Netcool/Impact
Version 6.1.0.1

Solutions Guide

SC23-8834-04

Netcool/Impact
Version 6.1.0.1

Solutions Guide

SC23-8834-04

Note
Before using this information and the product it supports, read the information in "Notices".

Edition notice

This edition applies to version 6.1.0.1 of IBM Tivoli Netcool/Impact and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

The *Solutions Guide* contains end-to-end information about using features in Netcool/Impact.

Intended audience

This publication is for users who are responsible creating Netcool/Impact data models, writing Netcool/Impact policies and running Netcool/Impact services.

Publications

This section lists publications in the Netcool/Impact library and related documents. The section also describes how to access Tivoli® publications online and how to order Tivoli publications.

Netcool/Impact library

- *Quick Start Guide*, CF39PML

  Provides concise information about installing and running Netcool/Impact for the first time.
- *Administration Guide*, SC23882904

  Provides information about installing, running and monitoring the product.
- *User Interface Guide*, SC23883004

  Provides instructions for using the Graphical User Interface (GUI).
- *Policy Reference Guide*, SC23883104

  Contains complete description and reference information for the Impact Policy Language (IPL).
- *DSA Reference Guide*, SC23883204

  Provides information about data source adaptors (DSAs).
- *Operator View Guide*, SC23885104

  Provides information about creating operator views.
- *Solutions Guide*, SC23883404

  Provides end-to-end information about using features of Netcool/Impact.
- *Integrations Guide*, SC27283402

  Contains instructions for integrating Netcool/Impact with other IBM® software and other vendor software.
- *Troubleshooting Guide*, GC27283302

  Provides information about troubleshooting the installation, customization, starting, and maintaining Netcool/Impact.

Accessing terminology online

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

http://www.ibm.com/software/globalization/terminology

Accessing publications online

Publications are available from the following locations:

- The *Quick Start* DVD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. Refer to the readme file on the DVD for instructions on how to access the documentation.
- Tivoli Information Center web site at http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcoolimpact.doc6.1/welcome.html. IBM posts publications for all Tivoli products, as they become available and whenever they are updated to the Tivoli Information Center Web site.

  Note: If you print PDF documents on paper other than letter-sized paper, set the option in the **File → Print** window that allows Adobe Reader to print letter-sized pages on your local paper.
- Tivoli Documentation Central at http://www.ibm.com/developerworks/wikis/display/tivolidoccentral/Impact. You can also access publications of the previous and current versions of Netcool/Impact from Tivoli Documentation Central.
- The Netcool/Impact wiki contains additional short documents and additional information and is available at https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/Tivoli%20Netcool%20Impact.

Ordering publications

You can order many Tivoli publications online at http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss.

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss.
2. Select your country from the list and click **Go**.
3. Click **About this site** in the main panel to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For additional information, see “Accessibility,” on page 183.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site at http://www.ibm.com/software/tivoli/education.

Support for problem solving

If you have a problem with your IBM software, you want to resolve it quickly. This section describes the following options for obtaining support for IBM software products:

- "Obtaining fixes"
- "Receiving weekly support updates"
- "Contacting IBM Software Support" on page x

Obtaining fixes

A product fix might be available to resolve your problem. To determine which fixes are available for your Tivoli software product, follow these steps:

1. Go to the IBM Software Support Web site at http://www.ibm.com/software/support.
2. Navigate to the **Downloads** page.
3. Follow the instructions to locate the fix you want to download.
4. If there is no **Download** heading for your product, supply a search term, error code, or APAR number in the search field.

For more information about the types of fixes that are available, see the *IBM Software Support Handbook* at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html.

Receiving weekly support updates

To receive weekly e-mail notifications about fixes and other software support news, follow these steps:

1. Go to the IBM Software Support Web site at http://www.ibm.com/software/support.
2. Click the **My IBM** in the toobar. Click **My technical support**.
3. If you have already registered for **My technical support**, sign in and skip to the next step. If you have not registered, click **register now**. Complete the registration form using your e-mail address as your IBM ID and click **Submit**.
4. The **Edit profile** tab is displayed.
5. In the first list under **Products**, select **Software**. In the second list, select a product category (for example, **Systems and Asset Management**). In the third list, select a product sub-category (for example, **Application Performance & Availability** or **Systems Performance**). A list of applicable products is displayed.
6. Select the products for which you want to receive updates.
7. Click **Add products**.
8. After selecting all products that are of interest to you, click **Subscribe to email** on the **Edit profile** tab.
9. In the **Documents** list, select **Software**.
10. Select **Please send these documents by weekly email**.
11. Update your e-mail address as needed.
12. Select the types of documents you want to receive.
13. Click **Update**.

If you experience problems with the **My technical support** feature, you can obtain help in one of the following ways:

Online

Send an e-mail message to erchelp@u.ibm.com, describing your problem.

By phone

Call 1-800-IBM-4You (1-800-426-4409).

World Wide Registration Help desk

For word wide support information check the details in the following link: https://www.ibm.com/account/profile/us?page=reghelpdesk

Contacting IBM Software Support

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational® products, and DB2® and WebSphere® products that run on Windows or UNIX operating systems), enroll in Passport Advantage® in one of the following ways:

  Online

  Go to the Passport Advantage Web site at http://www-306.ibm.com/software/howtobuy/passportadvantage/pao_customers.htm .

  By phone

  For the phone number to call in your country, go to the IBM Worldwide IBM Registration Helpdesk Web site at https://www.ibm.com/account/profile/us?page=reghelpdesk.
- For customers with Subscription and Support (S & S) contracts, go to the Software Service Request Web site at https://techsupport.services.ibm.com/ssr/login.
- For customers with IBMLink, CATIA, Linux, OS/390®, iSeries®, pSeries®, zSeries®, and other support agreements, go to the IBM Support Line Web site at http://www.ibm.com/services/us/index.wss/so/its/a1000030/dt006.
- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web site at http://www.ibm.com/servers/eserver/techsupport.html.

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. From other countries, go to the contacts page of the *IBM Software Support Handbook* on the Web at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html and click the name of your geographic region for phone numbers of people who provide support for your location.

To contact IBM Software support, follow these steps:

1. “Determining the business impact” on page xi
2. “Describing problems and gathering information” on page xi
3. “Submitting problems” on page xi

Determining the business impact

When you report a problem to IBM, you are asked to supply a severity level. Use the following criteria to understand and assess the business impact of the problem that you are reporting:

Severity 1
: The problem has a *critical* business impact. You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.

Severity 2
: The problem has a *significant* business impact. The program is usable, but it is severely limited.

Severity 3
: The problem has *some* business impact. The program is usable, but less significant features (not critical to operations) are unavailable.

Severity 4
: The problem has *minimal* business impact. The problem causes little impact on operations, or a reasonable circumvention to the problem was implemented.

Describing problems and gathering information

When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- Which software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can you re-create the problem? If so, what steps were performed to re-create the problem?
- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, and so on.
- Are you currently using a workaround for the problem? If so, be prepared to explain the workaround when you report the problem.

Submitting problems

You can submit your problem to IBM Software Support in one of two ways:

Online
: Click **Submit and track problems** on the IBM Software Support site at http://www.ibm.com/software/support/probsub.html. Type your information into the appropriate problem submission form.

By phone
: For the phone number to call in your country, go to the contacts page of the *IBM Software Support Handbook* at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents....

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace **$***variable* with **%***variable***%** for environment variables and replace each forward slash (**/**) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. Getting started

Netcool/Impact is a set of runnable server components that work together to provide event management and integration functionality.

Software system

From a software perspective, you can best understand Netcool/Impact as a set of interrelated, runnable server components, each of which must be installed and configured separately.

Impact Server
The Impact Server is responsible for managing the data model, running services, and policies that make up your Netcool/Impact implementation, and runs the policies in real time in response to events that occur in your environment.

GUI Server
The GUI Server is responsible for generating the dynamic Web-based user interface of Netcool/Impact.

For information about installing, configuring, and running these components, see the *Administration Guide*.

Development tool

From an implementation perspective, you can understand Netcool/Impact as a development tool that you use to customize, enhance, and expand the functionality of an existing Netcool installation.

Netcool/Impact is not an event management or integration solution that is ready for immediate use. Rather, it is a platform that you can use to build new functionality into your current installation of the Netcool/Impact product suite.

Because it is a development tool, most of the work that you do with Netcool/Impact is done during initial setup. You need to understand your implementation goals and to plan your implementation before you begin. Some of the development tasks are modelling data, configuring services, and writing policies.

Modeling data

Modeling data is the task through which you create an abstract representation of the business data and metadata that you want to use with Netcool/Impact.

This task requires you to be able to identify and locate physical sources of data in your environment, and to create a representation of this data.

Configuring services

Configuring services is the development task in which you set up the runnable subcomponents of the Impact Server to perform such operations as monitoring an ObjectServer for events, or triggering the execution of a policy at timed intervals.

Writing policies

Writing policies is the task in which you define the operations that you want to automate with Netcool/Impact.

You write policies using the Impact Policy Language (IPL) or JavaScript and then configure Netcool/Impact to run them when certain conditions occur in your environment.

Automation engine

From a real-time operations perspective, you can understand Netcool/Impact as an automation engine that runs invisibly in the background and does not require end-user interaction.

This means that once you set up Netcool/Impact, it does not require any additional management unless you want to change your implementation.

Uses

You can use Netcool/Impact in a wide variety of ways, depending on the needs of your environment.

One way to determine what you can do with Netcool/Impact is to analyze its core features and see how you can put together a solution that enhances the value of your Netcool installation. Another way is to look at typical solutions implemented by other Netcool users and see how you can adapt them for use in your environment. Finally, you can also examine the workflow in your network environment and see what parts of the process can be improved using Netcool/Impact.

Core features

One way to determine what you can do with Netcool/Impact is to analyze its core features and see how you can put them to use in your environment.

Some of the most important features of Netcool/Impact are automation, event source access, data access, integration with other applications, and predefined actions.

Automation

Automation is the act of setting up a task so that it is performed automatically at certain times or when certain conditions are met.

For example, you can set a wrist watch to beep once per hour, or once per day at a certain hour. You might also be able to set a more advanced wrist watch to monitor your calendar and beep at different times of the day to notify you when meetings or other appointments are about to occur. In this case, the benefit of automation is that it saves you the time and effort involved in performing a routine task over and over. In the wrist watch example, you are saved the time and effort of constantly looking at your watch to check the time, or having to continually refer to a calendar to see when your next meeting begins.

A more complicated example of automation is the factory assembly line. As items move down the assembly line, operations must be performed on them in order for

the products to be finished. In some situations, these operations are performed by human workers, who each complete a designated action and pass the items down the line to the next person.

This example might be effective for a small factory making simple objects with few users. However, the larger the number of workers you have and the more complicated the operations, the higher the risk of error is. In addition, the more items you need to produce, the faster the operations must be accomplished. At some point, the human workers cannot work fast or well enough.

In the assembly line, automation comes in where you want to take some of the routine, repeatable parts of the process and set up them up so that they are performed automatically with a minimum of human intervention. For example, you might add various clocks, machines and monitors to the process, or even intelligent robots that can perform some operations previously accomplished by human workers.

A Netcool operations environment is, in some ways, like a factory assembly line. Instead of items moving down a conveyer belt, however, you have Netcool alerts. In a purely manual environment, network operators process each alert by hand. As they appear in the event list, the operators must acknowledge them, investigate them, notify other personnel that they have occurred and perform various other tasks like creating new trouble tickets or work orders in other applications. Finally, when the alert condition is resolved, the operator must delete the alerts out of the event list.

For a small network of a few devices and one or two operators, manual processing of events might be satisfactory. As with the assembly line, however, the more alerts you have and the more complicated the operator response, the more difficult, and costly it is to manage the operation on a manual basis.

Like the assembly line, the network operations environment can benefit from automating certain processes. For example, tasks like notifying technicians or system administrators when an alert reaches a certain severity, or updating an event to include related business data can require time and effort on the part of one or more network operators. If you automate these processes, the operators can spend time on higher priority tasks or be redeployed to other roles in the business. This example translates into an event management environment that is less costly than a manual one and less prone to human error.

Using Netcool/Impact you can automate event management tasks that you would otherwise have to accomplish manually, or would not be able to accomplish at all. The framework provided for automations consists of the policy engine, the policy scripting language and various triggers that you can use to trigger the policies under different conditions. The policy engine perform the tasks you want to automate. These tasks are specified in the (IPL) or JavaScript policy scripting languages, which are programming language similar in syntax to C/C++ and Java.

Event access

Event access is the feature that allows Netcool/Impact to tap into the event stream that flows from Netcool probes and monitors into the Netcool/OMNIbus ObjectServer.

This feature is an essential part of most implementations of the product. To see how and why event access is important, you must first understand the Netcool event stream and how it is generated.

The Netcool event stream is the flow of alerts from Netcool probes and monitors into the ObjectServer. Each alert represents some status or activity on the network and can originate from any of hundreds of different kinds of systems, devices, or applications. Typically, alerts are designed to inform network operators that a fault condition has occurred somewhere in the environment. They can be used to communicate other types of status or event information as well.

Alerts themselves are generated by software components called probes and monitors, which watch the systems, devices or applications and generate the alerts when various conditions occur. After the probes and monitors generate the alerts, they are sent to the ObjectServer, where they can be viewed by network operators through the Netcool/OMNIbus event list.

As noted in the previous section, the network management environment benefits from automated event management tasks. This automation relies on the ability to tap into the Netcool event stream.

The framework provided for event access consists of three features: the event reader, the event listener, and the event processor. These are runnable services that you control from within Netcool/Impact.

The OMNIbus event reader works by monitoring an instance of the ObjectServer. When it discovers new or updated events, or discovers that an event has been deleted, it takes the event and pulls it back into Netcool/Impact for processing. Similarly, the event listener monitors other, non-ObjectServer sources of events, such as SQL databases. As with the event reader, it takes new or updated events when it discovers them and pulls them back to Netcool/Impact. The event processor is responsible for what happens after the events have been retrieved.

When you set up an event reader or event listener, you define one or more policies that are to be executed when an event matches a specified criteria. A policy is a set of instructions for Netcool/Impact that specifies the tasks that you want to automate. When coupled with the event reader or event listener, you can use a policy to specify a set of tasks that you want Netcool/Impact to perform automatically when certain kinds of alerts appear in the ObjectServer or other events appear in other event sources.

This combination of event access and automation functionality allows you to create a wide variety of event management solutions. These solutions include event enrichment, X events in Y time, event notification, and event gateways.

Data access

Data access provides connection to external data sources, such as SQL databases and LDAP servers.

To understand how this feature is important, you can think about all the ways you use data that is external to Netcool/OMNIbus in your network management environment. You might have one database that contains network inventory information and another database that contains information about billing and customer service. In addition, you might have an LDAP directory that you use to store information about the company personnel.

In an environment without Netcool/Impact, it is possible for network operators to be responsible for manually retrieving data from one or more of these data sources and using that information to deduce the importance of events or to decide how events must be handled. With Netcool/Impact, you can integrate this type of data

directly into alerts at the ObjectServer level or use this data to perform various types of analysis on the severity or relevance of individual alerts.

The mechanism for accessing data consists of data source adaptors (DSAs) and data models. At the software component level, DSAs provide the means to connect to a wide variety of SQL databases, LDAP servers, and other data sources. At the solution level, the data model is an abstract representation of the data in your environment. The data model is used within policies when to retrieve, add, update, and delete data from the data sources.

Third-party integration

In many network operations environments, you have systems in place that have been created by more than one software provider.

For example, in addition to the Netcool suite of products, you might have separate third-party applications for network inventory management, billing, problem tracking, customer service, and help desk. You might also have messaging systems and other infrastructure software that is not provided by IBM.

Netcool/Impact provides interfaces to a wide array of such third-party applications, including GE Smallworld, and Portal Infranet. Netcool/Impact can interface with a wide variety of other applications by interfacing directly with their underlying databases.

Predefined actions

Netcool/Impact provides a built-in set of predefined actions that you can include among your automated tasks.

One of the most important of these tasks is the embedded e-mail client software. You can use this software to send e-mail notifications to administrators and users when faults or other conditions on your network occur. Another important predefined action is the ability to run shell commands, scripts, and applications on local or remote systems.

Typical implementations

Another way to determine what you can do with Netcool/Impact is to look at typical solutions implemented by other Netcool users and see how you can adapt them for use in your environment.

The following sections discuss some of the most typical implementations. Other parts of this guide go into greater detail about these solutions.

Some typical Netcool/Impact solutions are event enrichment, X events in Y time, event notification, and event gateways.

Event enrichment

Event enrichment is the process by which Netcool/Impact monitors an event source for new events, looks up information related to them in an external data source and then adds the information to them.

Event enrichment is the one of the most common, and valuable things that users achieve with Netcool/Impact.

To understand the value of event enrichment, you must first understand how Netcool probes work and some of their intrinsic limitations.

Netcool probes are runnable software components that you install on the devices that they monitor. Of probes and monitors, probes are the most common means for generating alerts in the Netcool event stream. Each probe has a rules file that specifies how the alert data is formatted and sent to the ObjectServer when certain activities or status levels on the device occur.

One characteristic of the probe rules file is that it is (essentially) static. This means that, once you install and configure the probe, the contents of the rules file rarely change. Not only is change rare, but changing the rules file on the fly to adjust to the constantly changing parameters of a network environment would be impossible. As a result, it is difficult to include dynamic information about the network in the contents of alerts generated by probes.

Another characteristic of the probes rules file is that, generally, it is best to use an identical copy of the file on every instance of a device that you have in your inventory. For example, if you have dozens of routers in your network, it is most likely that you want to use the same rules file for each device. This ensures that every probe reports activity or status information to the ObjectServer in the same way and eliminates any complications that might occur if each probe is configured differently. Because of this, however, the probes are not able to send alerts to the ObjectServer that contain information specific to the individual device, beyond a few basic parameters like the host name and IP address.

In addition to these limitations, the scope of alert data provided by a probe is generally restricted to information that directly describes the alert condition. The probe cannot provide additional information about how the condition affects the network as a whole, or perform any sort of analysis or correlation regarding the alert.

Although these limitations are primarily associated with probes, they exist to some degree with other Netcool components that generate alert data.

Event enrichment allows you to bypass these and other limitations by combining the event and data access features of Netcool/Impact. In an event enrichment scenario, Netcool/Impact "catches" new and updated alerts as they are sent to the ObjectServer, and then goes to one or more external data sources to correlate information in the alerts with business data. The Netcool/Impact policy language provides the means to intelligently determine which data in your environment is related to the alert and then to add that information to the alert on the fly.

The process of event enrichment can be configured to run completely in the background, so that the intervention of Netcool/Impact in the event flow is not noticeable to a network operator.

One simple example of event enrichment is an environment where you are managing a network of servers, each of which is used by a different department in the business. In this environment, you use the ping probe to monitor the uptime of the server systems. If a ping does not reach the target system, the probe sends an alert to the ObjectServer that says that the server is not reachable.

In an environment without Netcool/Impact, network operators would have to manually look up the business department associated with the server in order to deduce the priority of any incoming alert. They might also have to use a separate calendar or scheduling program to find the on-call administrator responsible for maintaining the system. With Netcool/Impact, you can "catch" each alert as it

comes into the ObjectServer, look up the affected business department, and automatically adjust the severity of the alert accordingly.

In environments with a higher level of complexity and many network devices, systems, and applications, the need for event enrichment becomes more critical. This automated process can then be used to supplement Netcool alerts with a wide variety of topological, technical, contact, and other information.

X events in Y time

X events in Y time is the process in which Netcool/Impact monitors an event source for groups of events that occur together and takes the appropriate action based on the event information.

X events in Y time solutions acknowledge the fact that few fault conditions in a network environment occur in a vacuum. When one device fails, for example, it is often possible that other parts of the system will also fail, or that the probe that monitors it will continue to report faults from the device until the problem is resolved. X events in Y time solutions allow you to "program" Netcool/Impact to take action when a group of related events occurs within the same time window.

An example of an X events in Y time solution is an environment where you are monitoring a set of telecommunication switches. A potential fault condition exists in the environment where a device will cause an alert to appear multiple times in the same time window as a particular link goes up and down. Taken alone, the fault is a low priority, but if it occurs more than a dozen or so times within the same 5 second period, it indicates a continuing problem that needs to be addressed.

Without Netcool/Impact, a network operator might not be able to detect this fault condition by simply monitoring the ObjectServer event list, especially if there are many other devices in the network reporting other status and fault conditions. With Netcool/Impact, you can define a set of operations that you want to take place automatically every time this event happens, including increasing the severity of the alert in order to make sure that it is detected by network operators.

Event notification

Event notification is the process by which Netcool/Impact monitors an event source for new events and then notifies an administrator or users when a certain event or combination of events occurs.

A built-in e-mail client is provided that allows you to send mail through any SMTP server. You can use this feature to send mail notifications to administrators and other users, or you can use the remote execution feature provided by the JRExec server to launch an external e-mail program from the command line. You can also use the JRExec server to send notifications through a paging system that provides a command-line interface.

Event notification is often part of a more complicated event management automation.

Event gateways

An event gateway is an implementation of Netcool/Impact in which you send event information from the ObjectServer to a third-party application for processing.

Because Netcool/Impact can interface with so many different types of databases and other software, you can build event gateways that do not otherwise exist as part of the Netcool suite.

Workflow analysis

You can determine what you to do with by analyzing the current workflow in your environment and seeing which parts of your event management process it is most effective to automate.

Before you analyze the workflow, review the core Netcool/Impact features, and other concepts and techniques.

Chapter 2. Solutions

A solution is an implementation of Netcool/Impact that provides a specific type of event management functionality.

Solution components

The components of a solution are a data model, services, and policies.

Most solutions use a combination of these three components.

Data models

A data model is a model of the business and metadata used in a Netcool/Impact solution.

A data model consists of data sources, data types, data items, links, and event sources.

Working with services

Services are runnable components of the Impact Server that you start and stop using both the GUI and the CLI.

Policies

A policy is a set of operations that you want Netcool/Impact to perform.

These operations are specified using a one of the following programming languages, JavaScript or a language called the Netcool/Impact policy language, or IPL.

Solution types

Netcool/Impact allows you to implement a wide variety of solution types. Some common types are event enrichment, X events in Y time, event notification, and event gateways.

Event enrichment solution

Event enrichment is the process by which Netcool/Impact monitors an event source for new events, looks up information related to them in an external data source and then adds the information to them.

An event enrichment solution consists of the following components:

- A data model that represents the data you want to add to events
- An OMNIbus event reader service that monitors the event source
- One or more event enrichment policies that look up information related to the events and add the information to them

For a sample event enrichment solution, see Chapter 21, "Event enrichment tutorial," on page 149.

X events in Y time solution

X events in Y time is the process in which Netcool/Impact monitors an event source for groups of events that occur together and takes the appropriate action based on the event information.

An X events in Y time solution consists of the following components:

- A data model that contains internal data types used to store metadata for the solution
- An OMNIbus event reader service that monitors the event source
- The hibernation activator service, which wakes hibernating policies at timed intervals
- One or more policies that check the event source to see if a specified group of events is occurring and then take the appropriate action

Event notification solution

Event notification is the process by which Netcool/Impact monitors an event source for new events and then notifies an administrator or users when a certain event or combination of events occurs.

Event notification is often part of a more complicated event management automation that includes aspects of Netcool/Impact functionality.

An event notification solution has the following components:

- An event reader service that monitors the event source
- An e-mail sender service that sends e-mail to administrators or users, or the JRExec server used to launch an external notification program
- One or more policies that perform the event notification

Event gateway solution

An event gateway is an implementation of Netcool/Impact in which you send event information from the ObjectServer to a third-party application for processing.

An event gateway solution has the following components

- A data model that includes a data source and data type representing the third-party application
- An OMNIbus event reader service that monitors the event source
- One or more policies that send event information to the third-party application

Setting up a solution

To set up a Netcool/Impact solution, you create a data model, set up services, and create policies.

For more information, see “Setting up a solution.”

Creating a data model

While it is possible to design a solution that does not require a data model, almost all uses of Netcool/Impact require the ability to handle internal or external data of some sort.

To create a data model, you create a data source for each real world source of data that you want to use. Then, you create a data type for each structural element (for example, a database table) that contains the data you want to use.

Alternatively, you can create dynamic links between data types or static links between data items that make it easier to traverse the data programmatically from within a policy.

Setting up services

Different types of solutions require different sets of services, but most solutions require an OMNIbus event reader.

Solutions that use hibernations also require the hibernating policy activator. Solutions that receive, or send e-mail require an e-mail reader and the e-mail sender service.

The first category of services is built in services like the event processor and the command-line service manager. Netcool/Impact only allows you to have single instances of this type of service. The second category is services like the event reader and policy activator. You can create and configure multiple instances of this type of service.

Creating policies

You create policies in the GUI Server, that contains a policy editor, a syntax checker, and other tools you need to write, run, test, and debug your policies.

For more information, see Chapter 13, “Working with policies,” on page 91.

Running a solution

To start a solution, you start each of the service components.

Start the components in the following order:

- Hibernating policy activator, e-mail sender, and command execution manager.
- Event processor
- Event reader, event listener, e-mail reader, or policy activator

You can configure services to run automatically at startup, or you can start them manually using the Tivoli Integrated Portal GUI and CLI. By default, services that run automatically at startup run in the proper order. If all other services are already running, starting services like the event processor that trigger policies effectively starts the solution.

