#   By convention Applications have InstanceNames of:
#   APP-<SpecificApplicationInstanceName>/<Hostname>
#
KS  NT_ProcessDownAPPLICATION
InstanceName:"APP-${H_ApplicationName}/${A_UcsName}"
    Aggregate: {
       EventName:'DownSymptom'
    }
    PREHOOK(SELF):niqam/hook/NT_ProcessDown-pre.asl
    POSTHOOK(OI):niqam/hook/createApplication-post.asl


## Sample Template to notify of Discovery Events
#
#  The specified Class and/or Event are unknown to ASM.
#  The generated notifications cannot be correlated.
#
#This Template causes Events from Discovery to create
#expiring Notifications which may also be cleared by ACK. The
#sample hook scripts illustrate how special error messages
#may be detected and used to cause alternative Error Templates
#to be expanded. Use the AppMgr Developer Console to open and
#examine Knowledge Scripts for Error Messages of interest.
#
TEMPLATE DISCOVERY inherits from CLEAR_ON_ACK
    ClassName:'NETIQ-Discovery'
    State:"NOTIFY_WO_CLEAR"   ## there is no matching CLEAR
    Category:'Discovery'
    Expiration:  '28800        '    ## clear notifs from OI
    CloseSourceEvent:'TRUE'    ## clear Event in AppMgr
    ClearOnAcknowledge:'TRUE'    ## allow quick clear
```

```
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl '*
Discovery OK'

KS  Discovery_ExchangeDISCOVERY
KS  Discovery_NTDISCOVERY


## Sample definitions using private (renamed) Knowledge
#Script versions
#
#Note that the knowledge script names reflect the true file
#names *NOT* the all-uppercase version displayed on the tabs
#of the AppManager console script pane.  These examples are
#for illustrative purposes only. Delete or modify as
#required.
#
KS  smNT_CpuByProcessNT
    EventName:"${E_KPName}-${H_ApplicationName}"
    PREHOOK(SELF):niqam/hook/NT_CpuByProcess-pre.asl

KS  smNT_CpuLoadedNT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'CPU# *
Overloaded'

KS  smNT_LogicalDiskBusy NT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'L* busy'

KS  smNT_LogicalDiskIO NT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'L* busy'

KS  smNT_LogicalDiskSpace NT
    EventName:"${E_KPName}-${H_Disk}"
    PREHOOK(SELF):niqam/hook/NT_LogicalDiskSpace-pre.asl

KS  smNT_MemByProcess NT
    EventName:"${E_KPName}-${H_ApplicationName}"
    PREHOOK(SELF):niqam/hook/NT_MemByProcess-pre.asl

KS  smNT_MemUtilNT
    EventName:"${E_KPName}-${H_UsageType}"
    PREHOOK(SELF):niqam/hook/NT_MemUtil-pre.asl
KS  smNT_PagingHighNT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'High
Memory Paging'

#   'Disk 0 C: Busy'
KS  smNT_PhysicalDiskBusy NT
    EventName:"${E_KPName}-${H_W2}-${H_W3}"
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'Disk *
Busy'
```

```
#   'Pdsk 0 C: {write|xfer|read} busy'
KS  smNT_PhysicalDiskIONT
    EventName:"${E_KPName}-${H_W4}-${H_W2}-${H_W3}"
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'Pdsk *
busy'

KS  smNT_ProcessDownNT
    EventName:"${E_KPName}-${H_ApplicationName}"
    PREHOOK(SELF):niqam/hook/NT_ProcessDown-pre.asl

KS  smNT_ProcessUpNT
    EventName:"${E_KPName}-${H_ApplicationName}"
    PREHOOK(SELF):niqam/hook/NT_ProcessUp-pre.asl

KS  smNT_ProcessesNT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl '# of
Processes High'

KS  smNT_ServiceDownNT
    EventName:"${E_KPName}-${H_W4}"
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'Down
Service - *'

KS  smNT_SystemUpTimeNT
    PREHOOK(SELF):niqam/hook/eventMsgFilter-pre.asl 'Machine
has recently *'


#### eof ####
vim:ts=8:sw=4:sts=4:tw=79:fo=tcqrnol:noet:
```

Index