To stop a solution, you stop any services, like the event processor, that trigger your policies.

Chapter 3. Working with data models

You set up a data model once, when you first design your Netcool/Impact solution.

After that, you do not need to actively manage the data model unless you change the solution design. You can view, create, edit, and delete the components of a data model in the GUI Server.

Data model components

A data model is made up of components that represent real world sources of data and the actual data inside them.

Data sources
Data sources are elements of the data model that represent real world sources of data in your environment.

Data types
Data types are elements of the data model that represent sets of data stored in a data source.

Data items
Data items are elements of the data model that represent actual units of data stored in a data source.

Links Links are elements of the data model that define relationships between data types and data items.

Event sources
Event sources are special types of data sources. Each event source represents an application that stores and manages events.

Data sources

Data sources are elements of the data model that represent real world sources of data in your environment.

These sources of data include third-party SQL databases, LDAP directory servers, or other applications such as messaging systems and network inventory applications.

Data sources contain the information that you need to connect to the external data. You create a data source for each physical source of data that you want to use in your Impact solution. When you create an SQL database, LDAP, or Mediator data type, you associate it with the data source that you created. All associated data types are listed under the data source in the Data Sources and Types task pane. When you create a data type, you simply select the data source it must use.

Data types

Data types are elements of the data model that represent sets of data stored in a data source.

The structure of data types depends on the category of data source where it is stored. For example, if the data source is an SQL database, each data type

corresponds to a database table. If the data source is an LDAP server, each data type corresponds to a type of node in the LDAP hierarchy.

Data items

Data items are elements of the data model that represent actual units of data stored in a data source.

The structure of this unit of data depends on the category of the associated data source. For example, if the data source is an SQL database data type, each data item corresponds to a row in a database table. If the data source is an LDAP server, each data item corresponds to a node in the LDAP hierarchy.

Links

Links are elements of the data model that define relationships between data types and data items.

Static links define relationships between data items, and dynamic links define relationships between data types. Links are an optional component of the Netcool/Impact data model.

Setting up a data model

To set up a data model, you must first determine what data you need to use in your solution and where that data is stored. Then, you create a data source for each real world source of data and create a data type for each structural element that contains the data you need.

Procedure

1. Create data sources

   Identify the data you want to use and where it is stored. Then, you create one data source for each real world source of data. For example, if the data is stored in one MySQL database and one LDAP server, you must create one MySQL and one LDAP data source.

2. Create data types

   After you have set up the data sources, you create the required data types. You must create one data type for each database table (or other data element, depending on the data source) that contains data you want to use. For example, if the data is stored in two tables in an Oracle database, you must create one data type for each table.

3. Optional: Create data items

   For most data types, the best practice is to create data items using the native tools supplied by the data source. For example, if your data source is an Oracle database, you can add any required data to the database using the native Oracle tools. If the data source is the internal data repository, you must create data items using the GUI.

4. Optional: Create links

   After you create data types, you can define linking relationships between them using dynamic links. You can also define linking relationships between internal data items using static links. That makes it easier to traverse the data programmatically from within a policy. Use of links is optional.

5. Create event sources

Most process events are retrieved from a Netcool/OMNIbus ObjectServer. The ObjectServer is represented in the data model as an event source.

Data model architecture

This diagram shows the relationship between data sources, data types, and data items in a Netcool/Impact solution.

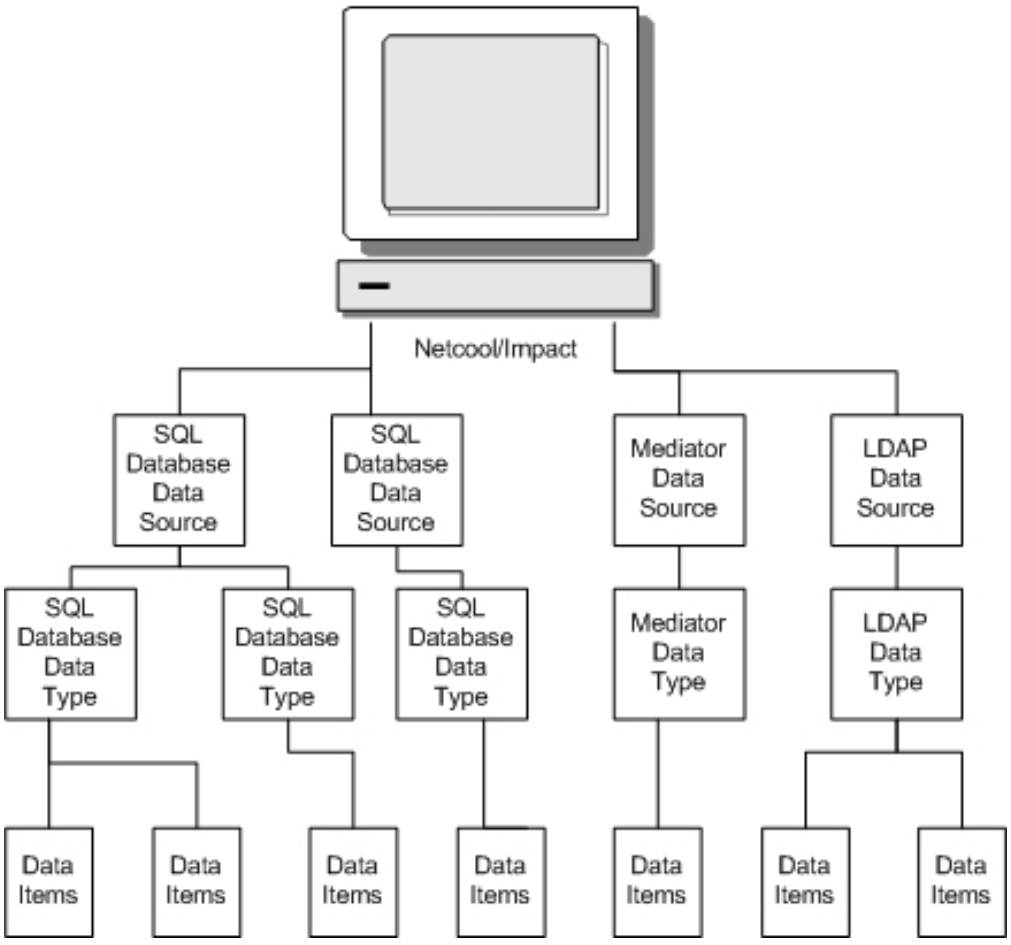

Figure 1. Data Model Architecture

Data model examples

The examples provided here are, most likely, scaled down versions of data models you might be required to implement in the real world.

They are designed to give you an idea of how all the different parts of a data model work together, rather than provide a realistic sampling of every type of data you might access with Netcool/Impact.

If you are uncertain about the definition of major concepts mentioned in these examples, such as data sources or data types, you can skip ahead to the next four chapters of this book, which provide detailed information about the various components of the data model. Once you have a better understanding of these concepts, you can return to this section.

Enterprise service model

The enterprise service model is a data model that is designed for use in an enterprise service environment.

The enterprise service environment is one of the most common network management scenarios for the Netcool product suite. While the data model described in this section is relatively simple, real world enterprise environments can often rival a small telecommunications or ISP environment in complexity.

The goal of the data model in this example is to provide the means to access a set of business data that has been previously collected and stored in an external database. This business data contains information about the users, departments, locations, and servers in the enterprise. If you were designing a complete solution for this environment, you would tap into this data model from within policies whenever you needed to access this data.

The enterprise service environment in this example consists of 125 users in five business departments, spread over three locations. Each user in the environment has a desktop computer and uses it to connect to a file server and an e-mail server.

The solution proposed to manage this environment is designed to monitor the file servers and e-mail servers for uptime. When a file server goes down, it notifies the on-call administrator through e-mail with a service request message. It also determines which business units are served by the file server and sends an e-mail to each user in the unit with a service interruption message. When an e-mail server goes down, it notifies the on-call administrator through pager.

All the data used by this solution is stored in a MySQL database. This database has six tables, named `USER`, `ADMIN`, `DEPT`, `LOC`, `FILESERVER`, and `EMAILSERVER`.

Enterprise service model elements

The enterprise service model consists of data sources, data types, data items, links, and event sources.

Data sources

Because all the data needed is stored in a single MySQL database, this data model only requires one data source. For the purposes of this example, the data source is named `MYSQL_01`.

Data types

Each table in the MYSQL database is represented by a single SQL database data type. For the purposes of this example, the data types are named `User`, `Admin`, `Department`, `Location`, `Fileserver`, and `Emailserver`. In this case, the names of the data types are the same as the table names.

Data items

Because the data is stored in an SQL database, the data items in the model are rows in the corresponding database tables.

Links The relationship between the data types in this data model can be described as a set of the following dynamic links:

- User -> Department
- User -> Location
- Location -> Emailserver
- Department -> Fileserver
- Emailserver -> Location.

- Fileserver -> Departments
- Administrator -> Location

Event sources

This data model has a single event source, which represents the Netcool/OMNIbus ObjectServer that stores events related to activity in their environment.

Web hosting model

The Web hosting model is a data model designed for use in a Web hosting environment.

The Web hosting environment is another common network management scenario for the Netcool product suite. Managing a Web hosting environment presents some unique challenges. This is because it requires you to assure the uptime of services, such as the availability of customer Web sites, that consist of groups of interrelated software and hardware devices, in addition to assuring the uptime of the devices themselves. As with the other examples in this chapter, the web services hosting environment described here is scaled down from what you might encounter in the real world.

The goal of the data model in this example is to provide the means to access a set of device inventory and service management data that is generated and updated in real time by a set of third-party application. This data contains information about the server hardware located in racks in the hosting facility and various other data that describes how instances of HTTP and e-mail server software is installed and configured on the hardware. As with the previous example, policies developed for use with this information would tap into this data model whenever they needed to access this data.

The Web services hosting model in this example consists of 10 HTTP server clusters and three e-mail servers clusters, spread over 20 machines. Each HTTP cluster and each e-mail cluster consist of one primary and one backup server. This environment serves 15 customers whose use is distributed across one or more clusters depending on their service agreement.

The solution that manages this environment is designed to monitor the uptime of the HTTP and e-mail services. When a problem occurs with one of these services, it determines the identity of the cluster that is causing the problem and the hardware where the component server instances are installed. It then modifies the original alert data in Netcool/OMNIbus to reflect this information. This solution also determines the customer that is associated with the service failure and sets the priority of the alert to reflect the customer's service agreement.

The data in this model is stored in two separate Oracle databases. The first database has five tables named `Node`, `HTTPInstance`, `HTTPCluster`, `EmailInstance`, and `EmailCluster`. The second database is a customer service database that has, among other tables, one named `Customer`.

Web hosting model elements

The Web hosting model consists of data sources, data types, data items, and links.

Data sources

Because this model has two real world sources of data, it requires two data sources. For this example, these sources are called `ORACLE_01` and `ORACLE_02`.

Data types

Each table in the MySQL database is represented by a single SQL database data type. For the purposes of this example, the data types are named `Node`, `HTTPInstance`, `HTTPCluster`, `EmailInstance`, `EmailCluster`, and `Customer`.

Data items

Because the data is stored in an SQL database, the data items in the model are rows in the corresponding database tables.

Links The relationship between the data types in this data model can be described as a set of the following dynamic links:

- HTTPServer -> Node
- EmailServer -> Node
- HTTPServer -> HTTPCluster
- EmailServer -> EmailCluster
- Customer -> HTTPCluster
- Customer -> HTTPServer

Chapter 4. Working with data sources

A data source is an element of the data model that represents a real world source of data in your environment.

Data sources overview

Data sources provide an abstract layer between Netcool/Impact and real world source of data.

Internally, data sources provide connection and other information that Netcool/Impact uses to access the data. When you create a data model, you must create one data source for every real world source of data you want to access in a policy.

The internal data repository of Netcool/Impact can also be used as a data source.

Data source categories

Netcool/Impact supports four categories of data sources.

SQL database data sources
An SQL database data source represents a relational database or another source of data that can be accessed using an SQL database DSA.

LDAP data sources
The LDAP data source represent LDAP directory servers.

Mediator data sources
Mediator data sources represent third-party applications that are integrated with Netcool/Impact through the DSA Mediator.

JMS data sources
A JMS data source abstracts the information that is required to connect to a JMS Implementation.

SQL database data sources

An SQL database data source represents a relational database or another source of data that can be accessed using an SQL database DSA.

Most commonly used commercial relational databases are supported, such as Oracle, Sybase, and Microsoft SQL Server. In addition, freely available databases like MySQL, and PostgreSQL are also supported. The Netcool/OMNIbus ObjectServer is also supported as a SQL data source.

The configuration properties for the data source specify connection information for the underlying source of data. Some examples of SQL database data sources are:

- A DB2 database
- A MySQL database
- An application that provides a generic ODBC interface
- A character-delimited text file

You create SQL database data sources using the GUI. You must create one such data source for each database that you want to access. When you create an SQL database data source, you need to specify such properties as the host name and port where the database server is running, and the name of the database. For the flat file DSA and other SQL database DSAs that do not connect to a database server, you must specify additional configuration properties.

Note that SQL database data sources are associated with databases rather than database servers. For example, an Oracle database server can host one or a dozen individual databases. Each SQL database data source can be associated with one and only one database.

LDAP data sources

The LDAP data source represent LDAP directory servers.

Netcool/Impact supports the OpenLDAP, and Microsoft Active Directory servers.

You create LDAP data sources in the GUI Server. You must create one such data source for each LDAP server that you want to access. The configuration properties for the data source specify connection information for the LDAP server, and any required security or authentication information.

Mediator data sources

Mediator data sources represent third-party applications that are integrated with Netcool/Impact through the DSA Mediator.

These data sources include a wide variety of network inventory, network provisioning, and messaging system software. In addition, providers of XML and SNMP data can also be used as mediator data sources.

Typically Mediator DSA data sources and their data types are installed when you install a Mediator DSA. The data sources are available for viewing and, if necessary, for creating or editing.

Attention: For a complete list of supported data source, see your IBM account manager.

Internal data repository

The internal data repository is a built-in data source for Netcool/Impact.

The primary responsibility of the internal data repository is to store system data.

Restriction: You must use internal data types solely for testing and demonstrating Netcool/Impact, or for low load tasks.

JMS data source

A JMS data source abstracts the information that is required to connect to a JMS Implementation.

This data source is used by the JMSMessageListener service, and the SendJMSMessage, and ReceiveJMSMessage functions.

Data source architecture

This diagram shows the relationship between Netcool/Impact, data sources, and the real world source of data in your environment.

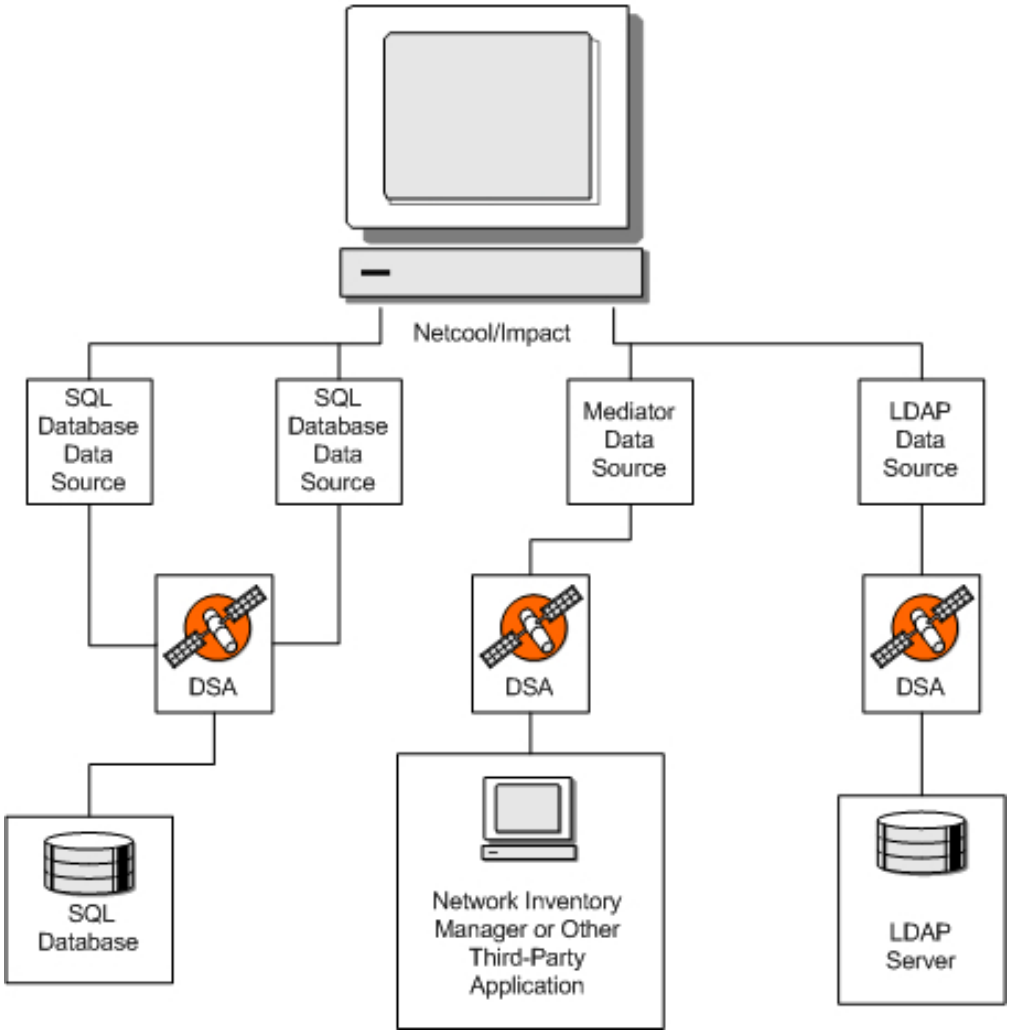

Figure 2. Data Source Architecture

Setting up data sources

When you create a Netcool/Impact data model, you must set up a data source for each real world source of data in your environment.

You set up data sources using the Tivoli Integrated Portal GUI. To set up a data source, you need to get the connection information for the data source, and then use the GUI to create and configure the data source.

Getting the connection information

Before you create an event source, you must get the connection information for the underlying application.

The connection information you need varies depending on the type of event source. For most SQL database data sources, this information is the host name and the port where the application is running, and a valid user name and password. For LDAP and Mediator data sources, see the *DSA Reference Guide* for the connection information required.

When you have the connection information for the underlying application, you can create the data source using the Tivoli Integrated Portal GUI.

Creating data sources

Use this procedure to create a user-defined data source.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. From the **Cluster** and **Project** lists, select the cluster and project you want to use.
3. In the **Data Model** tab, click the **New Data Source** icon in the toolbar. Select a template for the data source that you want to create. The tab for the data source opens.
4. Complete the required information, and click **Save** to create the data source.

Chapter 5. Working with data types

Data types are an element of the data model that represent sets of data stored in a data source.

Data types overview

Data types describe the content and structure of the data in the data source table and summarize this information so that it can be accessed during the execution of a policy.

Data types provide an abstract layer between Netcool/Impact and the associated set of data in a data source. Data types are used to locate the data you want to use in a policy. For each table or other data structure in your data source that contains information you want to use in a policy, you must create one data type. To use a data source in policies, you must create data types for it.

Attention: Some system data types are not displayed in the GUI. You can manage these data types by using the Command Line Interface (CLI).

The structure of the data that is stored in a data source depends on the category of the data source where the data is stored. For example, if the data source is an SQL database, each data type corresponds to a database table. If the data source is an LDAP server, each data type corresponds to a type of node in the LDAP hierarchy.

A data type definition contains the following information:

- The name of the underlying table or other structural element in the data source
- A list of fields that represent columns in the underlying table or another structural element (for example, a type of attribute in an LDAP node)
- Settings that define how Netcool/Impact caches data in the data type

Data type categories

Netcool/Impact supports four categories of data types.

SQL database data types
SQL database data types represent data stored in a database table.

LDAP data types
LDAP data types represent data stored at a certain base context level of an LDAP hierarchy.

Mediator data types
Mediator data types represent data that is managed by third-party applications such as a network inventory manager or a messaging service.

Internal data types
You use internal stored data types to model data that does not exist, or cannot be easily created, in external databases.

SQL database data types

SQL database data types represent data stored in a database table.

Each data item in an SQL database data type corresponds to a row in the table. Each field in the data item corresponds to a column. An SQL database data type can include all the columns in a table or just a subset of the columns.

LDAP data types

LDAP data types represent data stored at a certain base context level of an LDAP hierarchy.

Each data item in an LDAP data type corresponds to an LDAP node that exists at that level and each field corresponds to an LDAP attribute. LDAP data types are read-only, which means that you cannot add, update or delete data items in an LDAP data type.

Mediator data types

Mediator data types represent data that is managed by third-party applications such as a network inventory manager or a messaging service.

Typically, Mediator data types do not represent data stored in database tables. Rather, they represent collections of data that are stored and provided by the data source in various other formats. For example, sets of data objects or as messages.

These data types are typically created using scripts or other tools provided by the corresponding DSA. For more information about the mediator data types used with a particular DSA, see the *DSA Reference Guide*.

Internal data types

You use internal stored data types to model data that does not exist, or cannot be easily created, in external databases.

This includes working data used by policies, which can contain copies of external data or intermediate values of data. This data is stored directly in a data repository, and you can use it as a data source. To create and access this data you define internal data types.

Netcool/Impact provides the following categories of internal data types:

System data types
System data types are used to store and manage data used internally by Netcool/Impact.

Predefined internal data types
Pre-defined data types are special data types that are stored in the global repository.

User-defined internal data types
Internal data types that you create are user-defined internal data types.

Restriction: Use internal data types only for prototyping and demonstrating Netcool/Impact.

System data types

System data types are used to store and manage data used internally by Netcool/Impact.

These types include `Policy`, `Service`, and `Hibernation`. In most cases, you do not directly access the data in these data types. However, there are some occasions in

which you can use them in a policy. Some examples are when you start a policy from within another policy or work with hibernating policies.

Predefined internal data types

Pre-defined data types are special data types that are stored in the global repository.

The following predefined internal data types are provided:

- `Schedule`
- `TimeRangeGroup`
- `Document`

You use `Schedule` and `TimeRangeGroup` data types to manage Netcool/Impact scheduling. You can use the `Document` data type to store information about URLs located on your intranet.

Predefined data types are special data types that are stored in Netcool/Impact. The non-editable pre-defined data types are:

- TimeRangeGroup
- LinkType
- Hibernation

The following predefined data types can be edited to add new fields:

- Schedule
- Document
- FailedEvent
- ITNM

Restriction: You cannot edit or delete existing fields. None of the pre-defined data types can be deleted.

User-defined internal data types

Internal data types that you create are user-defined internal data types.

The data items in these data types are stored in the internal data repository, rather than in an external data source. User-defined data types function in much the same way as SQL database data types. You must use internal data types solely for testing and demonstrating Netcool/Impact, or for low load tasks. User-defined internal data types are slower than external SQL database data types.

Data type fields

A field is a unit of data as defined within a data type. The nature of this unit of data depends on the category of the data type that contains it.

If the data type corresponds to a table in an SQL database, each field corresponds to a table column. If the data type corresponds to a base context in an LDAP server, each field corresponds to a type of LDAP attribute.

When you set up an SQL database data type, the fields are auto-populated from the underlying table by Netcool/Impact. For other data types, you must manually define the fields when the data type is created.

ID

The ID attribute specifies the internal name used by Netcool/Impact to refer to the field.

By default, the field ID is the same as the name of the data element that corresponds to the field in the underlying data source. For example, if the data type is an SQL database data type, the underlying field corresponds to a column in the table. By default, the field ID is the same as the column name in the database.

You can change the field ID to any other unique name. For example, if the underlying column names in the data source are not human-readable, or are difficult to type and remember, you can use the ID field to provide a more easy-to-use alias for the field.

The field ID overrides the actual name and display name attributes for the field in all cases.

Field name

The field name attribute is the name of the corresponding data element in the underlying data source.

Although the Tivoli Integrated Portal GUI allows you to freely edit this field, it must be identical to how it appears in the data source in order to work. Otherwise, an error when trying to access the data type will be reported.

Format

The format is the data format of the field.

For SQL database data types, Netcool/Impact auto-discovers the columns in the underlying table and automatically deduces the data format for each field when you set up the data type. For other data types, you must manually specify the format for each field that you create.

Table 1 shows the supported data formats:

Table 1. Supported data formats

Format	Description
STRING	Represents text strings up to 4 KB in length.
INTEGER	Represents whole numbers.
LONG	Represents long whole numbers.
FLOAT	Represents floating point decimal numbers.
DOUBLE	Represents double-precision floating point decimal numbers.
DATE	Represents formatted date/time strings.
BOOLEAN	Represents Boolean values of true and false.
CLOB	Represents large-format binary data.
LONG_STRING	Represents text strings up to 16 KB in length (internal data types only).
PASSWORD_STRING	Represents password strings (internal data types only). The password appears in the GUI as a string of asterisks, rather than the actual password text.

Display name

The display name attribute allows you to specify a label for the field that is displayed only when you browse data items in the GUI. This attribute does not otherwise affect the functionality of the data type.

You can use this field to select a field from the menu to label data items according to the field value. Choose a field that contains a unique value that can be used to identify the data item for example, **ID**. To view the values on the data item you need to go to **View Data Items** for the data type and select the **Links** icon. Click the data item to display the details.

Description

The description attribute allows you to specify a short description for the field.

This description is only visible when you edit the data type using the GUI. Like the display name, it does not otherwise affect the functionality of the data type.

Data type keys

Key fields are fields whose value or combination of values can be used to identify unique data items in a data type.

For SQL database data types, you must specify at least one key field for each data type you create. Most often, the key field that you specify is a key field in the underlying data source. Internal data items contain a default field named `KEY` that is automatically used as the data type key.

You can use the policy function called `GetByKey` to retrieve data from the data type using the key field value as a query condition. Keys are also used when you create `GetByKey` dynamic links between data types.

Setting up data types

When you create a data model, you must set up a data type for each structural element in a data source whose data you want to use.

For example, if you are using an SQL database data source, you must set up a data type for each table that contains the data. If you are using an LDAP data source, you must set up a data type for each base context in the LDAP hierarchy that contains nodes that you want to access. You set up data types using the Tivoli Integrated Portal GUI.

To set up a data type, you get the name of the structural element (for example, the table) where the data is located, and then use the GUI to create and configure the data type.

Getting the name of the structural element

If the data type is an SQL database data type, you must know the fully qualified name of the underlying table in the database before you can set it up.

This name consists of the database name and the table name. Some databases use case-sensitive table names, so make sure that you record the proper case when you

get this information. If the data type is an LDAP data type, you must know the name of the base context level of the LDAP hierarchy where the nodes you want to access are located.

Configuring internal data types

This procedure uses an Administrator internal data type as an example.

About this task

To define the data type for Administrator, you specify the attributes (fields) that you want listed for every administrator, perhaps a name, a pager number, and an e-mail address. Then you create data items: the names, pager numbers, and e-mail addresses of the administrators. For internal data types, these attributes are the actual data items for the data type.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** > **Data Model**, to open the **Data Model** tab.

   Since internal data is stored in Netcool/Impact, it is not necessary to first configure a data source connection.
2. Select the data source that you want to create a data type for, right-click the data source and click **New Data Type** .
3. Enter the information in the **Custom Fields** tab **General Settings** section.

   Click **Save**.
4. To add additional fields to the data type:
   a. In the **Additional Fields** section of the tab, click the **New** button.
   b. Enter the information in the window.
   c. Continue to add fields to the table as appropriate.
   d. From the **Display Name Field** list situated under the **Additional Fields** table, you can select a field name that you want to use to name a data item elsewhere in the GUI.
   e. When you are finished, click **Save** in the editor toolbar.
5. In the Dynamic Links tab configure dynamic links. For information about dynamic links tab, see the section on“Links” on page 14.

Internal data type configuration window

Use this information to configure an internal data type.

Table 2. New Internal Data Type Editor Custom Fields tab

Editor element	Description
General settings	
Data Type Name	Type a unique name to identify the data type. Only letters, numbers, and the underscore character must be used in the data type name. If you use UTF-8 characters, make sure that the locale on the Impact Server where the data type is saved is set to the UTF-8 character encoding. If you receive an error message when saving a data type, check the **Global** tab for a complete list of data type names for the server. If you find the name you have tried to save, you need to change it.

Table 2. New Internal Data Type Editor Custom Fields tab (continued)

Editor element	Description
State: Persistent	Leave the box checked as Persistent (permanent) to permanently store the data items created for this data type. When the server is restarted, the data is loaded. If the box is cleared, the data is held in memory, but only while the server is running. When the server restarts, the data is lost because it was not backed up in a file. This feature is useful if you need data only on a temporary basis and then want to discard it. Persistent data types are always written to file. Therefore making internal data types temporary is faster.
New Field	Click to add a new field to the table.
Additional fields (New Field window)	
ID	Type a unique ID for the field.
Field Name	Type the actual field name. This can be the same as the ID. You can reference both the **ID** field and the **Field Name** field in policies. If you do not enter a Display Name (see below), Netcool/Impact uses the ID field name by default.
Format	Select a format for the field from the **Format** list:
Display Name Field:	You can use this field to select a field from the menu to label data items according to the field value. Choose a field that contains a unique value that can be used to identify the data item for example, **ID**. To view the values on the data item you need to go to **View Data Items** for the data type and select the **Links** icon. Click the data item to display the details.
Description	Type some text that describes the field.

SQL data types

SQL data types define real-time dynamic access to data in tables in a specified SQL database.

When the database is accessed, the fields from the database schema are assigned to the data type. Some of the SQL data sources automatically discover the fields in the table. Others do not support automatic table discovery; for these data sources, you must enter the table name to see the names of the fields.

The editor contains three tabs.

Table 3. External data type editor tabs

Tab	Description
Table Description	Name the data type, change the data source, if necessary, and add any number of fields from the data source to form a database table.
Dynamic Links	In this tab you can create links to other data types, both external and internal, to establish connections between information. Links between individual data items can represent any relationship between the items that policies must be able to look up. For example, a node linked to an operator allows a policy to look up the operator responsible for the node. For more information about dynamic links tab, see “Links” on page 14.

Table 3. External data type editor tabs (continued)

Tab	Description
Cache Settings	In this tab, you can set up caching parameters to regulate the flow of data between Netcool/Impact and the external data source. Use the guidelines in "SQL data type configuration window - Cache settings tab" on page 34, plus the parameters for the performance report for the data type to configure data and query caching.

Important: SQL data types in Netcool/Impact require all columns in a database table to have the Select permission enabled to allow discovery and to enable the save option when creating data types.

Configuring SQL data types

Use this procedure to configure an SQL data type.

Procedure

- Provide a unique name for the data type.
- Specify the name of the underlying data source for the data type.
- Specify the name of the database and the table where the underlying data is stored.
- Auto-populate the fields in the data type.
- Select a display name for the data type.
- Specify key fields for the data type.
- Optional: Specify a data item filter.
- Optional: Specify which field in the data type to use to order data items.
- Optional: Specify the direction to use when ordering data items.

What to do next

After you have saved the data type, you can close the Data Type Editor or you can configure caching and dynamic links for the data type.

SQL data type configuration window - Table Description tab

Use this information to configure the SQL data type.

Table 4. New External Data Type editor - Table Description tab

Editor element	Description
General Settings	
Data Type Name	Type a unique name to identify the data type. Only letters, numbers, and the underscore character must be used in the data type name. If you use UTF-8 characters, make sure that the locale on the Impact Server where the data type is saved is set to the UTF-8 character encoding. Data type names must be unique globally, not just within a project. If you receive an error message when saving a data type, check the **Global** project tab for a complete list of data type names for the server. If you find the name you have tried to save, you need to change it.

Table 4. New External Data Type editor - Table Description tab (continued)

Editor element	Description
Data Source: Name	This field is automatically populated, based on the data source you selected in the Data Sources and Types task pane. However, if you have other SQL data sources configured to use with Netcool/Impact, you can change the name to any of the SQL data sources in the list, if necessary. If you enter a new name, a message window prompts you to confirm your change. Click **OK** to confirm the change. If you change your mind about selecting a different data source, click **Cancel**.
State: Enabled	Leave the **State** check box checked to activate the data type so that it is available for use in policies.
Table Description	
Base Table	Specify the underlying database and table where the data in the data type is stored. The names of all the databases and tables are automatically retrieved from the data source so that you can choose them from a list. Type the name of the database and the table in the **Base Table** lists. The first list contains the databases in the data source. The second list contains the tables in the selected database, for example, alerts, and status.
Refresh	Click **Refresh** to populate the table. The table columns are displayed as fields in a table. To make database access as efficient as possible, delete any fields that are not used in policies.
Add Deleted Fields	If you have deleted fields from the data type that still exist in the SQL database, these fields do not show in the user interface. To restore the fields to the data type, mark the **Add Deleted Fields** check box and click **Refresh**.
New Field	Use this option if you need to add a new field to the table from the data source database. For example, in the case where the field was added to the database after you created the data type. Make sure that the field name you add has the same name as the field name in the data source. **Important:** Any new fields added to this table are not automatically added to the data source table. You cannot add fields to the database table in this way. For more information, see "SQL data type configuration window - adding and editing fields in the table" on page 32.

Table 4. New External Data Type editor - Table Description tab (continued)

Editor element	Description
Key field	Key fields are used when you retrieve data from the data type in a policy using the `GetByKey` function. They are also used when you define a `GetByKey` dynamic link. **Important:** You must define at least one key field for the data type, even if you do not plan to use the GetByKey functionality in your policy. If you do not, Netcool/Impact will not function properly. Generally, the key fields you define correspond to key fields in the underlying database table. To specify a key field, click the check box in the appropriate row in the **Key Field** column. You can add multiple key fields.
Display Name Field	You can use this field to select a field from the menu to label data items according to the field value. Choose a field that contains a unique value that can be used to identify the data item for example, **ID**. To view the values on the data item you need to go to **View Data Items** for the data type and select the **Links** icon. Click the data item to display the details.
Automatically Remove Deleted Fields	Mark the **Automatically Remove Deleted Fields** check box to remove any fields from the data type that have already been removed from the SQL database. This happens automatically when a policy that uses this data type is run.
Data Filtering and Ordering	

Filter	Type a restriction clause to limit the types of data items seen for the data type. For example, to limit the rows in a field called `City` to *New York*, you would enter:
	`City = "New York"`
	
	For example, to limit the rows to the *New York* or *Athens*, you would enter:
	`City = "New York" OR City = "Athens"`
	
	You can use any sql `Where` clause syntax.

Order By	Enter the names of one or more fields to use when sorting data items retrieved from the data source.

SQL data type configuration window - adding and editing fields in the table

Use this information to add or edit a field to the table for a SQL data type.

In the Table tab, in the **New Field** area, click **New** to add a field to the data type, or select the edit icon next to an existing field that you want to edit.

Table 5. External Data Type Editor - New Field window

Window element	Description
ID	By default, the **ID** is the same as the column name in the database. You can change it to any other unique name. For example, if the underlying column names in the data source are difficult to use, the ID field to provide an easier alias for the field.

Table 5. External Data Type Editor - New Field window (continued)

Window element	Description
Field Name	Type a name that can be used in policies. It represents the name in the SQL column. Type the name so that it is identical to how it appears in the data source; otherwise, Netcool/Impact reports an error when trying to access the data type.
Format	For SQL database data types, Netcool/Impact auto-discovers the columns in the underlying table and automatically detects the data format for each field when you set up the data type. For other data types, you must manually specify the format for each field that you create. For more information about formats, see the Working with Data Types chapter in the Solutions Guide. Select a format from the following list: • STRING • LONG_STRING • INTEGER • PASSWORD_STRING • LONG • FLOAT • DOUBLE • DATE • BOOLEAN • CLOB
Display Name	You can use this field to select a field from the menu to label data items according to the field value. Choose a field that contains a unique value that can be used to identify the data item for example, **ID**. To view the values on the data item you need to go to **View Data Items** for the data type and select the **Links** icon. Click the data item to display the details. If you do not enter a display name, Netcool/Impact uses the ID field name by default.
Description	Type some text that describes the field. This description is only visible when you edit the data type using the GUI.
Default Value	Type a default expression for the field. It can be any value of the specified format see the format row, or it can be a database-specific identifier such as an Oracle pseudonym; for example, `sequence.NEXTVAL`.

Table 5. External Data Type Editor - New Field window (continued)

Window element	Description
Insert Statements: Exclude this field	When you select the **Exclude this Field** check box Netcool/Impact does not set the value for the field when inserting and updating a new data item into the database. This field is used for insert and update statements only, not for select statements. Sybase data types: You must select this option when you map a field to an **Identity** field or a field with a default value in a Sybase database. Otherwise, Netcool/Impact overwrites the field on insert with the specified value or with a space character if no value is specified. ObjectServer data types: The **Tally** field automatically selects the **Exclude this Field** check box to be excluded from inserts and updates for the objectserver data type since this field is automatically set by Netcool® OMNIbus to control deduplication of events. The **Serial** field automatically selects the **Exclude this Field** check box to be excluded from inserts and updates when an ObjectServer data type points to `alerts.status`.
Type Checking: Strict	Click to enable strict type checking. When you enable strict type checking on the field, Netcool/Impact checks the format of the value of the field on insert or update to ensure that it is of the same format as the corresponding field in the data source. If it is not the same, Netcool/Impact does not perform the insert or update and a message to that effect is displayed in the server log. If you do not enable strict type checking, all type checking and format conversions are done at the data source level.

SQL data type configuration window - Cache settings tab

Use this information to configure caching for a SQL data type.

Table 6. External Data Type Cache Settings tab - caching types

Cache type	Description
Enable Data Caching	This check box toggles data caching on and off.
Maximum number of data items	Set the total number of data items to be stored in the cache during the execution of the policy.
Invalidate Cached Data Items After	Set to invalidate the cached items after the time periods selected.
Enable Query Caching	This check box toggles query caching on and off.
Maximum number of queries	Set the maximum number of database queries to be stored in the cache.
Invalidate Cached Queries After	Set to invalidate the cached items after the time periods selected.
Enable Count Cashing	Do not set. Available for compatibility with earlier versions only.
Performance Measurements Intervals	Use this option to set the reporting parameters for measuring how fast queries against a data type are executed.
Polling Interval	Select a polling interval for measuring performance statistics for the data type.

Table 6. External Data Type Cache Settings tab - caching types (continued)

Cache type	Description
Query Interval	Select the query interval for the performance check.

Auto-populating the data type fields

After you have specified the name of the database and table, the next step is to auto-populate the data type fields.

You can also specify the fields manually in the same way that you do for internal data types, but in most cases, using the auto-populate feature saves time and ensures that the field names are accurate.

When you auto-populate data type fields, the table description is retrieved from the underlying data source, and a field in the data type is created for each column in the table. The ID, actual name, and display name for the fields are defined using the exact column name as it appears in the table.

A set of built-in rules is used to determine the data format for each of the auto-populated fields. Columns in the database that contain text data, such as `varchar`, are represented as string fields. Columns that contain whole numbers, such as `int` and `integer`, are represented as integer fields. Columns that contain decimal numbers are represented as float fields. Generally, you can automatically assign the formats for data type fields without having to manually attempt to recreate the database data formats in the data type.

If you only want a subset of the fields in a table to be represented in the data type, you can manually remove the unwanted fields after auto-population. Removing unwanted fields can speed the performance of a data type.

To auto-populate data type fields, you click the **Refresh** button in the **Table Description** area of the Data Type tab. The table description is retrieved from the data source, and the fields are populated. The fields are displayed in the **Table Description** area.

After you auto-populate the data type fields, you can manually change the attributes of any field definition. Do not change the value of the actual name attribute. If you change this value, errors will be reported when you try to retrieve data from the data type.

Specifying a data item filter

The data item filter specifies which rows in the underlying database table can be accessed as data items in the data type.

This filter is an optional setting. The syntax for the data item filter is the same as the contents of the `WHERE` clause in the SQL `SELECT` statement supported by the underlying database.

For example, if you want to specify that only rows where the `Location` field is `New York` are accessible through this data type, you can use the following data item filter:

```
Location = 'New York'
```

If you want to specify that only rows where the `Location` is either `New York` or `New Jersey`, you can use the following:

```
Location = 'New York' OR Location = 'New Jersey'
```

Make sure that you enclose any strings in single quotation marks.

To specify the data item filter, type the filter string in the **Filter** text box in the **Data Item Filter and Ordering** area of the Data Type Editor.

Specifying data item ordering

Data item ordering defines the order in which data items are retrieved from the data type.

The order settings are used both when you retrieve data items using the `GetByFilter` function in a policy and when you browse data items using the GUI. You can order data items in ascending or descending alphanumeric order by any data type field. Data item ordering is an optional part of the data type configuration.

You specify data item ordering in the data type configuration as a comma-separated list of fields, where each field is accompanied with the `ASC` or `DESC` keyword.

For example, to retrieve data items in ascending order by the `Name` field, you use the following ordering string:

```
Name ASC
```

To retrieve data items in descending order first by the `Location` field and then in ascending order by `Name`, you use the following string:

```
Location DESC,Name ASC
```

To specify data item ordering:

1. In the Data Type Editor, scroll down so that the **Data Filtering and Ordering** area is visible.
2. Type the data item ordering string in the **Order By** field.

LDAP data types

An LDAP data type represents a set of entities in an LDAP directory tree.

The LDAP DSA determines which entities are part of this set in real time by dynamically searching the LDAP tree for those that match a specified LDAP filter within a certain scope. The DSA performs this search in relation to a location in the tree known as the base context.

The LDAP Data Type editor contains three tabs.

Table 7. LDAP Data Type editor tabs

Tab	Description
LDAP Info	In this tab, you configure the attributes of the data type. For more information about these attributes, see “LDAP data type configuration window - LDAP Info tab” on page 37.

Table 7. LDAP Data Type editor tabs (continued)

Tab	Description
Dynamic Links	In this tab you can create links to other data types, both external and internal, to establish connections between information. Links between individual data items can represent any relationship between the items that policies need to be able to look up. For example, a node linked to an operator allows a policy to look up the operator responsible for the node. For more information about creating links to other data types, see "Links" on page 14.
Cache Settings	In this tab, you can set up caching parameters to regulate the flow of data between Netcool/Impact and the external data source. For more information about, cache settings see "SQL data type configuration window - Cache settings tab" on page 34.

Important: You must create one LDAP data type for each set of entities that you want to access. The LDAP data type is a read-only data type which means that you cannot edit or delete LDAP data items from within the GUI.

Configuring LDAP data types

Use this procedure to configure an LDAP data type.

Procedure

- Provide a unique name for the data type.
- Specify the name of the underlying data source for the data type.
- Specify the base context level in the LDAP hierarchy where the elements you want to access are located.
- Specify a display name field.
- Optional: Specify a restriction filter.

LDAP data type configuration window - LDAP Info tab

Use this information to configure LDAP information for a LDAP data type.

Table 8. LDAP Data Type editor - LDAP Info Tab

Editor element	Description
General Settings	
Data Type Name	Type a unique name to identify the data type. Only letters, numbers, and the underscore character must be used in the data type name. If you use UTF-8 characters, make sure that the locale on the Impact Server where the data type is saved is set to the UTF-8 character encoding.
State: Enabled	Leave checked to enable the data type so that it can be used in policies.
LDAP Info	

Table 8. LDAP Data Type editor - LDAP Info Tab (continued)

Editor element	Description
Data Source Name	Type the name of the underlying data source. This field is automatically populated, based on your data source selection in the Data Types task pane of the Navigation panel. However, if you have more than one LDAP data source configured for use with Netcool/Impact, you can select any LDAP data source in the list, if necessary. If you enter a new name, a message window asks you to confirm your change.
Search scope	Select the search scope: • OBJECT_SCOPE • ONLEVEL_SCOPE • SUBTREE_SCOPE
Base Context	Type the base context that you want to be used when searching for LDAP entities. For example:`ou=people`, `o=companyname.com`.
Key Search Field	Type the name of a key field, for example, `dn`.
Display Name Field	You can use this field to select a field from the menu to label data items according to the field value. Choose a field that contains a unique value that can be used to identify the data item for example, **ID**. To view the values on the data item you need to go to **View Data Items** for the data type and select the **Links** icon. Click the data item to display the details.
Restriction Filter:	Optionally, type a restriction filter. The restriction filter is an LDAP search filter as defined in Internet RFC 2254. This filter consists of one or more Boolean expressions, with logical operators prefixed to the expression list. For more information, see the *LDAP Filter* information in the *Policy Reference Guide*.
Attribute Configuration	
New Field	For each field that you want to add to the data type, click **New**.

Mediator DSA data types

Mediator DSA data types are typically created using scripts or other tools provided by the corresponding DSA.

Usually the data types, and their associated data sources are installed when you install the Mediator DSA (Corba or Direct), so you do not have to create them. The installed data types are available for viewing and, if necessary, for editing.

For more information about the Mediator data types used with a particular DSA, see the DSA documentation.

Data type caching

You can use data type caching to reduce the total number of queries that are made against a data source for performance or other reasons.

Caching helps you to decrease the load on the external databases used by Netcool/Impact. Data caching also increases system performance by allowing you to temporarily store data items that have been retrieved from a data source.

Important: Caching works best for static data sources and for data sources where the data does not change often.

Caching works when data is retrieved during the processing of a policy. When you view data items in the GUI, cached data is retrieved rather than data directly from the data source.

You can specify caching for external data types to control the number of data items temporarily stored while policies are processing data. Many data items in the cache uses significant memory but can save bandwidth and time if the same data is referenced frequently.

Important: Data type caching works with SQL database and LDAP data types. Internal data types do not require data type caching.

You configure caching on a per data type basis within the GUI. If you do not specify caching for the data type, each data item is reloaded from the external data source every time it is accessed.

Configuring data caching

Use this procedure to configure data caching.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Expand the **Data Source** that contains the data type you want to edit.
3. Double click the data type or click the **Edit** icon on the toolbar to open the **Cache Settings** tab.
4. Select the **Enable Data Caching** check box.
5. Enter a number in the **Maximum Number of Data Items** field to set the maximum number of data items to cache.
6. Enter the amount of time to cache each data item in the**Invalidate Cached Items After** fields to set the expiration time for data items in the cache.

   Netcool/Impact calculates the expiration time separately for each data item in the cache.
7. Click **Save** to implement the changes to the data type.

Configuring query caching

Use this procedure to configure query caching.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Expand the **Data Source** that contains the data type you want to edit.
3. Double click the data type or click the **Edit** icon on the toolbar to open the **Cache Settings** tab.
4. Scroll down until the **Enable Query Caching** area is visible.
5. Select the **Enable Query Caching** check box.

6. Enter a number in the **Maximum Number of Data Items** field to set the maximum number of queries to cache.
7. Enter the amount of time to cache each data item in the**Invalidate Cached Items After** fields to set the expiration time for query results in the cache.

   The expiration time is calculated separately for each query in the cache.
8. Click **Save** to implement the changes to the data type.

Important: You must also enable data caching for query caching to work.

Count caching

Use this procedure to configure count caching.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Expand the **Data Source** that contains the data type you want to edit.
3. Double click the data type or click the **Edit** icon on the toolbar to open the **Cache Settings** tab.
4. Scroll down until the **Enable Count Caching** area is visible.
5. Select the **Enable Count Caching** check box.
6. Enter the amount of time to cache each data item in the**Invalidate Cached Items After** fields.

   You can configure the expiration time for items counted using this feature.
7. Click **Save** to implement the changes to the data type.

Chapter 6. Working with links

You set up links after you have created the data types required by your solution and after you have populated any internal data types in the model with information.

When you write policies, you use the GetByLinks function to traverse the links and retrieve data items that are linked to other data items.

Links overview

Links are an element of the data model that defines relationships between data items and between data types.

They can save time during the development of policies because they allow you to define a data relationship once and then reuse it several times when you need to find data related to other data in a policy. Links are an optional part of a data model. Dynamic links and static links are supported.

Link categories

Netcool/Impact provides two categories of links.

Static links
Static links define a relationship between data items in internal data types.

Dynamic links
Dynamic links define a relationship between data types.

Static links

Static links define a relationship between data items in internal data types.

Static links are supported for internal data types only. Static links are not supported for other categories of data types, such as SQL database and LDAP types, because the persistence of data items that are stored externally cannot be ensured.

A static link is manually created between two data items when relationships do not exist at the database level.

With static links, the relationship between data items is static and never changes after they have been created. You can traverse static links in a policy or in the user interface when you browse the linked data items. Static links are bi-directional.

Dynamic links

Dynamic links define a relationship between data types.

This relationship is specified when you create the link and is evaluated in real time when a call to the `GetByLinks` function is encountered in a policy. Dynamic links are supported for internal, SQL database and LDAP data types.

The relationships between data types are resolved dynamically at run time when you traverse the link in a policy or when you browse links between data items. They are dynamically created and maintained from the data in the database.

The links concept is similar to the JOIN function in an SQL database. For example, there might be a 'Table 1' containing customer information (name, phone number, address, and so on) with a unique Customer ID key. There may also be a 'Table 2' containing a list of servers. In this table, the Customer ID of the customer that owns the server is included. When these data items are kept in different databases, Netcool/Impact permits the creation of a link between Table 1 and Table 2 through the **Customer ID** field, so that you can see all the servers owned by a particular customer.

You can use dynamic links only at the database level. (When relationships do not exist at the database level, you needs to create static links.) You can create dynamic links for all types of data types (internal, external, and predefined). See “Data types” on page 13 for information about the kinds of data type.

Dynamic links are unidirectional links, configured from the source to the target data type.

Link by filter

A link by filter is a type of dynamic link where the relationship between two data types is specified using the link filter syntax. The link filter syntax is as follows:

```
target_field = %source_field% [AND (target_field = %source_field%) ...]
```

Where *target_field* is the name of a field in the target data type and *source_field* is the name of the field in the source data type. When you call the `GetByLinks` function in a policy, Netcool/Impact evaluates the data items in the target data type and returns those items whose *target_field* value is equal to the specified *source_field*.

If the value of *source_field* is a string, you must enclose it in single quotation marks.

The following examples show valid link filters:

```
Location = '%Name%'
(NodeID = %ID%) AND (Location = '%Name%')
```

Link by key

A link by key is a type of dynamic link where the relationship between two data types is specified by a foreign key expression.

The foreign key expression is the value that the key field in data items in the target data type must have in order to be considered linked to the source. The syntax of the foreign key expression is the name or names of fields in the source data type whose value must equal the key field in the target. You can concatenate fields using the addition (+) operator.

When you call the `GetByLinks` function in a policy, Netcool/Impact evaluates the data items in the target data type and returns those data items whose key field values match the specified key expression.

The following examples show valid key expressions:

```
LastName
FirstName + " " + LastName
LastName + ", " + FirstName
```

Link by policy

A link by policy is a type of dynamic link where the relationship between two data types is specified by a policy.

The policy contains the logic that is used to retrieve data items from the target data type. The linking policy specifies which data items to return by setting the value of the `DataItems` variable.

Setting up static links

Use this procedure to set up a static link.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model tab**.
2. Expand the **Data Source** that contains the internal data type you want to link.
3. Double click the data type or click the **Edit** icon on the toolbar to open the source data item you want to link.

   A Data Type Editor tab opens in the Main Work panel.
4. Click the **Links** button for the data item you want to link.
5. In the Static Links window that opens, select the data type that contains the data items you want to link to.
6. Select the data item to link to from the list of data items that appears.

Setting up dynamic links

You can set up a dynamic link by filter, by key, and by policy.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation**, and click **Data Model** to open the **Data Model tab**.
2. Expand the **Data Source** that contains the internal data type you want to link.
3. Double click the data type or click the **Edit** icon on the toolbar to open the data type you want to use as the source for the link.
4. In the Data Type Editor tab, select the Dynamic Links tab in the Data Type Editor.
5. Depending on the type of link you want to create, click **New Link By Filter**, **New Link By Key**, or **Link By Policy** button.

   This will bring up a new link editor window.

   Tip: To create a new link by policy, you may need to scroll down so that the **Link By Policy** area is visible.
6. Select the target data type from the **Target Data Types** list.
7. Select the exposed link type from the **Exposed Link Type** list.
8. Depending on the type of link you are creating, type in the filter, key expression, or select a policy.

- For a link by filter, type the filter syntax for the link in the **Filter into Target Data Type** field. For example: `Location = '%Facility%'`.
- For a link by key, type the key expression in the **Foreign Key Expression** field. For example: `FirstName + ' ' + LastName`.
- For a link by policy, select the linking policy from the **Policy To Execute to Find Links** list.

9. Click OK.

Chapter 7. Working with event sources

When you design your solution, you must create one event source for each application that you want to monitor for events, then you can create event reader services and associate them with the event source.

Typically, a solution uses a single event source. This event source is most often an ObjectServer database.

Event sources overview

An event source is a special type of data source that represents an application that stores and manages events, the most common such application being the ObjectServer database.

An event is a set of data that represents a status or an activity on a network. The structure and content of an event varies depending on the device, system, or application that generated the event but in most cases, events are Netcool/OMNIbus alerts.

The installer automatically creates a default ObjectServer event source, defaultobjectserver. This event source is configured using information you provide during the installation. You can also use other applications as non-ObjectServer event sources.

After you have set an event source, you do not need to actively manage it unless you change the solution design but, if necessary, you can use the GUI to modify or delete event sources.

ObjectServer event sources

The most common event source are ObjectServer event sources that represent instances of the Netcool/OMNIbus ObjectServer database.

ObjectServer events are alerts stored in the `alerts.status` table of the database. These alerts have a predefined set of alert fields that can be supplemented by additional fields that you define.

ObjectServer event sources are monitored using an OMNIbus event reader service. The event reader service queries the ObjectServer at intervals and retrieves any new, updated, or deleted events that matches its predefined filter conditions. The event reader then passes each event to the policy engine for processing.

Non-ObjectServer event sources

Non-ObjectServer event sources represent instances of other applications, such as external databases or messaging systems, that provide events to Netcool/Impact.

Non-ObjectServer events can take a wide variety of forms, depending on the nature of the event source. For SQL database event sources, an event might be the contents of a row in a table. For a messaging system event source, an event might be the contents of a message.

Non-ObjectServer event sources are monitored using an event listener service. The event listener service passively receives events from the event source and then passes them to the policy engine for processing.

The DatabaseEventReader service monitors Non-ObjectServer data sources. The Database Event Reader service queries the SQL data source at intervals and retrieves any new or updated events that match its predefined filter conditions. The Database Event Reader passes each event to the policy engine for processing.

Event source architecture

This diagram shows how event sources interact with event sources and event listeners with their underlying event management applications.

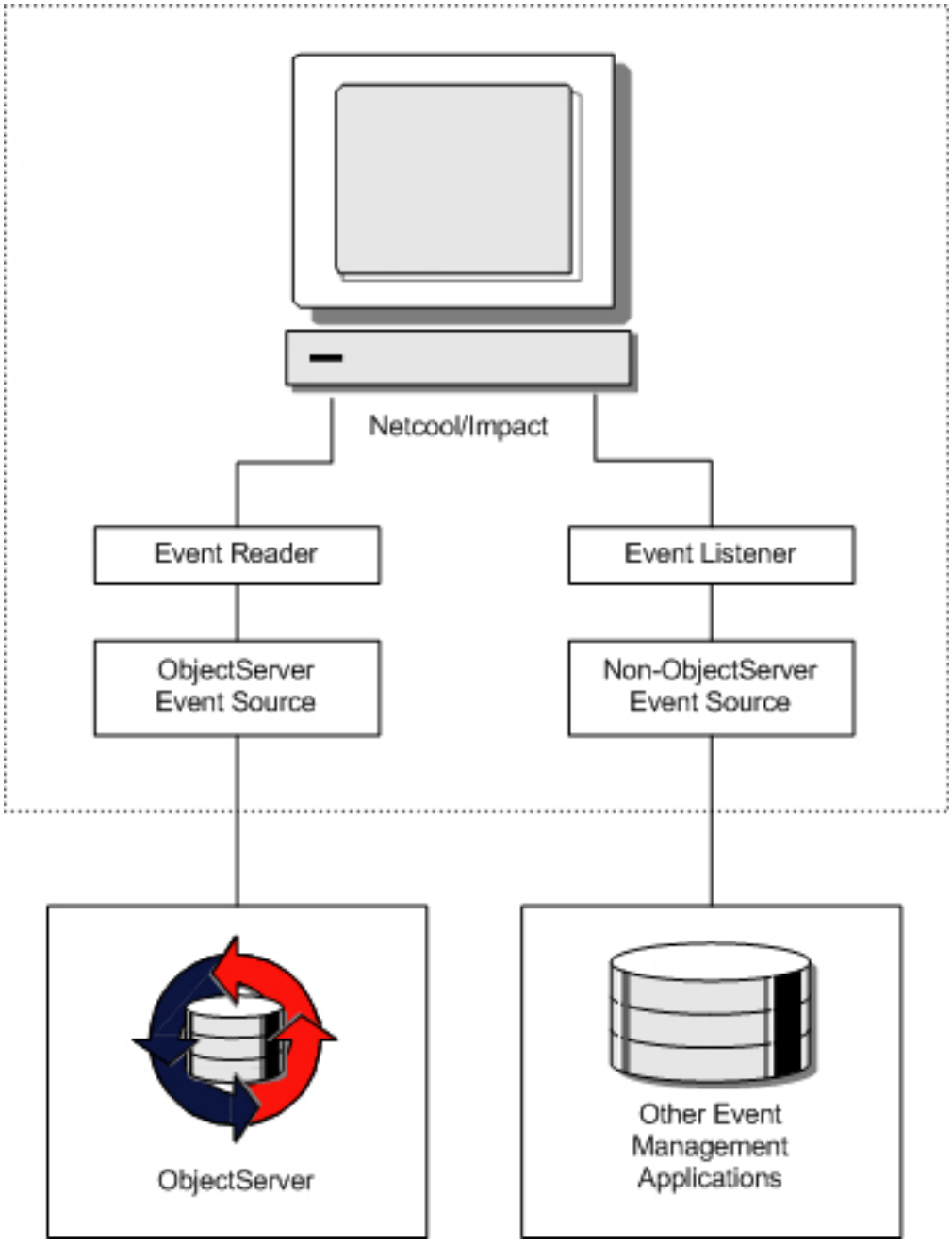

Figure 3. Event source architecture

Setting up ObjectServer event sources

Use this procedure to set up an ObjectServer event source.

Procedure

- Get the connection information for the ObjectServer.

  This information is the host name or IP address of the ObjectServer host system and the port number. The default port number for the ObjectServer is `4100`.
- Create and configure the event source.

  For more information, see *"Configuring the default ObjectServer data source"* in the *Administration Guide*.
- After you create the event source, you can then create and configure an associated event reader service.

  For more information about creating and configuring an event reader service, see the *User Interface Guide*.

Chapter 8. Working with services

You work with services by configuring predefined services, and creating and configure user-defined services.

Services overview

Services perform much of the functionality associated with the Impact Server, including monitoring event sources, sending and receiving e-mail, and triggering policies.

The most important service is the OMNIbus event reader, which you can use to monitor an ObjectServer for new, updated or deleted events. The event processor, which processes the events retrieved from the readers and listeners is also important to the function of Netcool/Impact.

Internal services control the application's standard processes, and coordinate the performed tasks, for example:

- Receiving events from the ObjectServer and other external databases
- Executing policies
- Responding to and prioritizing alerts
- Sending and receiving e-mail and instant messages
- Handling errors

Some internal services have defaults, that you can enable rather than configure your own services, or in addition to creating your own. For some of the basic internal services, it is only necessary to specify whether to write the service log to a file. For other services, you need to add information such as the port, host, and startup data.

User defined services are services that you can create for use with a specific policy.

Generally, you set up services once, when you first design your solution. After that, you do not need to actively manage the services unless you change the solution design.

To set up services, you must first determine what service functionality you need to use in your solution. Then, you create and configure the required services using the GUI. After you have set up the services, you can start and stop them, and manage the service logs.

Predefined services

Predefined services are services that are created automatically when you install Netcool/Impact. You can configure predefined services, but you cannot create new instances of the predefined services and you cannot delete existing ones.

These services are predefined:

- Event processor
- E-mail sender
- Hibernating policy activator

- Policy logger
- Command-line manager

User-defined services

User-defined services are services that you can create, modify, and delete. You can also use the default instance of these services that are created at installation.

You can create user-defined services by using the defaults that are stored in the global repository or select them from a list in the services task pane in the navigation panel. All user-defined services are also listed in the services panel where you can start them and stop them, just as you do the internal services. You can add these services to a project as project members.

These services are user-defined:

-
- Event readers
- Event listeners
- E-mail readers
- Policy activators

Chapter 9. OMNIbus event reader service

OMNIbus event readers are services that monitor a Netcool/OMNIbus ObjectServer event source for new, updated, and deleted alerts and then runs policies when the alert information matches filter conditions that you define.

The event reader service uses the host and port information of a specified ObjectServer data source so that it can connect to an Objectserver to poll for new and updated events and store them in a queue. The event processor service requests events from the event reader. When an event reader discovers new, updated, or deleted alerts in the ObjectServer, it retrieves the alert and sends it to an event queue. Here, the event waits to be handled by the event processor.

You configure this service by defining a number of restriction filters that match the incoming events, and passing the matching events to the appropriate policies. The service can contain multiple restriction filters, each one triggering a different policy from the same event stream, or it can trigger a single policy.

You can configure an event reader service to chain multiple policies together to be run sequentially when triggered by an event from the event reader.

Important: Before you create an OMNIbus event reader service, you must have a valid ObjectServer data source to which the event reader will connect to poll for new and updated events.

OMINbus event reader architecture

This diagram shows the relationship between Netcool/Impact, an OMNIbus event reader, and an ObjectServer.

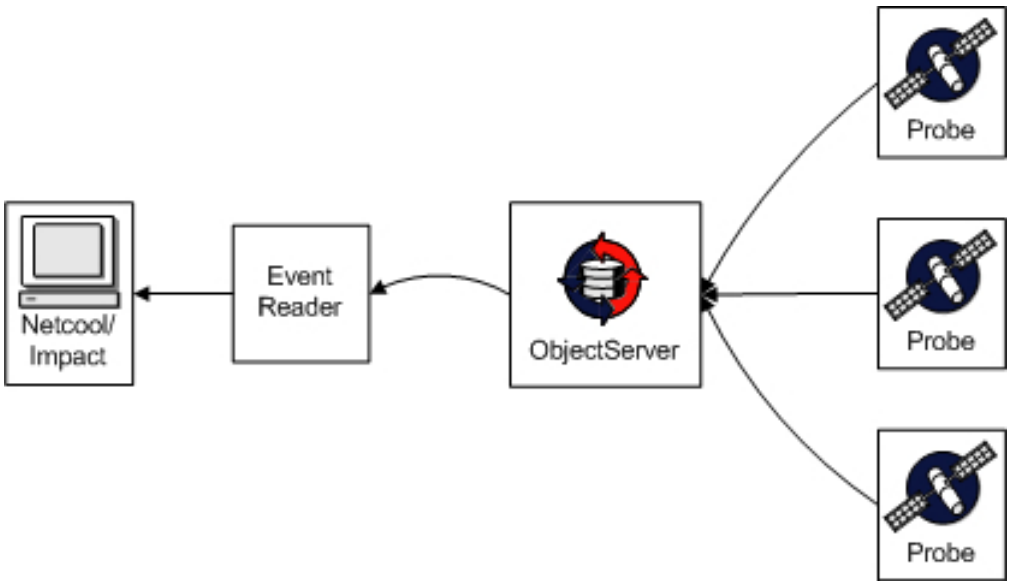

Figure 4. Event reader architecture

OMNIbus event reader process

The phases of the OMNIbus event reader process are startup, event polling, event querying, deleted event notification, and event queueing.

Startup
When the event reader is started it reads events using the StateChange or serial value that it used before being shut down. To read all the events on start-up, click **Clear State**.

Event Polling
During the event polling phase, the OMNIbus event reader queries the ObjectServer at intervals for all new and unprocessed events. You set the polling interval when you configure the event reader.

Event Querying
When the OMNIbus event reader queries the ObjectServer, either at startup, or when polling for events at intervals, it reads the state file, retrieves new or updated events, and records the state file.. For more information, see "Event querying."

Deleted Event Notification
If the OMNIbus event reader is configured to run a policy when an event is deleted from the ObjectServer, it listens to the ObjectServer through the IDUC interface for notification of deleted alerts. The IDUC delete notification includes the event field data for the deleted alert.

Event Queueing
After it retrieves new or updated events, or has received events through delete notification, the OMNIbus event reader compares the field data in the events to its set of filters.. For more information, see "Event queuing" on page 53.

Event querying

When the OMNIbus event reader queries the ObjectServer, either at startup, or when polling for events at intervals, it reads the state file, retrieves new or updated events, and records the state file.

Reading the state file
The state file is a text file used by the OMNIbus event reader to cache state information about the last event read from the ObjectServer.. The event reader reads the state file to find the `Serial` or `StateChange` value of the last read event. For more information, see "Reading the state file."

Retrieving new or updated events
The event reader connects to the ObjectServer and retrieves new or updated events that have occurred since the last read event. During this phase, the event reader retrieves all the new or updated events from the ObjectServer, using information from the state file to specify the correct subset of events.

Recording the state file
After the event reader retrieves the events from the ObjectServer, it caches the `Serial` or `StateChange` value of the last processed event.

Reading the state file

The state file is a text file used by the OMNIbus event reader to cache state information about the last event read from the ObjectServer.

If the event reader is configured to get only new events from the ObjectServer, the state file contains the `Serial` value of the last event read from the ObjectServer. If the event reader is configured to get both new and updated events from the ObjectServer, the file contains the `StateChange` value of the last read event.

The event reader reads the contents of the state file whenever it polls the ObjectServer and passes the `Serial` or `StateChange` value as part of the query.

Event queuing

After it retrieves new or updated events, or has received events through delete notification, the OMNIbus event reader compares the field data in the events to its set of filters.

If the event matches one or more of its filters, the event reader places the event in the event queue with a pointer to the corresponding policy. After the events are in the event queue, they can be picked up by the event processor service. The event processor passes the events to the corresponding policies to the policy engine for processing.

OMNIbus event reader configuration

You can configure the following properties of an OMNIbus event reader.

- Event reader name
- ObjectServer event source you want the event reader to monitor
- Interval at which you want the event reader to poll the ObjectServer
- Event fields you want to retrieve from the ObjectServer
- Event mapping
- Event locking
- Order in which the event reader retrieves events from the ObjectServer
- Start up, service log, and reporting options

OMNIbus event reader service General Settings tab

Use this information to configure the general settings of the OMNIbus event reader service.

Table 9. EventReader service - general settings tab

Table Element	Description
Service name	Type a unique name to identify the service.
Data Source	Select an OMNIbusObjectServer data source. The ObjectServer data source represents the instance of the Netcool/OMNIbus ObjectServer that you want to monitor using this service. You can use the default ObjectServer data source that is created during the installation, defaultobjectserver.
Polling Interval	The polling interval is the interval in milliseconds at which the event reader polls the ObjectServer for new or updated events. Select or type how often you want the service to poll the events in the event source. If you leave this field empty, the event reader polls the ObjectServer every 3 seconds (3000 milliseconds).

Table 9. EventReader service - general settings tab (continued)

Table Element	Description
Restrict Fields: Fields	You can complete this step when you have saved the **OMNIbusEventReader** service. You can specify which event fields you want to retrieve from the ObjectServer. By default, all fields are retrieved in the alerts. To improve OMNIbus event reader performance and reduce the performance impact on the ObjectServer, configure the event reader to retrieve only those fields that are used in the corresponding policies. Click the **Fields** button to access a list of all the fields available from the selected ObjectServer data source. You can reduce the size of the query by selecting only the fields that you need to access in your policy. Click the **Optimize List** button to implement the changes. The **Optimize List** button becomes enabled only when the **OMNIbusEventReader** service has been saved.
Startup: Automatically when server starts	Select to automatically start the service when the server starts. You can also start and stop the service from the GUI.
Service log: Write to file	Select to write log information to a file.
Collect Reports: Enable	Select to enable data collection for the Policy Reports.
Clear State: Clear	When you click the **Clear State** button, the `Serial` and `StateChange` information stored for the event reader is reset to 0. The event reader retrieves all events in the ObjectServer at startup and places them in the event queue for processing. If the event reader is configured to get updated events, it queries the ObjectServer for all events where `StateChange >= 0`. Otherwise, it queries the ObjectServer for events where `Serial > 0`. You can use the **Clear State** button only to clear the event reader state when the service is stopped. Clicking the button while the service is running does not change the state of the event reader.
Clear Queue: Clear	Click to clear unprocessed events.

OMNIbus event reader service Event Mapping tab

In the Event Mapping tab, you set events to trigger policies when they match a filter.

Table 10. Event Mapping tab

Window element	Description
Event Matching	
Test events with all filters	Select this option to test events with all filters and run any matching policies. If an event matches more than one filter, all policies that match the filtering criteria will be triggered.
Stop testing after first match	Select this option to stop testing after the first matching policy, and trigger only the first matching policy.
Actions	
Get updated events	Select to receive updated events as well as new events from the ObjectServer (All new events are automatically sent). See also the description of the **Order By** field below for more information.

Table 10. Event Mapping tab (continued)

Window element	Description
Get status events	Select to receive the status events that the **Self Monitoring** service inserts into the ObjectServer.
Run policy on deletes	Select if you want the event reader to receive notification when alerts are deleted from the ObjectServer. Then, select the policy you want to run when notification occurs from the **Policy** list.
Policy	Select a policy to run when events are cleared from the ObjectServer.
Event Locking: enable	Select if you want to use event order locking and type the locking expression in the **Expression** field. Event locking is a feature that allows a multi-threaded event processor to categorize incoming alerts based on the values of specified alert fields and then to process them within a category one at a time. With event locking enabled, if more than one event exists with a certain lock value, then these events are not processed at the same time. These events are processed in a specific order in the queue. You use event locking in situations where you want to prevent a multi-threaded event processor from attempting to access a single resource from more than one instance of a policy running simultaneously.

Expression	The locking expression consists of one or more alert field names.
	
	To lock on a single field, specify the field name, for example:
	`Node`
	
	To lock more than one field, concatenate them with the + sign, for example:
	`Node+Severity`
	
	If the value of that field is the same in both events, then one event is locked and the second thread must wait until the first one is finished.

New Mapping	Click to add a new filter.

Table 10. Event Mapping tab (continued)

Window element	Description
Order by	If you want to order incoming events retrieved from the ObjectServer, type the name of an alert field or a comma-separated list of fields. The event reader will sort incoming events in ascending order by the contents of this field. This field or list of fields is identical to the contents of an ORDER BY clause in an SQL statement. If you specify a single field, the event reader sorts incoming events by the specified field value. If you specify multiple fields, the events are grouped by the contents of the first field and then sorted within each group by the contents of the second field, and so on. For example, to sort incoming events by the contents of the **Node** field, type `Node`. To sort events first by the contents of the **Node** field and then by the contents of the **Summary** field, type `Node, Summary`. You can also specify that the sort order is ascending or descending using the `ASC` or `DESC` key words. For example, to sort incoming events by the contents of the **Node** field in ascending order, type the following `Node ASC`. Note that all events retrieved from the ObjectServer are initially sorted by either the **Serial** or **StateChange** field before any additional sorting operations are performed. If you select the **Get updated events** option (see the **Actions** check box in the **Event Mapping** section of the window), the events are sorted by the **StateChange** field. If this option is not specified, incoming events are sorted by the **Serial** field.
Analyze Event Mapping Table	Click to analyze the filters in the Event Mapping table.

Mappings

Event mappings allow you to specify which policies you want to be run when certain events are retrieved.

Each mapping consists of a filter that specifies the type of event and a policy name. You must specify at least one event mapping for the event reader to work.

The syntax for the filter is the same as the `WHERE` clause in an SQL `SELECT` statement. This clause consists of one or more comparisons that must be true in order for the specified policy to be executed. For more information about the SQL filter syntax, see the *Policy Reference Guide*.

The following examples show event mapping filters.

```
AlertKey = 'Node not responding'
AlertKey = 'Node not reachable by network ping' AND Node = 'ORA_Host_01'
```

Event matching

You can specify whether to run only the first matching policy in the event mappings or to run every policy that matches.

If you choose to run every policy that matches, the OMNIbus event reader will place a duplicate of the event in the event queue for every matching policy. The event will be processed as many times as their are matching filters in the event reader.

Actions

By default, the event broker monitors the ObjectServer for new alerts, but you can also configure it to monitor for updated alerts and to be notified when an alert is deleted.

In addition, you can configure it to get all the unprocessed alerts from the ObjectServer at startup.

Event locking

Event locking allows a multithreaded event broker to categorize incoming alerts based on the values of specified alert fields and then to process them within a category one at a time in the order that they were sent to the ObjectServer.

Event locking locks the order in which the event broker processes alerts within each category.

Remember: When event locking is enabled in the reader, the events read by it are only processed in the primary server of the cluster.

You use event locking in situations where you want to preserve the order in which incoming alerts are processed, or in situations where you want to prevent a multithreaded event processor from attempting to access a single resource from more than one instance of a policy running simultaneously.

You specify the way the event reader categorizes incoming alerts using an expression called a locking expression. The locking expression consists of one or more alert field names concatenated with a plus sign (+) as follows:

field[+*field*...]

Where *field* is the name of an alert field in the `alerts.status` table of the ObjectServer.

When an event reader retrieves alerts from the ObjectServer, it evaluates the locking expression for each incoming alert and categorizes it according to the contents of the alert fields in the expression.

For example, when using the locking expression `Node`, the event broker categorizes all incoming alerts based on the value of the `Node` alert field and then processes them within a category one at a time in the order that they were sent to the ObjectServer.

In the following example:

```
Node+AlertKey
```

The event broker categorizes all incoming alerts based on the concatenated values of the `Node` and `AlertKey` fields. In this example, an alert whose `Node` value is `Node1` and `AlertKey` value is `123456` is categorized separately

Event order

The reader first sorts based on StateChange or Serial value depending on whether Get Updates is used or not.

Each event has a unique Serial so the Order by field is ignored. In instances where there is more than one event with the same StateChange, the reader uses the Order By field to sort events after they are sorted in ascending order of StateChange.

Chapter 10. Database event listener service

The database event listener service monitors an Oracle event source for new, updated, and deleted events.

This service works only with Oracle databases. When the service receives the data, it evaluates the event against filters and policies specified for the service and sends the event to the matching policies. The service listens asynchronously for events generated by an Oracle database server and then runs one or more policies in response.

You configure the service using the GUI. The configuration properties allow you to specify one or more policies that are to be run when the listener receives incoming events from the database server.

Setting up the database server

Before you can use the database event listener, you must configure the database client and install it into the Oracle database server.

About this task

The database client is the component that sends events from the database server to Netcool/Impact. It consists of a set of Oracle Java schema objects and related properties files. When you install the Impact Server, the installer copies a tar file containing the client program files to the local system.

Perform these steps to set up the database server:

Procedure

1. Copy the client tar file to the system where Oracle is running and extract its contents.
   a. Copy the client tar file, `$IMPACT_HOME/install/agents/oracleclient.tar`, from Netcool/Impact into a temporary directory on the system where Oracle is running.
   b. Extract the tar contents using the UNIX tar command or a Windows archive utility, for example WinZip.
2. Edit the nameserver properties file on the database client side.

   The client tar file contains the `nameserver.props` file that the database client uses to determine the NameServer connection details. For information about configuring this file, see "Editing the nameserver.props file for the database client" on page 60.
3. Optional: Edit the listener properties file.

   The client tar file contains the `impactdblistener.props` with additional settings for the database client. For information about configuring this file, see "Editing the listener properties file" on page 61.
4. Install the client files into the database server using the Oracle `loadjava` utility.

Oracle provides the $ORACLE_HOME/bin/loadjava utility that you can use to install the client files into the database server. For information about installing the client files into the database server, see “Installing the client files into Oracle” on page 61.

5. Grant database permissions.

   You must grant a certain set of permissions in the Oracle database server in order for the database event listener to function.. For more information about granting database permissions, see “Granting database permissions” on page 62.

Editing the nameserver.props file for the database client

The client tar file contains the nameserver.props file that the database client uses to determine the NameServer connection details.

The database client uses the name server to find and connect to the primary instance of the Impact Server.

Restriction: In clustering configurations of Netcool/Impact, the database event listener only runs in the primary server.

The following example shows a sample of the nameserver.props file that the database client can use to connect to a single-server configuration of the NameServer.

```
nameserver.0.host=NCI1
nameserver.0.port=9080
nameserver.0.location=/nameserver/services

nameserver.userid=tipadmin
nameserver.password=tippass

nameserver.count=1
```

In this example, the NameServer is located on the NCI1 Impact Server, and is running on the default port, 9080. The NameServer user and password have default values, tipadmin, and tippass.

The following example shows a sample of the nameserver.props file that the database client can use to connect to a cluster that consists of two NameServer instances.

```
nameserver.0.host=NCI1
nameserver.0.port=9080
nameserver.0.location=/nameserver/services

nameserver.1.host=NCI2
nameserver.1.port=9080
nameserver.1.location=/nameserver/services

nameserver.userid=tipadmin
nameserver.password=tippass

nameserver.count=2
```

In this example, the NameServers are located on systems named NCI1, and NCI2 Impact Servers, and are running on the default port, 9080.

Editing the listener properties file

The client tar file contains the impactdblistener.props with additional settings for the database client.

Edit this file so that it contains the correct name for the Impact Server cluster. You can also change debug and delimiter properties.

Table 11 shows the properties in the listener properties file:

Table 11. Database client listener properties file

Property	Description
impact.cluster.name	Name of the Impact Server cluster where the database event listener is running. The default value for this property is NCICLUSTER.
impact.dblistener.debug	Specifies whether to run the database client in debug mode. The default value for this property is true.
impact.dblistener.delim	Specifies the delimiter character that separates name/value pairs in the VARRAY sent by Java stored procedures to the database client. The default value for this property is the pipe character (|). You cannot use the colon (:) as a delimiter.

Installing the client files into Oracle

Oracle provides the $ORACLE_HOME/bin/loadjava utility that you can use to install the client files into the database server.

Before you begin

If you are migrating to Netcool/Impact 6.1, remove any preexisting jar and properties files. To remove the preexisting files, use the following command:

```
dropjava -user username/password <file>
```

username, and *password* is a valid user name and password for a user whose schema contains the database resources where the Java stored procedures are run.

Procedure

1. Navigate to the ORACLE_HOME/bin directory.
2. Install the client jar, and properties files.
   a. Install the nameserver.jar file using the following command:
      ```
      loadjava -user username/password -resolve nameserver.jar
      ```
   b. Install the impactdblistener.jar file using the following command:
      ```
      loadjava -user username/password -resolve impactdblistener.jar
      ```
   c. Install the nameserver.props file using the following command:
      ```
      loadjava -user username/password -resolve nameserver.props
      ```
   d. Install the impactdblistener.props file using the following command:
      ```
      loadjava -user username/password -resolve impactdblistener.props
      ```

   Important: You must follow this order of installation, otherwise loadjava will not be able to resolve external references between files, and report errors during installation.

Granting database permissions

You must grant a certain set of permissions in the Oracle database server in order for the database event listener to function.

Procedure

1. Grant the permissions by entering the following commands at an Oracle command prompt:

```
exec dbms_java.grant_permission( 'SCHEMA','SYS:java.net.SocketPermission',
 'hostname:port','connect,resolve' )
/
exec dbms_java.grant_permission( 'SCHEMA','SYS:java.net.SocketPermission',
 'hostname:listener_port','connect,resolve' )
/
exec dbms_java.grant_permission( 'SCHEMA', 'SYS:java.lang.RuntimePermission',
 'shutdownHooks'  , '');
/
exec dbms_java.grant_permission( 'SCHEMA','SYS:java.util.logging.LoggingPermission'
,'control', '' );
/
exec dbms_java.grant_permission('SCHEMA', 'SYS:java.util.PropertyPermission',
 '*', 'read, write')
/
exec dbms_java.grant_permission( 'SCHEMA', 'SYS:java.lang.RuntimePermission',
 'getClassLoader', '' )
/
exec dbms_java.grant_permission( 'SCHEMA','SYS:java.net.SocketPermission',
 'hostname:40000','connect,resolve' );
```

   SCHEMA is the name of your database schema; *hostname* is the name of the host where you are running the Impact Server; *port* is the HTTP port on the server; and *listener_port* is the port used by the database event listener.

2. Optional: You may need to grant socket permissions to additional ports for Oracle.

   For example, the next two port numbers in the allocation sequence for use in connecting to the database event listener service. You can adjust the communication port on the Impact Server so that the Oracle client can grant permissions to connect to the Impact Server on that port using the `impact.server.rmiport` property. For example:

   ```
   IMPACT_HOME/etc/<servername>_server.props  impact.server.rmiport=50000
   ```

   Grant the permission to connect to this port in your Oracle database (port 50000 in the example), otherwise the Impact Server starts at a random port. You have to grant permissions for a different port each time the Impact Server is restarted.

Database event listener service configuration window

You configure the database event listener service by setting events to trigger policies when they match a filter.

Table 12. Database Event Listener service configuration window

Window element	Description
Event Matching	
Test events with all filters	Click this icon if, when an event matches more than one filter, you want to trigger all policies that match the filtering criteria.

Table 12. Database Event Listener service configuration window (continued)

Window element	Description
Stop testing after first match	Click this icon if you want to trigger only the first matching policy. You can choose to test events with all filters and run any matching policies or to stop testing after the first matching policy.
New Mapping: New	Click this icon to create an event filter.
Analyze Event Mapping Table	Click this icon to view any conflicts with filter mappings that you have set for this service.
Startup: Automatically when server starts	Select to automatically start the service when the server starts. You can also start and stop the service from the GUI.
Service log: Write to file	Select to write log information to a file.

Sending database events

Perform these tasks to configure the database to send events.

- Create a call spec that publishes the `sendEvent()` function from the database client library.
- Create triggers that call the resulting stored procedure.

Before you create these objects in the database, you must understand what kind of database events you want to send and what conditions will cause them to be sent. For example, if you want to send an event to Netcool/Impact every time a row is inserted into a table, you must know the identity of the table, the subset of row information to send as part of the event and the name of the condition (for example, after insert) that triggers the operation.

For more information about Java stored procedures, call specs, and triggers, see the *Oracle Java Stored Procedure Developer's Guide*.

Creating the call spec

The database client exposes a function named `sendEvent()` that allows Oracle schema objects (in this case, triggers) to send events to Netcool/Impact.

The `sendEvent()` function is located in the class `com.micromuse.response.service.listener.database. DatabaseListenerClient`, which you compiled and loaded when you installed the client into the database server.

The function has the following syntax:

```
sendEvent(java.sql.Array x)
```

Where each element in array *x* is a string that contains a name/value pair in the event.

In order for Oracle objects to call this function, you must create a call spec that publishes it to the database as a stored procedure. The following example shows a call spec that publishes `sendEvent()` as a procedure named `test_varray_proc`:

```
CREATE OR REPLACE PROCEDURE test_varray_proc(v_array_inp db_varray_type)
    AS LANGUAGE JAVA
    NAME
    'com.micromuse.response.service.listener.database.DatabaseListenerClient.
    sendEvent(java.sql.Array)';
/
```

In this example, `db_varray_type` is a user-defined `VARRAY` that can be described using the following statement:

```
CREATE TYPE db_varray_type AS VARRAY(30) OF VARCHAR2(100);
```

This call spec and `VARRAY` type are used in examples elsewhere in this chapter.

When you call the procedure published with this call spec, you pass it an Oracle `VARRAY` in which each element is a string that contains a name/value pair in the event. The name and value in the string are separated using the pipe character (|) or another character as specified when you configured the database client.

Creating triggers

You can create triggers for DML events, DDL events, system events, and user events.

DML events triggers

DML events are sent to Netcool/Impact when the database performs operations that change rows in a table.

These include the standard SQL `INSERT`, `UPDATE`, and `DELETE` commands.

You configure the database to send DML events by creating triggers that are associated with these operations. Most often, these triggers take field data from the rows under current change and pass it to the database client using the call spec you previously created. In this way, the database reports the inserts, updates, and deletes to Netcool/Impact for processing as events.

When the database client receives the field data from the trigger, it performs a `SELECT` operation on the table to determine the underlying data type of each field. Because the corresponding row is currently under change, Oracle is likely to report a mutating table error (`ORA-04091`) when the database client performs the `SELECT`.

To avoid receiving this error, your DML triggers must create a copy of the row data first and then use this copy when sending the event.

The following example contains table type declarations, variable declarations, and trigger definitions that create a temporary copy of row data. You can modify this example for your own use. This example uses the type `db_varray_type` described in the previous section. The triggers in the example run in response to changes made to a table named `dept`.

This example contains:

- Type declaration for `deptTable`, which is a nested table of `db_varray_type`.
- Variable declaration for `dept1`, which is a table of type `deptTable`. This table stores the copy of the row data.
- Variable declaration for `emptyDept`, which is a second table of type `deptTable`. This table is empty and is used to reset `dept1`.
- Trigger definition for `dept_reset`, which is used to reset `dept1`.

- Trigger definition for dept_after_row, which populates dept1 with field data from the changed rows.
- Trigger definition for dept_after_stmt, which loops through the copied rows and sends the field data to the database client using the call spec defined in the previous section.

The trigger definition for dept_after_row is intentionally left incomplete in this example, because it varies depending on whether you are handling INSERT, UPDATE or DELETE operations.

This is an example definition for this trigger:

```
CREATE OR REPLACE PACKAGE dept_pkg AS

/* deptTable is a nested table of VARRAYs that will be sent */
/* to the database client */
TYPE deptTable IS TABLE OF db_varray_type INDEX BY BINARY_INTEGER;

/* dept1 will store the actual VARRAYs
dept1 deptTable;

/* emptyDept is used for initializing dept1 */
emptyDept deptTable;

end;
/

CREATE OR REPLACE TRIGGER dept_reset
BEFORE INSERT OR UPDATE OR DELETE ON dept
BEGIN

/* Initialize dept1 */
dept_pkg.dept1 := dept_pkg.emptyDept;
end;
/

/* CREATE OR REPLACE TRIGGER dept_after_row
/* AFTER INSERT OR UPDATE OR DELETE ON dept
/* FOR EACH ROW
/* BEGIN

/* This trigger intentionally left incomplete. */
/* See examples in following sections of this chapter. */

end;
/

CREATE OR REPLACE TRIGGER dept_after_stmt
AFTER INSERT OR UPDATE OR DELETE ON dept
BEGIN

/* Loop through rows in dept1 and send field data to database client */
/* using call proc defined in previous section of this chapter */

for i in 1 .. dept_pkg.dept1.count loop
    test_varray_proc(dept_pkg.dept1(i));
end loop;

end;
/
```

Insert events triggers

To send an event to Netcool/Impact when Oracle performs an INSERT operation, you must first create a trigger that copies the inserted row data to a temporary table.

You then use another trigger as shown in the example to loop through the temporary table and send the row data to the database client for processing.

A typical insert trigger contains a statement that populates a VARRAY with the wanted field data and then assigns the VARRAY as a row in the temporary table. Each element in the VARRAY must contain a character-delimited set of name/value pairs that the database client converts to event format before sending it to Netcool/Impact. The default delimiter character is the pipe symbol (|).

The VARRAY must contain an element for a field named EVENTSOURCE. This field is used by the database client to determine the table where the database event originated.

The following example shows a typical VARRAY for insert events:

```
db_varray_type('EVENTSOURCE | SCOTT.DEPT',
'DEPTNO | '||:NEW.DEPTNO, 'LOC | '||:NEW.LOC,
'DNAME | '||:NEW.DNAME, 'IMPACTED | '||:NEW.IMPACTED);
```

In this example, the VARRAY contains an EVENTSOURCE field and fields that contain values derived from the inserted row, as contained in the NEW pseudo-record passed to the trigger. The value of the EVENTSOURCE field in this example is the dept table in the Oracle SCOTT schema.

The following example shows a complete trigger that copies new row data to the temporary table dept1 in package dept_pkg.

```
CREATE OR REPLACE TRIGGER dept_after_row
AFTER INSERT ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT', 'DEPTNO | '||:NEW.DEPTNO,
'LOC | '||:NEW.LOC, 'DNAME | '||:NEW.DNAME,
'IMPACTED | '||:NEW.IMPACTED);

end;
/
```

For a complete example that shows how to send an insert event, see "Insert event trigger example" on page 69.

Update and delete events triggers

You can send update and delete events using the same technique you use to send insert events.

When you send update and delete events, however, you must obtain the row values using the OLD pseudo-record instead of NEW.

The following example shows a trigger that copies updated row data to the temporary table dept1 in package dept_pkg.

```
CREATE OR REPLACE TRIGGER dept_after_row
AFTER UPDATE ON dept
FOR EACH ROW
```

```
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT',
'DEPTNO | '||:OLD.DEPTNO, 'LOC | '||:OLD.LOC,
'DNAME | '||:OLD.DNAME, 'IMPACTED | '||:OLD.IMPACTED);

end;
/
```

The following example shows a trigger that copies deleted row data to the temporary table dept1.

```
CREATE OR REPLACE TRIGGER dept_after_row
AFTER DELETE ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT',
'DEPTNO | '||:OLD.DEPTNO, 'LOC | '||:OLD.LOC,
'DNAME | '||:OLD.DNAME, 'IMPACTED | '||:OLD.IMPACTED);

end;
/
```

DDL events triggers

DDL events are sent to Netcool/Impact when the database performs an action that changes a schema object.

These actions include the SQL CREATE, ALTER, and DROP commands.

To send DDL events, you create a trigger that populates a VARRAY with data that describes the DDL action and the database object that is changed by the operation. Then, you pass the VARRAY element to the database client for processing. As with DML events, the VARRAY contains a character-delimited set of name/value pairs that the database client converts to event format before sending to Netcool/Impact.

DDL events require two VARRAY elements: EVENTSOURCE, as described in the previous section, and TRIGGEREVENT. Typically, you populate the TRIGGEREVENT element with the current value of Sys.sysevent.

The following example shows a typical VARRAY for DDL events.

```
db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
```

The following example shows a complete trigger that sends an event to Netcool/Impact before Oracle executes a CREATE command.

```
CREATE OR REPLACE TRIGGER ddl_before_create
BEFORE CREATE
ON SCOTT.schema

DECLARE
my_before_create_varray db_varray_type;

BEGIN
my_before_create_varray :=
db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name,
'USERNAME | '||Sys.login_user,'INSTANCENUM | '||Sys.instancenum,
```

```
'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_before_create_varray);

end;
/
```

System events triggers

System events are sent to Netcool/Impact when the Oracle server starts up, shuts down or reports a system level error.

System events only work if the user who owns the corresponding triggers has SYSDBA privileges (for example, the SYS user).

To send DDL events, you create a trigger that populates a VARRAY with data that describes the system action. Then, you pass the VARRAY element to the database client for processing. As with DDL events, system events require the TRIGGEREVENT element to be populated, typically with the value of Sys.sysevent.

The following example shows a typical VARRAY for system events.

```
db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
' OBJECTNAME | '||Sys.database_name,
' USER_NAME | '||Sys.login_user,
' INSTANCE_NUM | '||Sys.instance_num);
```

The following example shows a complete trigger that sends an event to Netcool/Impact at Oracle startup.

```
CREATE OR REPLACE TRIGGER databasestartuptrigger
AFTER STARTUP
ON database

BEGIN
v_array_inp := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
' OBJECTNAME | '||Sys.database_name,
' USER_NAME | '||Sys.login_user, ' INSTANCE_NUM | '||Sys.instance_num);
test_varray_proc(v_array_inp);
```

User events triggers

User events are sent to Netcool/Impact when a user logs in to or out of Oracle.

To send user events, you create a trigger that populates a VARRAY with data that describes the user action. Then, you pass the VARRAY element to the database client for processing. As with system events, user events require the TRIGGEREVENT element to be populated, typically with the value of Sys.sysevent. If you do not specify a value for the EVENTSOURCE element, the database client uses the name of the database,

The following example shows a typical VARRAY for user events.

```
db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE |'||Sys.database_name, 'LOGINUSER | ' ||Sys.login_user,
'INSTANCENUM | '||Sys.instance_num, 'TRIGGERNAME | USER_LOGIN');
```

The following example shows a complete trigger that sends an event to Netcool/Impact at when a user logs in.

```
CREATE OR REPLACE TRIGGER user_login
AFTER logon
on schema

DECLARE
```

```
my_login_varray db_varray_type;

BEGIN
my_login_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE |'||Sys.database_name, 'LOGINUSER | ' ||Sys.login_user,
'INSTANCENUM | '||Sys.instance_num, 'TRIGGERNAME | USER_LOGIN');
test_varray_proc(my_login_varray);

end;
/
```

Insert event trigger example

This example shows how to create a set of Oracle triggers that send an insert event to Netcool/Impact.

```
CREATE OR REPLACE PACKAGE dept_pkg AS
TYPE deptTable IS TABLE OF db_varray_type INDEX BY BINARY_INTEGER;

dept1 deptTable;
emptyDept deptTable;

end;
/

CREATE OR REPLACE TRIGGER dept_reset
BEFORE INSERT OR UPDATE OR DELETE ON dept
BEGIN

dept_pkg.dept1 := dept_pkg.emptyDept;

end;
/

CREATE OR REPLACE TRIGGER dept_after_row
AFTER INSERT ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT', 'DEPTNO | '||:NEW.DEPTNO,
'LOC | '||:NEW.LOC, 'DNAME | '||:NEW.DNAME, 'IMPACTED | '||:NEW.IMPACTED);

end;
/

CREATE OR REPLACE TRIGGER dept_after_stmt
AFTER INSERT OR UPDATE OR DELETE ON dept
BEGIN

for i in 1 .. dept_pkg.dept1.count loop
    test_varray_proc(dept_pkg.dept1(i));
end loop;

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Update event trigger example

This example shows how to create a set of Oracle triggers that send an update event to Netcool/Impact.

```
CREATE OR REPLACE PACKAGE dept_pkg AS
TYPE deptTable IS TABLE OF db_varray_type INDEX BY BINARY_INTEGER;

dept1 deptTable;
emptyDept deptTable;

end;
/

CREATE OR REPLACE TRIGGER dept_reset
BEFORE INSERT OR UPDATE OR DELETE ON dept
BEGIN

dept_pkg.dept1 := dept_pkg.emptyDept;

end;
/

CREATE OR REPLACE TRIGGER dept_after_row
AFTER UPDATE ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT', 'DEPTNO | '||:OLD.DEPTNO,
'LOC | '||:OLD.LOC, 'DNAME | '||:OLD.DNAME, 'IMPACTED | '||:OLD.IMPACTED);

end;
/

CREATE OR REPLACE TRIGGER dept_after_stmt
AFTER INSERT OR UPDATE OR DELETE ON dept
BEGIN

for i in 1 .. dept_pkg.dept1.count loop
    test_varray_proc(dept_pkg.dept1(i));
end loop;

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Delete event trigger example

This example shows how to create a set of Oracle triggers that send a delete event to Netcool/Impact.

```
CREATE OR REPLACE PACKAGE dept_pkg AS
TYPE deptTable IS TABLE OF db_varray_type INDEX BY BINARY_INTEGER;

dept1 deptTable;
emptyDept deptTable;

end;
/

CREATE OR REPLACE TRIGGER dept_reset
BEFORE INSERT OR UPDATE OR DELETE ON dept
BEGIN

dept_pkg.dept1 := dept_pkg.emptyDept;

end;
```

```
/

CREATE OR REPLACE TRIGGER dept_after_row
AFTER DELETE ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT', 'DEPTNO | '||:OLD.DEPTNO,
'LOC | '||:OLD.LOC, 'DNAME | '||:OLD.DNAME, 'IMPACTED | '||:OLD.IMPACTED);

end;
/

CREATE OR REPLACE TRIGGER dept_after_stmt
AFTER INSERT OR UPDATE OR DELETE ON dept
BEGIN

for i in 1 .. dept_pkg.dept1.count loop
    test_varray_proc(dept_pkg.dept1(i));
end loop;

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Before create event trigger example

This example shows how to create a trigger that sends an event before Oracle executes a `CREATE` command.

```
CREATE OR REPLACE TRIGGER ddl_before_create
BEFORE CREATE
ON SCOTT.schema

DECLARE
my_before_create_varray db_varray_type;

BEGIN
my_before_create_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_before_create_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

After create event trigger example

This example shows how to create a trigger that sends an event after Oracle executes a `CREATE` command.

```
CREATE OR REPLACE TRIGGER ddl_after_create
AFTER CREATE
ON SCOTT.schema

DECLARE
```

```
my_after_create_varray db_varray_type;

BEGIN
my_after_create_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_after_create_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Before alter event trigger example

This example shows how to create a trigger that sends an event before Oracle executes an `ALTER` command.

```
CREATE OR REPLACE TRIGGER ddl_before_alter
BEFORE ALTER
ON SCOTT.schema

DECLARE
my_before_alter_varray db_varray_type;

BEGIN
my_before_alter_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_before_alter_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

After alter event trigger example

This example shows how to create a trigger that sends an event after Oracle executes an `ALTER` command.

```
CREATE OR REPLACE TRIGGER ddl_after_alter
AFTER ALTER
ON SCOTT.schema

DECLARE
my_after_alter_varray db_varray_type;

BEGIN
my_after_alter_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_after_alter_varray);

end;
/
```

In this example, test_varray_proc is a call spec that publishes the sendEvent() function exposed by the database client. The type db_varray_type is a user-defined data type that represents an Oracle VARRAY. The example uses the Oracle SCOTT sample schema.

Before drop event trigger example

This example shows how to create a trigger that sends an event before Oracle executes an DROP command.

```
CREATE OR REPLACE TRIGGER ddl_before_drop
BEFORE DROP
ON SCOTT.schema

DECLARE
my_before_drop_varray db_varray_type;

BEGIN
my_before_drop_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_before_drop_varray);

end;
/
```

In this example, test_varray_proc is a call spec that publishes the sendEvent() function exposed by the database client. The type db_varray_type is a user-defined data type that represents an Oracle VARRAY. The example uses the Oracle SCOTT sample schema.

After drop event trigger example

This example shows how to create a trigger that sends an event after Oracle executes an DROP command.

```
CREATE OR REPLACE TRIGGER ddl_after_drop
AFTER DROP
ON SCOTT.schema

DECLARE
my_after_drop_varray db_varray_type;

BEGIN
my_after_drop_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'USERNAME | '||Sys.login_user,
'INSTANCENUM | '||Sys.instancenum, 'OBJECTTYPE | '||Sys.dictionary_obj_type,
'OBJECTOWNER | '||Sys.dictionary_obj_owner);
test_varray_proc(my_after_drop_varray);

end;
/
```

In this example, test_varray_proc is a call spec that publishes the sendEvent() function exposed by the database client. The type db_varray_type is a user-defined data type that represents an Oracle VARRAY. The example uses the Oracle SCOTT sample schema.

Server startup event trigger example

This example shows how to create a trigger that sends an event to Netcool/Impact at Oracle startup.

```
CREATE OR REPLACE TRIGGER databasestartuptrigger
AFTER STARTUP
ON database
```

```
BEGIN
v_array_inp := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
' OBJECTNAME | '||Sys.database_name, ' USER_NAME | '||Sys.login_user,
' INSTANCE_NUM | '||Sys.instance_num);
test_varray_proc(v_array_inp);
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Server shutdown event trigger example

This example shows how to create a trigger that sends an event to Netcool/Impact at Oracle shutdown.

```
CREATE OR REPLACE TRIGGER databaseshutdowntrigger
BEFORE SHUTDOWN
ON database

DECLARE
v_array_inp db_varray_type;

BEGIN
v_array_inp := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
' OBJECTNAME | '||Sys.database_name, ' USER_NAME | '||Sys.login_user,
' INSTANCE_NUM | '||Sys.instance_num);
test_varray_proc(v_array_inp);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Server error event trigger example

This example shows how to create a trigger that sends an event to Netcool/Impact when Oracle encounters a server error.

```
CREATE OR REPLACE TRIGGER server_error_trigger_database
AFTER SERVERERROR
ON database

DECLARE
my_varray db_varray_type;

BEGIN
my_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE | '||Sys.database_name, 'INSTANCENUM | ' ||Sys.instance_num,
'LOGINUSER | '||Sys.login_user, 'ERRORNUM | '||Sys.server_error(1));
test_varray_proc(my_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Logon event trigger example

This example shows how to create a trigger that sends an event to Netcool/Impact when a user logs in to the database.

```
CREATE OR REPLACE TRIGGER user_login
AFTER logon
on schema

DECLARE
my_login_varray db_varray_type;

BEGIN
my_login_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE |'||Sys.database_name, 'LOGINUSER | ' ||Sys.login_user,
'INSTANCENUM | '||Sys.instance_num, 'TRIGGERNAME | USER_LOGIN');
test_varray_proc(my_login_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Logoff event trigger example

This example shows how to create a trigger that sends an event to Netcool/Impact when a user logs out of the database.

```
CREATE OR REPLACE TRIGGER user_logoff
BEFORE logoff
on schema

DECLARE
my_logoff_varray db_varray_type;

BEGIN
my_logoff_varray := db_varray_type('TRIGGEREVENT | '||Sys.sysevent,
'EVENTSOURCE |'||Sys.database_name, 'LOGINUSER | ' ||Sys.login_user,
'INSTANCENUM | '||Sys.instance_num, 'TRIGGERNAME | USER_LOGOFF');
test_varray_proc(my_logoff_varray);

end;
/
```

In this example, `test_varray_proc` is a call spec that publishes the `sendEvent()` function exposed by the database client. The type `db_varray_type` is a user-defined data type that represents an Oracle `VARRAY`. The example uses the Oracle `SCOTT` sample schema.

Writing database event policies

Policies that work with database events can handle incoming events, and return events to the database.

Handling incoming database events

The database event listener passes incoming events to Netcool/Impact using the built-in `EventContainer` variable.

When the database event listener receives an event from the database, it populates the `EventContainer` member variables with the values sent by the database trigger

using the Oracle VARRAY. You can access the values of `EventContainer` using the @ or dot notations in the same way you access the field values in any other type of event.

The following example shows how to handle an incoming database event. In this example, the event was generated using the example trigger described in "Insert events triggers" on page 66.

```
// Log incoming event values

Log("Department number: " + @DEPTNO);
Log("Location: " + @LOC);
Log("Database name: " + @DNAME);
Log("Impacted: " + @IMPACTED);
```

The example prints the field values in the event to the policy log.

Returning events to the database

The database event listener supports the use of the `ReturnEvent` function in a policy to update or delete events. To use `ReturnEvent` in a database event policy, you must perform the following tasks:

Procedure

- Make sure that the database trigger that sends the event populates a special set of connection event fields.
- Call the `ReturnEvent` function in the policy that handles the events.

Populating the connection event fields

For the policy that handles events to return them to the event source, you must populate a special set of event fields in the database trigger.

These fields specify connection information for the database server. The database event listener uses this information to connect to the database when you return an updated or deleted event.

Table 13 shows the event fields that you must populate in the trigger.

Table 13. Database trigger connection event fields

Field	Description
`RETURNEVENT`	You must set a value of `TRUE` in this event field.
`USERNAME`	User name to use when connecting to the Oracle database server.
`PASSWORD`	Password to use when connecting to the Oracle database server.
`HOST`	Host name or IP address of the system where Oracle is running.
`PORT`	Connection port for the Oracle database server.
`SID`	Oracle server ID.
`KEYFIELD`	Key field in the database table, or any other field that uniquely identifies a table row.

When the database client sends the event to Netcool/Impact, it encrypts the connection information (including the database user name and password) specified in the event fields. The connection information is then unencrypted when it is received by Netcool/Impact.

The following example shows a trigger that sends an event to Netcool/Impact when a new row is inserted into the dept table. In this example, you populate the connection event fields by specifying elements in the Oracle VARRAY that you pass to the database.

```
CREATE OR REPLACE TRIGGER dept_after_row
AFTER INSERT ON dept
FOR EACH ROW
BEGIN

dept_pkg.dept1(dept_pkg.dept1.count + 1) :=
db_varray_type('EVENTSOURCE | SCOTT.DEPT', 'DEPTNO | '||:NEW.DEPTNO,
'LOC | '||:NEW.LOC, 'DNAME | '||:NEW.DNAME, 'IMPACTED | '||:NEW.IMPACTED,
'RETURNEVENT | TRUE', 'USERNAME | ora_user', 'PASSWORD | ora_passwd',
'HOST | ora_host', 'PORT | 4100', 'SID | ora_01', 'KEYFIELD | DEPTNO');

end;
/
```

Returning events to the database

You can send updated or deleted events to the database server using the ReturnEvent function.

ReturnEvent sends the event information to the database event listener, which assembles an UPDATE or DELETE command using the information. The database event listener then sends the command to the database server for processing. The UPDATE or DELETE command updates or deletes the row that corresponds to the original sent event. For more information about ReturnEvent, see the *Policy Reference Guide*.

The following policy example shows how to return an updated event to the database.

```
// Log incoming event values

Log("Department number: " + @DEPTNO);
Log("Location: " + @LOC);
Log("Database name: " + @DNAME);
Log("Impacted: " + @IMPACTED);

// Update the value of the Location field

@LOC = "New York City";

// Return the event to the database

ReturnEvent(EventContainer);
```

The following example shows how to delete an event from the database.

```
// Set the value of the DeleteEvent variable to true

@DeleteEvent = true; // @DeleteEvent name is case-sensitive

// Set the event field variables required by the database event listener
// in order to connect to Netcool/Impact

// Return the event to the database

ReturnEvent(EventContainer);
```

Chapter 11. OMNIbus event listener service

The OMNIbus event listener service is used to integrate with Netcool/OMNIbus and receive immediate notifications of fast track events.

The OMNIbus event listener is used to get fast track notifications from OMNIbus using the Accelerated Event Notification feature of OMNIbus. It receives notifications through the Insert, Delete, Update, or Control (IDUC) channel. To set up the Netcool/OMNIbus event listener, you must set its configuration properties using the GUI. The configuration properties allow you to specify one or more policies that are to be run when the OMNIbus event listener receives incoming events from Netcool/OMNIbus. For more information about OMNIbus triggers and accelerated event notification, see the *OMNIbus Administration Guide*.

Important:

- The OMNIbus event listener service works only with OMNIbus 7.3 to monitor ObjectServer events.
- If the Impact Server and OMNIbus server are located in different network domains, for the OMNIbus event listener service to work correctly, you must set the **Iduc.ListeningHostname** property in the OMNIbus server. This property must contain the IP address or fully qualified hostname of the the OMNIbus server. For more information about this property, refer to the OMNIbus documentation.

Setting up the OMNIbus event listener service

Use this procedure to create the OMNIbus event listener service.

Procedure

1. In the Tivoli Integrated Portal, in the navigation tree, click **System Configuration** > **Event Automation** > **Services**, to open the **Services** tab.
2. If required, select a cluster from the **Cluster** list.
3. Click the **Create New Service** icon in the toolbar and select **OMNIbusEventListener** to open the configuration window.
4. Enter the required information in the configuration window.
5. Click the **Save** icon in the toolbar to create the service.
6. Start the service to establish a connection to the ObjectServer and subscribe to the IDUC channel to get notifications for inserts, updates, and deletes.

Using the OMNIbus event listener service

Starting the OMNIbus event listener service establishes a connection between Netcool/Impact and the objectserver.

To ensure that the OMNIbus event listener service started successfully, check the service logs. A message like the following example is displayed if the service started:

```
Initializing Service
Connecting to the Data Source: defaultobjectserver
IDUC Connection: Established:
Iduc Hostname : swordfish
```

```
Iduc Port : 35920
Iduc Spid : 2
Registered for the IDUC Feed
Service Started
```

Triggers

You must create triggers before Netcool/Impact can receive accelerated events from Netcool/OMNIbus.

Triggers notify Netcool/Impact of accelerated events. For more information about creating triggers, refer to the *IBM Tivoli Netcool/OMNIbus Administration Guide* available from the following web site:

https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/OMNIbus.

This example shows how to create a trigger to immediately notify Netcool/Impact when there is an alert with a severity of 5:

```
create or replace trigger ft_insert1
group trigger_group1
priority 1
after insert on alerts.status
for each row
begin
        if (new.Severity >= 5)
        then
                iduc evtft 'default' , insert, new
        end if;
end;
```

Another example shows how to create a trigger that sends an accelerated event to Netcool/Impact when an event with Customer internet_banking is deleted:

```
create or replace trigger ft_delete1
group trigger_group1
priority 1
before delete on alerts.status
for each row
begin
        if (old.Customer = 'internet_banking')
        then
                iduc evtft 'default' , delete, old
        end if;
end;
```

The following example shows how to create a trigger that immediately notifies Netcool/Impact if a reinsert of the event with the Node as ‘New York' is received:

```
create or replace trigger ft_reinsert1
group trigger_group1
priority 1
after reinsert on alerts.status
for each row
begin
        if (new.Node = 'New York')
        then
                iduc evtft 'default' , insert, new
        end if;
end;
```

The following example shows how to create a signal trigger that notifies you when a gateway connection is established with the objectserver:

```
create or replace trigger notify_isqlconn
group trigger_group1
priority 1
on signal connect
begin
if( %signal.process = 'GATEWAY' )
then
iduc sndmsg 'default', 'Gateway Connection from '
+ %signal.node + ' from user ' + %signal.username + ' at ' +
to_char(%signal.at)
end if;
end;
```

Yet another example shows how to create a signal trigger that notifies you when connection gets disconnected:

```
create or replace trigger notify_isqldisconn
 group trigger_group1
 priority 1
 on signal disconnect
 begin if( %signal.process = 'isql' )
 then iduc sndmsg 'default', 'ISQL Disconnect from ' + %signal.node +
 ' from user ' + %signal.username + ' at ' + to_char(%signal.at)
 end if;
 end;
```

Using the ReturnEvent function

You can use the ReturnEvent function to insert, update, or delete events that Netcool/Impact receives from Netcool/OMNIbus. To read more about the ReturnEvent function, refer to the Policy Reference Guide.

This example shows how to use the ReturnEvent function to set the Node to Impacted and to increment the Severity by 1:

```
@Node = 'Impacted';
@Severity = @Severity + 1;
ReturnEvent(EventContainer);
```

Another example shows how to delete the event from alerts.status using the ReturnEvent function:

```
@DeleteEvent = TRUE;
ReturnEvent(EventContainer);
```

Using Spid to control which events get sent over from OMNIbus

Netcool/Impact cannot subscribe to an OMNIbus channel, but you can use the Spid instead of a channel name to control which events get sent over to Netcool/Impact.

When the OMNIbusEventListener Service starts, it prints out the details of the connection in the IMPACT_HOME/log/<*servername*>_omnibuseventlistener.log, including the connection Spid. In the following example, the Spid is 2:

```
21 Feb 2012 11:16:07,363: Initializing Service
21 Feb 2012 11:16:07,363: Connecting to the Data Source: defaultobjectserver
21 Feb 2012 11:16:07,405: Service Started
21 Feb 2012 11:16:07,522: Attempting to connect for IDUC notifications
21 Feb 2012 11:16:07,919: Established connection to the Data Source defaultobjectserver
21 Feb 2012 11:16:08,035: IDUC Connection: Established:
21 Feb 2012 11:16:08,036: Iduc Hostname : nc050094
21 Feb 2012 11:16:08,036: Iduc Port     : 60957
21 Feb 2012 11:16:08,036: Iduc Spid     : 2
```

Knowing that Netcool/Impact is connected with the Spid 2, you can use the Client ID, and configure the trigger to send the Accelerated Event Notification only to the client with Spid=2 (Impact). An OMNIbus trigger has the following syntax:

```
IDUC EVTFT destination, action_type, row
```

where:

- *destination* = spid | iduc_channel
  - *spid* = integer_expression (The literal client connection ID)
  - *iduc_channel* = string_expression (Channel name)
- *action_type* = INSERT | UPDATE | DELETE
- *row* = variable (Variable name reference of a row in the automation)

For example, the following trigger would tell OMNIbus to send notifications only to Spid=2, which in this case is Netcool/Impact:

```
create or replace trigger ft_insert1
group trigger_group1
priority 1
after insert on alerts.status
for each row
begin
if (new.Severity >= 5)
then
iduc evtft 2 , insert, new
end if;
end;
```

For more information about OMNIbus triggers and accelerated event notification, see the *"OMNIbus Administration Guide"*.

Chapter 12. Working with other services

This chapter contains information about working with other Netcool/Impact services.

OMNIbus event listener service

The OMNIbus event listener service is used to integrate with Netcool/OMNIbus and receive immediate notifications of fast track events.

The OMNIbus event listener is used to get fast track notifications from OMNIbus using the Accelerated Event Notification feature of OMNIbus. It receives notifications through the Insert, Delete, Update, or Control (IDUC) channel. To set up the Netcool/OMNIbus event listener, you must set its configuration properties using the GUI. The configuration properties allow you to specify one or more policies that are to be run when the OMNIbus event listener receives incoming events from Netcool/OMNIbus. For more information about OMNIbus triggers and accelerated event notification, see the *OMNIbus Administration Guide*.

Important:

- The OMNIbus event listener service works only with OMNIbus 7.3 to monitor ObjectServer events.
- If the Impact Server and OMNIbus server are located in different network domains, for the OMNIbus event listener service to work correctly, you must set the **`Iduc.ListeningHostname`** property in the OMNIbus server. This property must contain the IP address or fully qualified hostname of the the OMNIbus server. For more information about this property, refer to the OMNIbus documentation.

Setting up the OMNIbus event listener service

Use this procedure to create the OMNIbus event listener service.

Procedure

1. In the Tivoli Integrated Portal, in the navigation tree, click **System Configuration** > **Event Automation** > **Services**, to open the **Services** tab.
2. If required, select a cluster from the **Cluster** list.
3. Click the **Create New Service** icon in the toolbar and select **OMNIbusEventListener** to open the configuration window.
4. Enter the required information in the configuration window.
5. Click the **Save** icon in the toolbar to create the service.
6. Start the service to establish a connection to the ObjectServer and subscribe to the IDUC channel to get notifications for inserts, updates, and deletes.

Using the OMNIbus event listener service

Starting the OMNIbus event listener service establishes a connection between Netcool/Impact and the objectserver.

To ensure that the OMNIbus event listener service started successfully, check the service logs. A message like the following example is displayed if the service started:

```
Initializing Service
Connecting to the Data Source: defaultobjectserver
IDUC Connection: Established:
Iduc Hostname : swordfish
Iduc Port : 35920
Iduc Spid : 2
Registered for the IDUC Feed
Service Started
```

Triggers

You must create triggers before Netcool/Impact can receive accelerated events from Netcool/OMNIbus.

Triggers notify Netcool/Impact of accelerated events. For more information about creating triggers, refer to the *IBM Tivoli Netcool/OMNIbus Administration Guide* available from the following web site:

https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/OMNIbus.

This example shows how to create a trigger to immediately notify Netcool/Impact when there is an alert with a severity of 5:

```
create or replace trigger ft_insert1
group trigger_group1
priority 1
after insert on alerts.status
for each row
begin
        if (new.Severity >= 5)
        then
                iduc evtft 'default' , insert, new
        end if;
end;
```

Another example shows how to create a trigger that sends an accelerated event to Netcool/Impact when an event with Customer internet_banking is deleted:

```
create or replace trigger ft_delete1
group trigger_group1
priority 1
before delete on alerts.status
for each row
begin
        if (old.Customer = 'internet_banking')
        then
                iduc evtft 'default' , delete, old
        end if;
end;
```

The following example shows how to create a trigger that immediately notifies Netcool/Impact if a reinsert of the event with the Node as ‘New York' is received:

```
create or replace trigger ft_reinsert1
group trigger_group1
priority 1
after reinsert on alerts.status
for each row
begin
        if (new.Node = 'New York')
        then
                iduc evtft 'default' , insert, new
        end if;
end;
```

The following example shows how to create a signal trigger that notifies you when a gateway connection is established with the objectserver:

```
create or replace trigger notify_isqlconn
group trigger_group1
priority 1
on signal connect
begin
if( %signal.process = 'GATEWAY' )
then
iduc sndmsg 'default', 'Gateway Connection from '
+ %signal.node + ' from user ' + %signal.username + ' at ' +
to_char(%signal.at)
end if;
end;
```

Yet another example shows how to create a signal trigger that notifies you when connection gets disconnected:

```
create or replace trigger notify_isqldisconn
 group trigger_group1
 priority 1
 on signal disconnect
 begin if( %signal.process = 'isql' )
 then iduc sndmsg 'default', 'ISQL Disconnect from ' + %signal.node +
 ' from user ' + %signal.username + ' at ' + to_char(%signal.at)
 end if;
 end;
```

Using the ReturnEvent function

You can use the ReturnEvent function to insert, update, or delete events that Netcool/Impact receives from Netcool/OMNIbus. To read more about the ReturnEvent function, refer to the Policy Reference Guide.

This example shows how to use the ReturnEvent function to set the Node to Impacted and to increment the Severity by 1:

```
@Node = 'Impacted';
@Severity = @Severity + 1;
ReturnEvent(EventContainer);
```

Another example shows how to delete the event from alerts.status using the ReturnEvent function:

```
@DeleteEvent = TRUE;
ReturnEvent(EventContainer);
```

Policy activator service

The policy activator service activates policies at startup or at the intervals you specify for each selected policy.

This is a default service that you can use instead of creating your own, or in addition to creating your own.

Policy activator configuration

In a policy activator you can configure the policy activator name, the activation interval, the policy you want to run at intervals, and the start up and logging options.

Policy activator service configuration window

Use this information to configure the policy activator service.

Table 14. Create New Policy Activator Service configuration window

Window element	Description
Service name	Type a unique name to identify the service.
Activation Interval	Select how often (in seconds) the service must activate the policy. The minimum value is `0`; the default value is `10`.
Policy	Select the policy you want the policy activator to run.
Startup: Automatically when server starts	Select to automatically start the service when the server starts. You can also start and stop the service from the GUI.
Service log: Write to file	Select to write log information to a file.

Policy logger service

The policy logger service is responsible for managing the policy log.

The log is a text stream used to record messages generated during the runtime of a policy. The log contains both Netcool/Impact system messages and messages that you create when you write a policy. The policy logger service specifies an error-handling policy to activate when an error occurs during the execution of a policy. It also specifies the logging levels for debugging policies and which items must be logged. When you configure this service, you select a policy to handle the errors as they occur.

Policy logger configuration

You can configure the following properties of the policy logger.

- Error handling policy
- Highest log level
- Logging of SQL statements
- Logging of pre-execution function parameters
- Logging of post-execution function parameters
- Policy profiling
- Logging and reporting options

Policy logger service configuration window

Use this information to configure the policy logger service.

Table 15. Policy Logger Service configuration window

Window element	Description
Error-handling Policy	The error handling policy is the policy that is run by default when an error is not handled by an error handler within the policy where the error occurred.

Table 15. Policy Logger Service configuration window (continued)

Window element	Description
Highest Log Level	You can specify a log level for messages that you print to the policy log from within a policy using the `Log` function. When a `log()` statement in a policy is processed, the specified log level is evaluated against the number that you select for this field. If the level specified in this field is greater than or equal to the level specified in the policy `log()` statement, the message is recorded in the policy log.
Log what	Select what you want to appear in the log: • All SQL statements. Select to print all the contents of all SQL statements made in calls to SQL database data sources. Logging SQL statements can help you debug a policy that uses external data sources. • Pre-execution Action Module Parameters. Select to print the values of all the parameters passed to a built-in action function before the function is called in a policy. These parameters include the values of built-in variables such as `DataItems` and `DataItem`. • Post-execution Action Module Parameters • All Action Module Parameters
Policy Profiling: Enable	Select to enable policy profiling. Policy profiling calculates the total time that it takes to run a policy and prints this time to the policy log You can use this feature to see how long it takes to process variable assignments and functions. You can also see how long it takes to process an entire function and the entire policy.
Service log: Write to file	Select to write log information to a file. You can also enable the collecting of report information through the service.
Append Thread Name to Log File Name	Select this option to name the log file by appending the name of the thread to the default log file name.
Append Policy Name to Log File Name	Select this option to name the log file by appending the name of the policy to the default log file name.
Collect Reports: Enable	Select to enable data collection for the Policy Reports. If you choose to enable the **Collect Reports** option, reporting related logs are written to the policy logger file only when the log level is set to `3`. To see reporting related logs for a less detailed logging level for example, log level 1, the `NCHOME/impact/etc/<servername>_policylogger.props` file can be customized by completing the following steps: 1. Add `impact.policylogger.reportloglevel=1` to the `NCHOME/impact/etc/<servername>_policylogger.props` property. 2. Restart the Impact Server to implement the change.

Hibernating policy activator service

The hibernating policy activator service monitors hibernating policies and awakens them at specified intervals.

You use the hibernating policy activator with X events in Y time solutions and similar solutions that require the use of hibernating policies. When you configure this service, you specify how often the service reactivates hibernating policies waiting to be activated. It can be a specific period or absolute time that you have defined.

Hibernating policy activator configuration

In the hibernation policy activator you can configure the wakeup interval, and the start up and logging options.

Hibernating policy activator configuration window

Use this information to configure the hibernating policy activator.

Table 16. Hibernating Policy Activator service configuration window

Window element	Description
Polling Interval	Select a polling time interval (in seconds) to establish how often you want the service to check hibernating policies to see whether they are due to be woken up. The default value is 3 seconds.
Process wakes up immediately	Select to run the policy immediately after wake-up. The wakeup interval is the interval in seconds at which the hibernating policy activator checks hibernating policies in the internal data repository to see if they are ready to be woken.
Startup: Automatically when server starts	Select to automatically start the service when the server starts. You can also start and stop the service from the GUI.
Service log: Write to file	Select to write log information to a file.
Clear All Hibernations: Clear	Should it become necessary, click to clear all hibernating policies from the Impact Server.

Command execution manager service

The command execution manager is the service responsible for operating the command and response feature.

The service queues `JRExecAction` function calls to run external commands. The command execution manager only allows you to specify whether to print the service log to a file. There are no other configuration properties.

Command execution manager service configuration window

You can configure the command execution manager service to print the service log to a file.

Command line manager service

Use the command-line manager service to access the Impact Server from the command line to configure services parameters as well as start and stop services.

When you configure this service, you specify the port to which you connect when you use the command line. You can also specify whether you want the service to start automatically when the Impact Server starts. The command-line manager is the service that manages the CLI. You can configure the port where the command-line service runs, and the startup and logging options for the service.

Command line manager service configuration window

Use this information to configure the command line manager service.

Table 17. Command Line Manager Service Configuration window

Window element	Description
Port	Select a port number where you want to run the service from the list or type the number. You telnet to this port when you use the CLI. The default is `2000`.
Startup: Automatically when server starts	Select to automatically start the service when the server starts. You can also start and stop the service from the GUI.
Service log: Write to file	Select to write log information to a file.

Chapter 13. Working with policies

A policy is a set of operations that you want Netcool/Impact to perform.

Before you begin developing policies, you must be familiar with the policy log, policy context, and policy scope aspects of the product.

Policy language

You use the Impact Policy Language (IPL), or JavaScript to write the policies that you want Netcool/Impact to run.

The IPL is a scripting language similar in syntax to programming languages like C/C++ and Java. It provides a set of data types, built-in variables, control structures, and functions that you can use to perform a wide variety of event management tasks. It also allows you to create your own variables and functions, as in other programming languages.

JavaScript a scripting programming language commonly used to add interactivity to web pages. It can also be used in browser environments. JavaScript uses the same programming concepts that are used in IPL to write policies. For more information about JavaScript syntax, see http://www.w3schools.com/js/default.asp.

Policy log

The policy log is a text stream that records messages created during the runtime of a policy.

The messages in the policy log provide information about the system status and about any exceptions that might occur. You can write custom messages to the log from within a policy using the `Log` function.

Policy context

The policy context is the set of all the variables whose values are assigned in the current policy.

The policy context includes built-in variables such as `EventContainer` as well as the variables that you define. You can access the value of this context from within a policy using the CurrentContext function. This function returns a string that contains the names and current value of all the variables in the policy.

Policy scope

The scope of all variables in a policy is global.

This means that everywhere you use a function, it will reference the same value, regardless of whether you use it in the main program body or within a user-defined function.

Printing to the policy log

Printing messages to the policy log is one of the most useful capabilities of Netcool/Impact when it comes to testing and debugging policies.

You print messages to the policy log using the `Log` function. The `Log` function takes the message you want to print as its input parameter.

This example is a version of the classic "Hello, World!" program used to teach developers how to program in the C programming language. In the C version, you print Hello, World! to the standard output. You are not permitted to access the standard output stream using the policy language but you can print the message to the policy log.

The policy, which consists of a single line, is as follows.

```
Log("Hello, World!");
```

Here, you simply call the `Log` function and pass the string `Hello, World!` as an input parameter. As in programming languages like C/C+ and Java, you enclose string literals in double quotation marks.

When you run the policy, it prints the following message to the policy log:

```
Hello, World!
```

User-defined variables

User-defined variables are variables that you define when you write a policy.

You can use any combination of letters and numbers as variable names as long as the first variable starts with a letter:

You do not need to initialize variables used to store single values, such as strings or integers. For context variables, you call the `NewObject` function, which returns a new context. For event container variables, you call `NewEvent`. You do not need to initialize the member variables in contexts and event containers.

The following example shows how to create and reference user-defined variables:

```
MyInteger = 1;
MyFloat = 123.4;
MyBoolean = True;
MyString = "Hello, World!";

MyContext = NewObject();
MyContext.Member = "1";

MyEvent = NewEvent();
MyEvent.Summary = "Event Summary";

Log(MyInteger + ", " + MyEvent.Summary);
```

In the example in this section, you create a set of variables and assign values to them. Then, you use the `Log` function in two different ways to print the value of the variables to the policy log.

The first way you use `Log` is to print out each of the values as a separate call to the function. The second way is to print out all the variables in the policy context at

once, using the CurrentContext function. The CurrentContext function returns a string that contains the names and values of all the variables currently defined in the policy.

```
VarOne = "One";
VarTwo = 2;
VarThree = 3.0;
VarFour = VarOne + ", " + VarTwo + ", " + VarThree;

Log(VarOne);
Log(VarTwo);
Log(VarThree);
Log(VarFour);

Log(CurrentContext());
```

When you run this policy, it prints the following message to the policy log:

```
One
2
3.0
One, Two, Three
"Prepared with user supplied parameters "=(Escalation=5, EventContainer=(),
VarTwo=Two, VarOne=One, ActionNodeName=TEMP, VarFour=One, Two, Three,
VarThree=Three, ActionType=1)
```

As shown above, you do not have to declare variables before assigning their values in the way that you do in languages like C/C++ and Java. Arrays and scalar variables like integers or strings are created automatically the first time you assign a value to them. Contexts and event containers, however, must be explicitly created using the NewObject and NewEvent functions, as described later in this guide.

Array

The array is a native data type that you can use to store sets of related values.

An array in Netcool/Impact represents a heterogeneous set of data, which means that it can store elements of any combination of data types, including other arrays and contexts. The data in arrays is stored as unnamed elements rather than as member variables.

In IPL you assign values to arrays using the curly braces notation. This notation requires you to enclose a comma-separated list of the values to assign in curly braces. The values can be specified as literals or as variables whose values you have previously defined in the policy:

```
arrayname = {element1, element2, elementn}
```

Attention: Arrays in IPL and JavaScript are zero-based, which means that the first element in the array has an index value of 0.

In JavaScript, use the square braces notation to assign array values as a comma-separated series of numeric, string, or boolean literals:

```
arrayname = [element1, element2, elementn]
```

Important: You can create an array of any size by manually defining its elements. You cannot import it from a file. You cannot have an array in an array unless it is a multi-dimensional array.

You access the value of arrays using the square bracket notation. This notation requires you to specify the name of the array followed by the index number of the

element enclosed in square brackets. Use the following syntax to access the elements of a one-dimensional array and a multi-dimensional array:

arrayname[*element index*]

arrayname[*first dimension element index*][*second dimension element index*]

Examples

Here is an example of a one-dimensional array in IPL:

```
MyArray = {"Hello, World!", 12345};
Log(MyArray[0] + ", " + MyArray[1]);
```

Here is an example of a one-dimensional array in JavaScript:

```
MyArray = ["Hello, World!", 12345];
Log(MyArray[0] + ", " + MyArray[1]);
```

It prints the following text to the policy log:

```
Hello.World!, 12345
```

Here in an example of a two-dimensional array in IPL:

```
MyArray = {{"Hello, World!", 12345}, {"xyz", 78, 7, "etc"}};
Log(MyArray[0][0] + "." + MyArray[1][0]);
```

Here in an example of a two-dimensional array in JavaScript:

```
MyArray = [["Hello, World!", 12345], ["xyz", 78, 7, "etc"]];
Log(MyArray[0][0] + "." + MyArray[1][0]);
```

It prints the following text to the policy log:

```
Hello.World!.xyz
```

This example policy in IPL, uses the same two-dimensional array and prints the label and the value of an element to the parser log:

```
MyArray = {{"Hello, World!", 12345}, {"xyz", 78, 7, "etc"}};
log("MyArray is " + MyArray);
log("MyArray Length is " + length(MyArray));
ArrayA = MyArray[0];
log("ArrayA is " + ArrayA + " Length is " + length(ArrayA));
i = 0;
While(i < length(ArrayA)) {
    log("ArrayA["+i+"] = " + ArrayA[i]);
    i = i+1;
}
ArrayB = MyArray[1];
log("ArrayB is " + ArrayB + " Length is " + length(ArrayB));
i = 0;
While(i < length(ArrayB)) {
    log("ArrayB["+i+"] = " + ArrayB[i]);
    i = i+1;
}
```

This example policy in JavaScript, uses the same two-dimensional array and prints the label and the value of an element to the parser log:

```
MyArray = [["Hello, World!", 12345], ["xyz", 78, 7, "etc"]];
log("MyArray is " + MyArray);
log("MyArray Length is " + Length(MyArray));
ArrayA = MyArray[0];
Log("ArrayA is " + ArrayA + " Length is " + Length(ArrayA));
i = 0;
while(i < Length(ArrayA)) {
    Log("ArrayA["+i+"] = " + ArrayA[i]);
```

```
    i = i+1;
}
ArrayB = MyArray[1];
Log("ArrayB is " + ArrayB + " Length is " + Length(ArrayB));
i = 0;
while(i < length(ArrayB)) {
    Log("ArrayB["+i+"] = " + ArrayB[i]);
    i = i+1;
}
```

Here is the output in the parser log:

```
ArrayA[0] = Hello World!
ArrayA[1] = 12345
```

In the following policy, you assign a set of values to arrays and then print the values of their elements to the policy log.

```
Array1 = {"One", "Two", "Three", "Four",
"Five"};
Array2 = {1, 2, 3, 4, 5};
Array3 = {"One", 2, "Three", 4, "Five"};

String1 = "One";
String2 = "Two";
Array4 = {String1, String2};

Log(Array1[0]);
Log(Array2[2]);
Log(Array3[4]);
Log(Array4[1]);

Log(CurrentContext());
```

Here, you assign sets of values to four different arrays. In the first three arrays, you assign various string and integer literals. In the fourth array, you assign variables as the array elements.

When you run the policy, it prints the following message to the policy log:

```
One
3
4
Two
"Prepared with user supplied parameters "=(String2=Two, ActionType=1,
String1=One, EventContainer=(), ActionNodeName=TEMP, Escalation=6,
Array4={One, Two}, Array3={One, 2, Three, 4, Five}, Array2={1, 2,
3, 4, 5},
Array1={One, Two, Three, Four, Five})
```

Context

Context is a data type that you can use to store sets of data.

Contexts are like the `struct` data type in C/C++. Contexts can be used to store elements of any combinations of data types, including other contexts and arrays. This data is stored in a set of variables called member variables that are "contained" inside the context. Member variables can be of any type, including other contexts.

You reference member variables using the dot notation. This is also the way that you reference member variables in a `struct` in languages like C and C++. In this notation, you specify the name of the context and the name of the member variable

separated by a period (.). You use this notation when you assign values to member variables and when you reference the variables elsewhere in a policy.

Important: A built-in context is provided, called the policy context, that is created automatically whenever the policy is run. The policy context contains all of the variables used in the policy, including built-in variables.

Unlike arrays and scalar variables, you must explicitly create a context using the `NewObject` function before you can use it in a policy. You do not need to create the member variables in the context. Member variables are created automatically the first time you assign their value.

The following example shows how to create a new context, and how to assign and reference its member variables:

```
MyContext = NewObject();
MyContext.A = "Hello, World!";
MyContext.B = 12345;

Log(MyContext.A + ", " + MyContext.B);
```

This example prints the following message to the policy log:

```
Hello, World!, 12345
```

The following policy shows how to create a context called `MyContext` and assign a set of values to its member variables.

```
MyContext
= NewObject();

MyContext.One = "One";
MyContext.Two = 2;
MyContext.Three = 3.0;

String1 = MyContext.One + ", " + MyContext.Two + ", " + MyContext.Three;

Log(String1)
```

When you run this policy, it prints the following message to the policy log:

```
One, 2, 3.0
```

If statements

You use the `if` statement to perform branching operations.

Use the `if` statement to control which statements in a policy are executed by testing the value of an expression to see if it is true. The `if` statement in the Impact Policy Language is the same as the one used in programming languages like C/C++ and Java.

The syntax for an `if` statement is the `if` keyword followed by a Boolean expression enclosed in parentheses. This expression is followed by a block of statements enclosed in curly braces. Optionally, the `if` statement can be followed by the `else` or `elseif` keywords, which are also followed by a block of statements.

```
if (condition){
    statements
} elseif (condition){
```

```
    statements
} else {
    statements
}
```

Where *condition* is a boolean expression and *statements* is a group of one or more statements. For example:

```
if (x == 0) {
    Log("x equals zero");
} elseif (x == 1){
    Log("x equals one");
} else {
    Log("x equals any other value.");
}
```

When the `if` keyword is encountered in a policy, the Boolean expression is evaluated to see if it is true. If the expression is true, the statement block that follows is executed. If it is not true, the statements is skipped in the block. If an `else` statement follows in the policy, the corresponding `else` statement block is executed.

In this example policy, you use the `if` statement to test the value of the `Integer1` variable. If the value of `Integer1` is `0`, the policy runs the statements in the statement block.

```
Integer1 = 0;

if (Integer1 == 0) {
    Log("The value of Integer1 is zero.");
}
```

When you run this policy, it prints the following message to the policy log:

```
The value of Integer1 is zero.
```

Another example shows how to use the `else` statement. Here, you set the value of the `Integer1` variable to `2`. Since the first test in the `if` statement fails, the statement block that follows the `else` statement is executed.

```
Integer1 = 2;

if (Integer1 == 1) {
    Log("The value of Integer1 is one.");
} else {
    Log("The value of Integer1 is not one.");
}
```

When you run this example, it prints the following message to the policy log:

```
The value of Integer1 is not one.
```

While statements

You use the `while` statement to loop over a set of instructions until a certain condition is met.

The `while` statement allows you to repeat a set of operations until a specified condition is true. The `while` statement in the Impact Policy Language is the same as the one used in programming languages like C, C++, and Java.

The syntax for the `while` statement is the `while` keyword followed by a Boolean expression enclosed in parentheses. This expression is followed by a block of statements enclosed in curly braces.

```
while (condition) { statements }
```

where `condition` is a boolean expression and `statements` is a group of one or more statements. For example:

```
I = 10;
while(I > 0) {
    Log("The value of I is: " + I);
    I = I - 1;
}
```

When the `while` keyword is encountered in a policy, the Boolean expression is evaluated to see if it is true. If the expression is true, the statements in the following block are executed. After the statements are executed, Netcool/Impact again tests the expression and continues executing the statement block repeatedly until the condition is false.

The most common way to use the `while` statement is to construct a loop that is executed a certain number of times depending on other factors in a policy. To use the `while` statement in this way, you use an integer variable as a counter. You set the value of the counter before the `while` loop begins and decrement it inside the loop. The While statement tests the value of the counter each time the loop is executed and exits when the value of the counter is zero.

The following example shows a simple use of the `while` statement:

```
Counter = 10;

while (Counter > 0) {
    Log("The value of Counter is " + Counter);
    Counter = Counter - 1;
}
```

Here, you assign the value of `10` to a variable named `Counter`. In the `while` statement, the policy tests the value of `Counter` to see if it is greater than zero. If `Counter` is greater than zero, the statements in the block that follows is executed. The final statement in the block decrements the value of `Counter` by one. The While loop in this example executes 10 times before exiting.

When you run this example, it prints the following message to the policy log:

```
The value of Counter is 10
The value of Counter is 9
The value of Counter is 8
The value of Counter is 7
The value of Counter is 6
The value of Counter is 5
The value of Counter is 4
The value of Counter is 3
The value of Counter is 2
The value of Counter is 1
```

The following example shows how to use the While statement to iterate through an array. You often use this technique when you handle data items retrieved from a data source.

```
MyArray = {"One", "Two", "Three", "Four"};

Counter = Length(MyArray);
```

```
while (Counter > 0) {
 Index = Counter - 1;
 Log(MyArray[Index]);
 Counter = Counter - 1;
}
```

Here, you set the value of `Counter` to the number of elements in the array. The While statement loops through the statement block once for each array element. You set the Index variable to the value of the Counter minus one. This is because arrays in IPL are zero-based. This means that the index value of the first element is `0`, rather than `1`.

When you run this example, it prints the following message to the policy log:

```
Four
Three
Two
One
```

In these examples, when you use this technique to iterate through the elements in an array, you access the elements in reverse order. To avoid doing this, you can increment the counter variable instead of decrementing it in the loop. This requires you to test whether the counter is less than the number of elements in the array inside the While statement.

The following example shows how to loop through an array while incrementing the value of the counter variable.

```
MyArray = {"One", "Two", "Three", "Four"};

ArrayLength = Length(MyArray);
Counter = 0;

while (Counter < ArrayLength) {
    Log(MyArray[Counter]);
    Counter = Counter + 1;
}
```

When you run this policy, it prints the following message to the policy log:

```
One
Two
Three
Four
```

User-defined functions

User-defined functions are functions that you use to organize your code in the body of a policy.

Once you have defined a function, you can call it in the same way as the built-in action and parser functions. Variables passed to a function are passed by reference, rather than by value. This means that changing the value of a variable within a function also changes the value of the variable in the general scope of the policy.

User-defined functions cannot return a value as a return parameter. You can return a value by defining an output parameter in the function declaration and then assigning a value to the variable in the body of the function. Output parameters are specified in the same way as any other parameter.

You can also declare your own functions and call them within a policy. User-defined functions help you encapsulate and reuse functionality in your policy.

The syntax for a function declaration is the `Function` keyword followed by the name of the function and a comma-separated list of input parameters. The list of input parameters is followed by a statement block enclosed in curly braces.

Unlike action and parser functions, you cannot specify a return value for a user-defined function. However, because the scope of variables in IPL policy is global, you can approximate this functionality by setting the value of a return variable inside the function.

Function declarations must appear in a policy before any instance where the function is called. The best practice is to declare all functions at the beginning of a policy.

The following example shows how to declare a user-defined function called `GetNodeByHostname`. This function looks up a node in an external data source using the supplied host name.

```
Function GetNodeByHostName(Hostname) {

 DataType = "Node";
 Filter = "Hostname ='" + Hostname + "'";
 CountOnly = False;

 MyNodes = GetByFilter(DataType, Filter, CountOnly);
 MyNode[0] = MyNodes;

}
```

You call user-defined functions in the same way that you call other types of functions. The following example shows how to call the function declared above.

```
GetNodeByHostName("ORA_HOST_01");
```

Here, the name of the node that you want to look up is `ORA_HOST_01`.The function looks up the node in the external data source and returns a corresponding data item named `MyNode`. For more information about looking up data and on data items, see the next chapter in this book.

Function declarations

Function declarations are similar to those in scripting languages like JavaScript. Valid function names can include numbers, characters and underscores, but cannot start with a number.

The following is an example of a user-defined function:

```
Function MyFunc(DataType, Filter, MyArray) {
    MyArray = GetByFilter(DataType, Filter, False);
}
```

Calling user-defined functions

You can call a user-defined function as follows:

```
Funcname([param1, param2 ...])
```

The following example shows a user-defined function call:

```
MyFunc("User", "Location = 'New York'", Users);
```

Examples of user-defined functions

The following example show how variables are passed to a function by reference:

```
// Example of vars by reference

Function IncrementByA(NumberA, NumberB) {
    NumberB = NumberB + NumberA;
}

SomeInteger = 10;
SomeFloat = 100.001;

IncrementByA(SomeInteger, SomeFloat);

Log("SomeInteger is now: " + SomeInteger);
// will return: IntegerA is now 10

Log("SomeFloat is now: " + SomeFloat);
// will return: FloatB is now 110.001
```

The following example shows how policies handle return values in user-defined functions:

```
// Example of no return output

Function LogTime(TimeToLog) {
    If (TimeToLog == NULL) {
        TimeToLog = getdate();
    }
    Log("At the tone the time will be: "+ localtime(TimeToLog));
}

LoggedTime = LogTime(getdate());

Log("LoggedTime = "+LoggedTime);

// will return: "LoggedTime = NULL" as nothing can be
// returned from user functions
```

Scheduling policies

You can set up Netcool/Impact to run policies at specific times.

Running policies using the policy activator

You can use a policy activator service to run one policy at specified intervals during Netcool/Impact run time.

For example, if you want to run a policy named `CHECK_SYSTEM_STATUS` every 60 minutes during the day, create a policy activator, specify the name of the policy and the time interval. Then start the service in the GUI Server. If you want to run a different policy at specific times of day or week, you must use schedules.

Running policies using schedules

You can use schedules with a policy activator to run one or more policies at specific times.

Procedure

1. Create a schedule.

The first step in setting up policies to run at specific times is to create a schedule data type in the GUI. For more information, see “Creating a schedule data type.”

2. Create an internal data type that represents each policy as a task.

   After you create the schedule data type, you must create a data type that represents each policy as a task. The task data type can be an internal data type and typically has two user-defined fields. A field that contains a descriptive name for the task and one that contains the name of the policy associated with the task. For more information, see “Creating task data types.”

3. Create task data items.

   After you create the task data type, the next step is to create a task data item for each policy that you want to schedule. For more information, see “Creating task data items” on page 103.

4. Add the tasks to the schedule.

   After you have created the task data items, the next step is to add the tasks to the schedule that you created at the beginning. This requires you to specify the task that you want to schedule and the date or time at which you want the associated policy to be run. For more information, see “Adding the tasks to the schedule” on page 103.

5. Specify time ranges for each task.

6. Write a top scheduler policy that launches the tasks.

   A top scheduler policy is a policy that is responsible for checking the schedule to see whether any other policy is currently due to be run. For more information, see “Writing a top scheduler policy” on page 104.

7. Create a policy activator and configure it to run the top scheduler.

   The policy activator runs the top scheduler policy at intervals. When the top scheduler policy runs, it checks to see if any other policies are currently "on call" and then runs them. You can configure the policy activator to run at any interval of time. For more accurate timing of scheduled policies, use smaller intervals. For more information, see “Creating a policy activator” on page 104.

8. Start the policy activator.

   To start the policy activator, click the **Start Service** icon associated with the new policy activator where it is displayed in the **Services** tab in the toolbar.

Creating a schedule data type

You can create a schedule data type in the GUI Server.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Select a cluster from the **Cluster** list. From the **Project** list, select **Global**.
3. In the **Data Model** tab, click **Schedule**, right click and select **New Data Type** to open the **New Data Type for Schedule** tab.
4. Enter a unique name for the schedule in the **Data Type Name** field.
5. Click the **Save** icon to implement to create the schedule data type.

Creating task data types

Use this procedure to create the task data type.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Select a cluster and a project from the**Cluster** and **Project** lists.
3. In the **Data Model** tab, click **Internal**, right click and select **New Data Type** to open the **New Data Type for Internal** tab.
4. Enter a unique name for the data type in the **Data Type Name** field, for example, `Tasks`.
5. Create new fields that contain a descriptive title for the task and the name of the policy as follows:
   a. Click **New Field** to open the New field window.

      Use this window to define the attributes for the data type fields.
   b. Enter a unique ID in the **ID** field, for example, `TaskName` or `PolicyName`.
   c. From the **Format** list, select **String**.

      The **Display Name** and **Description** fields in this window are optional. For fields in internal data types, the actual name and display name must always be the same as the field ID. If you leave these fields empty, they will be automatically populated with the ID value.
   d. Click **OK** to save the changes and return to the data type tab.
6. From the **Display Name Field** list, select the field that contains the task name. This display name is displayed when you browse data items in the data type. It does not otherwise affect the behavior of the data type.
7. Click the **Save** icon to implement the changes and to create the task data type.

Creating task data items

Use this procedure to create a task data item.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model tab**.
2. Select a cluster and a project from the**Cluster** and **Project** lists.
3. In the **Data Model** tab, expand the **Internal** data type, select the task data item, right click and select **View Data Items**.
4. Click the **New** icon on the menu.
5. Enter values for the `Key` field and for the task name and policy name fields that you defined when you create the tasks data type. The value for the `Key` field can be the same as the task name. However, if the data items are created in the internal data type or any other data type to be used in the schedule configuration, the `Key` field must be unique across the data type and all tasks and policies.
6. Click OK. Then click **Save** to create the data item.

Adding the tasks to the schedule

Use this procedure to add a task to the schedule.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model tab**.
2. In the **Data Model** tab, expand **Schedule** data source, select the schedule task you created, then right click and select **New Data Items** to open the **Data items: Schedule** tab.

3. Click **New** to open the **Schedule Editor**.
4. In the **Schedule Name** field, enter a name for the schedule.
5. Add a description to the **Description** field.
6. From the **Edit Members By Type** list, select the name of the task data type that you created and click **Edit**.
7. In the **Select Schedule Members** window that opens, select the tasks that you want to schedule and click **Add**.
8. Click **OK**.
9. In the **Schedule Editor** window, select the task that you want to schedule in the **Schedule Members** list.
10. Select the type of time range you want to associate with the task from the **Add New Time Range** list and then click **New** . The possible types of time ranges are `Daily`, `Weekly` and `Absolute`.
11. In the Edit Time Range window that opens, specify the time range and time zone during which you want the policy to run. The exact time at which the policy is run depends on both this time range and the frequency at which the policy activator runs the top scheduler policy. Click **OK**.
12. Click **OK** again to exit the Schedule Editor window.

Writing a top scheduler policy

A top scheduler policy is a policy that is responsible for checking the schedule to see whether any other policy is currently due to be run.

It is also responsible for launching the policy. The top scheduler policy calls the `GetScheduleMember` function and retrieves the task data item that is currently "on call." It then obtains the name of the policy associated with the task and runs it using the `Activate` function.

The following example shows a typical top scheduler policy. In this example, the name of the schedule data type is `Schedule` and the name of the schedule itself is `TasksSchedule`. The `Tasks` data type contains a field named `PolicyName` that specifies the name of the policy to run.

```
// Call GetByKey and retrieve the schedule data item that contains
// the schedule of tasks

DataType = "Schedule";
Key = "TasksSchedule";
MaxNum = 1;

Schedules = GetByKey(DataType, Key, MaxNum);

// Call GetScheduleMember and retrieve the task that is currently
// "on call"

Tasks = GetScheduleMember(Schedules[0], 0, False, GetDate());

// Call Activate and launch the policy associated with the task

Activate(Null, Tasks[0].PolicyName);
```

Creating a policy activator

Use this procedure to create the policy activator.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Services** to open the **Services** tab.

2. In the **Services** tab, click the **Create New Service** icon on the toolbar. Click **Policy Activator** to open the **New Policy Activator** tab.
3. In the **Service Name** field, enter a unique name for the policy activator.
4. In the **Activation Interval** field, enter the interval in seconds at which you want the policy activator to run the top scheduler policy
5. From the **Policy** list, select the top scheduler policy that you created.
6. Select the **Startup** check box if you want the policy activator to run automatically when the server starts.
7. Select the **Service Log** check box if you want to write the service logs to a file.
8. Click the **Save** icon on the toolbar to create the policy activator.

Chapter 14. Handling events

From within an IPL policy, you can access and update field values of incoming events; add journal entries to events; send new events to the event source; and delete events in the event source.

Events overview

An event is a set of data that represents a status or an activity on a network. The structure and content of an event varies depending on the device, system, or application that generated the event but in most cases, events are Netcool/OMNIbus alerts.

These events are generated by Netcool probes and monitors, and are stored in the ObjectServer database. Events are obtained using event readers, event listeners, and e-mail readers services.

Incoming event data using are stored using the built-in `EventContainer` variable. This variable is passed to the policy engine as part of the context when a policy is executed. When you write a policy, you can access the fields in the event using the member variables of `EventContainer`.

Event containers

The event container is a native Netcool/Impact data type used to store event data.

The event container consists of a set of event field and event state variables.

EventContainer variable

The `EventContainer` is a built-in variable that stores the field data for incoming events.

Each time an event is passed to the policy engine for processing, it creates an instance of `EventContainer`, populates the event field variables and stores it in the policy context. You can then access the values of the event fields from within the policy.

Event field variables

Event field variables are member variables of an event container that store the field values in an event.

There is one event field variable for each field in an event. The names of event field variables are the same as the event field names. For example, if an event has fields named `AlertKey`, `Node`, `Severity`, and `Summary`, the corresponding event container has event field variables with the same names.

Event state variables

Event state variables are a set of predefined member variables that you can use to specify the state of an event when you send it to the event source using the `ReturnEvent` function.

Two event state variables are used: `JournalEntry` and `DeleteEvent`. For information about using `JournalEntry`, see "Adding journal entries to events" on page 109. For information about using `DeleteEvent`, see "Deleting events" on page 110.

User-defined event container variables

User-defined event container variables are variables that you create using the `NewEvent` function.

You use these variables when you send new events to the event source, or when you want to temporarily store event data within a policy.

Accessing event fields

You can use either the dot notation or the @ notation to access the values of event fields.

Using the dot notation

You use the dot notation to access the value of event fields in the same way you access the values of member variables in a `struct` in languages like C and C++.

The following policy shows how to use the dot notation to access the value of the `Node`, `Severity`, and `Summary` fields in an incoming event and print them to the policy log:

```
Log(EventContainer.Node);
Log(EventContainer.Severity);
Log(EventContainer.Summary);
```

Using the @ notation

If you are using IPL, you can use the @ notation to access event fields.

The @ notation is shorthand that you can use to reference the event fields in the built-in `EventContainer` variable without having to spell out the `EventContainer` name. If you are using JavaScript you must use `EventContainer.Identifier`.

The following policy shows how to use the @ notation to access the value of the `Node`, `Severity`, and `Summary` fields in an incoming event and print them to the policy log:

```
Log(@Node);
Log(@Severity);
Log(@Summary);
```

Updating event fields

To update fields in an incoming event, you assign new values to event field variables in the `EventContainer`.

An event with a new value assigned to its field variable will not be updated until you call the `ReturnEvent` function.

The following examples show how to update the Summary and Severity fields in an incoming event.

```
@Summary = "Node down";
@Summary = @Summary + ": Updated by Netcool/Impact";
@Severity = 3;
@Severity = @Severity + 1;
```

Adding journal entries to events

You can use IPL and JavaScript to add journal entries to existing Netcool/OMNIbus events.

About this task

You can only add journal entries to events that exist in the ObjectServer database. You cannot add journal entries to new events that you have created using the NewEvent function in the currently running policy. Follow these steps to add a journal entry to an event.

Procedure

1. Assign the journal text to the JournalEntry variable.

   JournalEntry is an event state variable used to add new journal entries to an existing event. For more information, see "Assigning the JournalEntry variable."
2. Send the event to the event source using the ReturnEvent function.

   Call ReturnEvent and pass the event container as an input parameter, in the following manner:

   ```
   ReturnEvent(EventContainer);
   ```

Example

The following example shows how to add a new journal entry to an incoming event.

```
// Assign the journal entry text to the JournalEntry variable

@JournalEntry = 'Modified on ' + LocalTime(GetDate()) + "\r\n" +
 'Modified by Netcool\Impact.';

// Send the event to the event source using ReturnEvent

ReturnEvent(EventContainer);
```

Assigning the JournalEntry variable

JournalEntry is an event state variable used to add new journal entries to an existing event.

Netcool/Impact uses special rules for interpreting string literals assigned to JournalEntry. Text stored in JournalEntry must be assigned using single quotation marks, except for special characters such as \r, \n and \t, which must be assigned using double quotation marks. If you want to use both kinds of text in a single entry, you must specify them separately and then concatenate the string using the + operator.

To embed a line break in a journal entry, you use an \r\n string.

The following examples show how to assign journal text to the JournalEntry variable.

```
@JournalEntry = 'Modified by Netcool/Impact';
@JournalEntry = 'Modified on ' + LocalTime(GetDate());
@JournalEntry = 'Modified on ' + LocalTime(GetDate()) + "\r\n" +
 'Modified by Netcool/Impact';
```

Sending new events

Use this procedure to send new events to an event source.

Procedure

1. Create an event container using the NewEvent function

   To create an event container, you call the NewEvent function and pass the name of the event reader associated with the event source, in the following manner:

   ```
   MyEvent = NewEvent("defaulteventreader");
   ```

   The function returns an empty event container.
2. Populate the event fields by assigning values to its event field variables.

   For example:

   ```
   MyEvent.Node = "192.168.1.1";
   MyEvent.Summary = "Node down";
   MyEvent.Severity = 5;
   MyEvent.AlertKey = MyEvent.Node + ":" + MyEvent.Summary;
   ```
3. Send the event to the data source using the ReturnEvent function.

   Call the ReturnEvent function, and pass the new event container as an input parameter, in the following manner:

   ```
   ReturnEvent(MyEvent);
   ```

Example

The following example shows how to create, populate, and send a new event to an event source.

```
// Create a new event container

MyEvent = NewEvent("defaulteventreader");

// Populate the event container member variables

MyEvent.Node = "192.168.1.1";
MyEvent.Summary = "Node down";
MyEvent.Severity = 5;
MyEvent.AlertKey = MyEvent.Node + ":" + MyEvent.Summary;

// Add a journal entry (optional)

MyEvent.JournalEntry = 'Modified on ' + LocalTime(GetDate()) + "\r\n" +
 'Modified by Netcool/Impact";

// Send the event to the event source

ReturnEvent(MyEvent);
```

Deleting events

Use this procedure to delete an incoming event from the event source.

Procedure

1. Set the `DeleteEvent` variable in the event container.

   The `DeleteEvent` variable is an event state variable that you use to specify that an event is to be deleted when it is sent back to the event source. You must set the value of `DeleteEvent` to `True` in order for an event to be deleted. For example:

   ```
   @DeleteEvent = True;
   ```

2. Send the event to the event source using the `ReturnEvent` function.

   For example:

   ```
   ReturnEvent(EventContainer);
   ```

Examples of deleting an incoming event from the event source

These examples show how to delete an incoming event from the event source using IPL, and JavaScript.

- Impact Policy Language:

  ```
  // Set the DeleteEvent Variable

  @DeleteEvent = True;

  // Send the event to the event source

  ReturnEvent(EventContainer);
  ```

- JavaScript:

  ```
  // Set the DeleteEvent Variable

  EventContainer.DeleteEvent = true;

  // Send the event to the event source

  ReturnEvent(EventContainer);
  ```

Chapter 15. Handling data

You can handle data in a policy.

From within a policy you can retrieve data from a data source by filter, by key, or by link; delete, or add data to a data source; update data in a data source; and call database functions, or stored procedures.

You can access data stored in a wide variety of data sources. These include many commercial databases, such as Oracle, Sybase, and Microsoft SQL Server. You can also access data stored in LDAP data source and data stored by various third-party applications, including network inventory managers and messaging systems.

Data items

Data items are elements of the data model that represent actual units of data stored in a data source.

The structure of this unit of data depends on the category of the associated data source. For example, if the data source is an SQL database data type, each data item corresponds to a row in a database table. If the data source is an LDAP server, each data item corresponds to a node in the LDAP hierarchy.

Field variables

Field variables are member variables in a data item. There is one field variable for each data item field. Field variable names are the same as the names in the underlying data item fields. For example, if you have a data item with two fields named `UserID` and `UserName`, it will also have two field variables named `UserID` and `UserName`.

DataItem and DataItems variables

The `DataItems` variable is a built-in variable of type array that is used by default to store data items returned by `GetByFilter`, `GetByKey`, `GetByLinks` or other functions that retrieve data items. If you do not specify a return variable when you call these functions, Netcool/Impact assigns the retrieved data items to the `DataItems` variable.

The `DataItem` variable references the first item (index 0) in the `DataItems` array.

Retrieving data by filter

Retrieving data by filter means that you are getting data items from a data type where you already know the value of one or more of the fields.

When you retrieve data by filter, you are saying: "Give me all the data items in this type, where certain fields contain these values."

Filters

A filter is a text string that sets out the conditions under which Netcool/Impact retrieves the data items.

The use of filters with internal, SQL, LDAP, and some Mediator data types is supported. The format of the filter string varies depending on the category of the data type.

SQL filters

SQL filters are text strings that you use to specify a subset of the data items in an internal or SQL database data type.

For SQL database and internal data types, the filter is an SQL `WHERE` clause that provides a set of comparisons that must be true in order for a data item to be returned. These comparisons are typically between field names and their corresponding values.

Syntax

For SQL database data types, the syntax of the SQL filter is specified by the underlying data source. The SQL filter is the contents of an SQL `WHERE` clause specified in the format provided by the underlying database. When the data items are retrieved from the data source, this filter is passed directly to the underlying database for processing.

For internal data types, the SQL filter is processed internally by the policy engine. For internal data types, the syntax is as follows:

```
Field
    Operator
    Value [AND | OR | NOT (Field
    Operator
    Value) ...]
```

where *Field* is the name of a data type field, *Operator* is a comparative operator, and *Value* is the field value.

Attention: Note that for both internal and SQL data types, any string literals in an SQL filter must be enclosed in single quotation marks. The policy engine interprets double quotation marks before it processes the SQL filter. Using double quotation marks inside an SQL filter causes parsing errors.

Operators

The type of comparison is specified by one of the standard comparison operators. The SQL filter syntax supports the following comparative operators:

- >
- <
- =
- <=
- =>
- !=
- LIKE

  Restriction: You can use the LIKE operator with regular expressions as supported by the underlying data source.

The SQL filter syntax supports the AND, OR and NOT boolean operators.

Tip: Multiple comparisons can be used together with the AND, OR, and NOT operators.

Order of operation

You can specify the order in which expressions in the SQL are evaluated using parentheses.

Examples

Here is an example of an SQL filter:

```
Location = 'NYC'
Location LIKE 'NYC.*'
Facility = 'Wandsworth' AND Facility = 'Putney'
Facility = 'Wall St.' OR Facility = 'Midtown'
NodeID >= 123345
NodeID != 123234
```

You can use this filter to get all data items where the value of the `Location` field is `New York`:

```
Location = 'New York'
```

Using this filter you get all data items where the value of the `Location` field is `New York` or `New Jersey`:

```
Location = 'New York' OR Location = 'New Jersey'
```

To get all data items where the value of the Location field is `Chicago` or `Los Angeles` and the value of the `Level` field is `3`:

```
(Location = 'New York' OR Location = 'New Jersey') AND Level = 3
```

LDAP filters

LDAP filters are filter strings that you use to specify a subset of data items in an LDAP data type.

The underlying LDAP data source processes the LDAP filters. You use LDAP filters when you do the following tasks:

- Retrieve data items from an LDAP data type using `GetByFilter`.
- Retrieve a subset of linked LDAP data items using `GetByLinks`.
- Delete individual data items from an LDAP data type.
- Specify which data items appear when you browse an LDAP data type in the GUI.

Syntax

An LDAP filter consists of one or more boolean expressions, with logical operators prefixed to the expression list. The boolean expressions use the following format:

```
Attribute
    Operator
    Value
```

where *Attribute* is the LDAP attribute name and *Value* is the field value.

The filter syntax supports the =, ~=, <, <=, >, >=, and ! operators, and provides limited substring matching using the * operator. In addition, the syntax also supports calls to matching extensions defined in the LDAP data source. White

space is not used as a separator between attribute, operator, and value, and those string values are not specified using quotation marks.

For more information on LDAP filter syntax, see Internet RFC 2254.

Operators

As with SQL filters, LDAP filters provide a set of comparisons that must be true in order for a data item to be returned. These comparisons are typically between field names and their corresponding values. The comparison operators supported in LDAP filters are:

- =
- ~=,
- <
- <=
- >
- >=
- !

One difference between LDAP filters and SQL filters is that any Boolean operators used to specify multiple comparisons must be prefixed to the expression. Another difference is that string literals are not specified using quotation marks.

Examples

Here is an example of an LDAP filter:

```
(cn=Mahatma Gandhi)
(!(location=NYC*))
(&(facility=Wandsworth)(facility=Putney))
(|(facility=Wall St.)(facility=Midtown)(facility=Jersey City))
(nodeid>=12345)
```

You can use this example to get all data items where the common name value is `Mahatma Gandhi`:

```
(cn=Mahatma Gandhi)
```

Using this example you get all data items where the value of the `location` attribute does not begin with the string `NYC`:

```
(!(location=NYC*))
```

To get all data items where the value of the `facility` attribute is `Wandsworth` or `Putney`:

```
(|(facility=Wandsworth)(facility=Putney))
```

Mediator filters

You use Mediator filters with the GetByFilter function to retrieve data items from some Mediator data types.

The syntax for Mediator filters varies depending on the underlying DSA. For more information about the Mediator syntax for a particular DSA, see the DSA documentation.

Retrieving data by filter in a policy

To retrieve data by filter, you call the GetByFilter function and pass the name of the data type and the filter string.

The function returns an array of data items that match the conditions in the filter. If you do not specify a return variable, GetByFilter assigns the array to the built-in variable DataItems.

Example of retrieving data from an SQL database data type

These examples show how to retrieve data from an SQL database data type.

In the first example, you get all the data items from a data type named Node where the value of the Location field is New York and the value of the TypeID field is 012345.

Then, you print the data item fields and values to the policy log using the Log and CurrentContext functions.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";Filter = "Location = 'New York' AND TypeID = 012345";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyNodes = GetByFilter("Node", "Location = 'New York' AND TypeID = 012345", False);
Log(CurrentContext());
```

In the second example, you get all the data items from a data type named Node where the value of the IPAddress field equals the value of the Node field in an incoming event. As above, you print the fields and values in the data items to the policy log.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";
Filter = "IPAddress = '" + @Node + "'";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values.

Log(CurrentContext());
```

Make sure that you understand the filter syntax used in the sample code. When using the value of a variable inside an SQL filter string, the value must be encapsulated in single quotation marks. This is because Netcool/Impact processes the filter string in two stages. During the first stage, it evaluates the variable. During the second stage, it concatenates the filter string and sends it to the data source for processing.

A shorter version of this example is as follows:

```
MyNodes = GetByFilter("Node", "Location = '" + @Node + "'", False);
Log(CurrentContext());
```

Example of retrieving data from an LDAP data type

These examples show how to retrieve data from an LDAP data type.

In the first example, you get any data items from a data type named `User` where the value of the `cn` (common name) field is `Brian Huang`. Then, you print the data item fields and values to the policy log using the `Log` and `CurrentContext` functions.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "User";
Filter = "(cn=Brian Huang)";
CountOnly = False;

MyUsers = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyUsers = GetByFilter("User", "(cn=Brian Huang)", False);
Log(CurrentContext());
```

In the second example, you get all data items from a data type named `Node` where the value of the Location field is `New York` or `New Jersey`. As above, you print the fields and values in the data items to the policy log.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";
Filter = "(|(Location=NewYork)(Location=New Jersey))";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyNodes = GetByFilter("Node", "(|(Location=New York)(Location=New Jersey))", False);
Log(CurrentContext());
```

Example of looking up data from a Smallworld DSA Mediator data type

The following example shows how to look up data from a Smallworld DSA Mediator data type.

Smallworld is a network inventory manager developed by GE Network Solutions. Netcool/Impact provides a Mediator DSA and a set of predefined data types that allow you to read network data from the Smallworld NIS.

In this example, you get all the data items from the `SWNetworkElement` data type where the value of ne_name is `DSX1 PNL-01 (ORP)`. Then, you print the data item fields and values to the policy log using the `Log` and `CurrentContext` functions.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "SWNetworkElement";
Filter = "ne_name = 'DSX1 PNL-01 (ORP)'";
CountOnly = False;

MyElements = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyElements = GetByFilter("SWNetworkElement", \
"ne_name = 'NSX1 PNL-01 (ORP)'", False);
Log(CurrentContext());
```

Retrieving data by key

Retrieving data by key means that you are getting data items from a data type where you already know the value one or more key fields.

When you retrieve data items by key, you are saying, "Give me a certain number of data items in this type, where the key fields equal these values." Because key fields typically designate a unique data item, the number of data items returned is typically one.

Keys

A key is a special field in a data type that uniquely identifies a data item.

You specify key fields when you create a data type. The most common way to use the key field is to use it to identify a key field in the underlying data source. For more information about data type keys, see “Data type keys” on page 27.

Key expressions

The key expression is a value or array of values that key fields in the data item must equal in order to be returned.

The following key expressions are supported:

Single key expressions
A single key expression is an integer, float, or string that specifies the value that the key field in a data item must match in order to be retrieved.

Multiple key expressions
A multiple key expression is an array of values that the key fields in a data item must match in order to be retrieved. For more information, see “Multiple key expressions.”

Multiple key expressions

A multiple key expression is an array of values that the key fields in a data item must match in order to be retrieved.

Netcool/Impact determines if the key field values match by comparing each value in the array with the corresponding key field on a one-by-one basis. For example, if you have a data type with two key fields named `Key_01` and `Key_02`, and you use a key expression of `{"KEY_12345", "KEY_93832"}`, the function compares

KEY_12345 with the value of Key_01 and KEY_93832 with the value of Key_02. If both fields match the specified values, the function returns the data item. If only one field or no fields match, the data item is not returned.

Retrieving data by key in a policy

To retrieve data by key, you call the GetByKey function and pass the name of the data type and the filter string.

The function returns an array of data items that match the conditions in the filter. If you do not specify a return variable, GetByKey assigns the array to the built-in variable DataItems.

Example of returning data from a data type using a single key expression

In this example, you retrieve a data item from a data type called Node where the value of the key field is ID-00001.

Then, you print the data item fields and values to the policy log using the Log and CurrentContext functions.

```
// Call GetByKey and pass the name of the data type
// and the key expression.

DataType = "Node";
Key = "ID-00001";
MaxNum = 1;

MyNodes = GetByKey(DataType, Key, MaxNum);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyNodes = GetByKey("Node", "ID-00001", 1);
Log(CurrentContext());
```

Example of returning data by key using a multiple key expression

In this example, you retrieve a data item from a data type called Customer where the values of its key fields are R12345 and D98776.

You print the fields and values in the data items to the policy log.

```
// Call GetByKey and pass the name of the data type.
// the key expression.

Type = "Customer";
Key = {"R12345", "D98776"};
MaxNum = 1;

MyCustomers = GetByKey(Type, Key, MaxNum);
// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyCustomers = GetByKey("Customer", {"R12345", "D98776"}, 1);
Log(CurrentContext());
```

Retrieving data by link

Retrieving data by link means that you are getting data items from data types that are linked to one or more data items that you have previously retrieved.

When you retrieve data items by link, you are saying: "Give me data items in these data types that are linked to these data items that I already have." The data items that you already have are called the source data items. The data items that you want to retrieve are known as the targets.

Links overview

Links are an element of the data model that defines relationships between data items and between data types.

They can save time during the development of policies because they allow you to define a data relationship once and then reuse it several times when you need to find data related to other data in a policy. Links are an optional part of a data model. Dynamic links and static links are supported.

Retrieving data by link in a policy

To retrieve data items by link, you must first retrieve source data items using the `GetByFilter` or `GetByKey` functions.

Then, you call `GetByLinks` and pass an array of target data types and the sources. The function returns an array of data items in the target data types that are linked to the source data items. Optionally, you can specify a filter that defines a subset of target data items to return. You can also specify the maximum number of returned data items.

Example of retrieving data by link

These examples show how to retrieve data by link.

In the first example, you call `GetByFilter` and retrieve a data item from the `Node` data type whose `Hostname` value matches the `Node` field in an incoming event. Then you call `GetByLinks` to retrieve all the data items in the `Customers` data type that are linked to the `Node`. In this example, you print the fields and values in the data items to the policy log before exiting.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";
Filter = "Hostname = '" + @Node + "'";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);

// Call GetByLinks and pass the target data type,
// the maximum number of data items to retrieve and
// the source data item.

DataTypes = {"Customer"};
Filter = "";
MaxNum = "10000";
DataItems = MyNodes;

MyCustomers = GetByLinks(DataTypes, Filter, MaxNum, DataItems);
```

```
// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is:

```
MyNodes = GetByFilter("Node", "Hostname = '" + @Node + "'", False");
MyCustomers = GetByLinks({"Customer"}, "", 10000, MyNodes);
Log(CurrentContext());
```

In the second example, you use a link filter to specify a subset of data items in the target data type to return. As above, you call `GetByFilter` and retrieve a data item from the `Node` data type whose `Hostname` value matches the `Node` field in an incoming event. Then you call `GetByLinks` to retrieve all the data items in the `Customers` data type whose `Location` is `New York` that are linked to the `Node`. You then print the fields and values in the data items to the policy log before exiting.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";
Filter = "Hostname = '" + @Node + "'";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);

// Call GetByLinks and pass the target data type,
// the maximum number of data items to retrieve and
// the source data item.

DataTypes = {"Customer"};
Filter = "Location = 'New York'";
MaxNum = "10000";
DataItems = MyNodes;

MyCustomers = GetByLinks(DataTypes, Filter, MaxNum, DataItems);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is:

```
MyNodes = GetByFilter("Node", "Hostname = '" + @Node + "'", False");
MyCustomers = GetByLinks({"Customer"}, "Location = 'New York'", 10000, MyNodes);
Log(CurrentContext());
```

Adding data

Use this procedure to add a data item to a data type.

Procedure

1. Create a context using the `NewObject` function.

   The following example shows how to create a context named `MyNode`.

   ```
   MyNode = NewObject();
   ```

2. Populate the member variables in the context with data that corresponds to the values you want to set in the new data item.

   The name of each member variable must be exactly as it appears in the data type definition, as in the following example:

```
MyNode.Name = "Achilles";
MyNode.IPAddress = "192.168.1.1";
MyNode.Location = "London";
```

3. Add the data item.

   You can add the data item to the data type by calling the AddDataItem function and passing the name of the data type and the context as input parameters. The following example shows how to add the data item to a data type.

   ```
   AddDataItem("Node", MyNode);
   ```

Example of adding a data item to a data type

In this example, the data type is named User.

The User data type contains the following fields: Name, Location, and ID.

```
// Create new context.

MyUser = NewObject();

// Populate the member variables in the context.

MyUser.ID = "00001";
MyUSer.Name = "Jennifer Mehta";
MyUser.Location = "New York";

// Call AddDataItem and pass the name of the data type
// and the context.

DataType = "User";

AddDataItem(DataType, MyUser);
```

A shorter version of this example would be as follows:

```
MyUser=NewObject();
MyUser.ID = "00001";
MyUser.Name = "Jennifer Mehta";
MyUser.Location = "New York";
AddDataItem("User", MyUser);
```

Updating data

You can update single data items, and multiple data items.

To update single a data item, you must first retrieve the data from the data type using GetByFilter, GetByKey or GetByLinks. Then you can update the data item fields by changing the values of the corresponding field variables.

When you change the value of the field variables, the values in the underlying data source are updated in real time. This means that every time you set a new field value, Netcool/Impact requests an update at the data source level.

To update multiple data items in a data type, you call the BatchUpdate function and pass the name of the data type, a filter string that specifies which data items to update, and an update expression. Netcool/Impact updates all the matching data items with the specified values.

The update expression uses the same syntax as the SET clause in the UPDATE statement supported by the underlying data source. This clause consists of a comma-separated list of the fields and values to be updated.

Updating multiple data items is only supported for SQL database data types.

Example of updating single data items

In this example, you call `GetByFilter` and retrieve a data item from a data type called `Node`.

Then you change the value of the corresponding field variables.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "Node";
Filter = "Location = '" + @Node + "'";
CountOnly = False;

MyNodes = GetByFilter(DataType, Filter, CountOnly);
MyNode = MyNodes[0];

// Update the values of the field variables in MyNode
// Updates are made in real time in the data source

MyNode.Name = "Host_01";
MyNode.ID = "00001";

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
MyNodes = GetByFilter("Node", "Location = '" + @Node + "'", False);
MyNodes[0].Name = "Host_01";
MyNodes[0].ID = "00001";
Log(CurrentContext());
```

Example of updating multiple data items

In this example, you update all the data items in the `Customer` data type whose `Location` is `New York`.

The update changes the values of the `Location` and `Node` fields. Then, you retrieve the same data items using `GetByFilter` to verify the update. Before exiting, you print the data item field values to the policy log.

```
// Call BatchUpdate and pass the name of the data type,
// the filter string and an update expression

DataType = "Customer";
Filter = "Location = 'New York'";
UpdateExpression = "Location = 'London', Node = 'Host_02'";

BatchUpdate(DataType, Filter, UpdateExpression);

// Call GetByFilter and pass the name of the data type
// and a filter string

DataType = "Customer";
Filter = "Location = 'London'";
CountOnly = False;

MyCustomers = GetByFilter(DataType, Filter, CountOnly);

// Log the data item field values.

Log(CurrentContext());
```

A shorter version of this example is as follows:

```
BatchUpdate("Customer", "Location = 'New York'", "Location = 'London',
 Node = 'Host_02'");
MyCustomers = GetByFilter("Customer", "Location = 'London'", False);
Log(CurrentContext());
```

Deleting data

You can delete single data items, or multiple data items.

Before you can delete a single data item from a data type, you must first retrieve it from the data source. You can retrieve the data item using the `GetByFilter`, `GetByKey` or `GetByLinks` functions. After you have retrieved the data item, you can call the `DeleteDataItem` function and pass the data item as an input parameter.

To delete multiple data items, you call the `BatchDelete` function and pass it the name of the data type, and either a filter or the data items you want to delete. When you delete data items by filter, you are saying: "Delete all data items in this type, where certain fields contain these values."

The filter is a text string that sets out the conditions that a data item must match in order for it to be deleted. The syntax for the filter is that of an SQL `WHERE` clause that provides a set of comparisons that must be true in order for a data item to be returned. This syntax specified by the underlying data source. When Netcool/Impact goes to the data source to delete the data items, it passes this filter directly to the data source for processing.

Deleting data items by filter is only supported for SQL database data types.

You can also delete data items by passing them directly to the `BatchDelete` function as an array.

Example of deleting single data items

In this example, you delete a data item from a data type named `User` where the value of the `Name` field is `John Rodriguez`.

Because the data type (in this case) only contains one matching data item, you can reference it as `MyUsers[0]`.

```
// Call GetByFilter and pass the name of the data type
// and the filter string.

DataType = "User";
Filter = "Name = 'John Rodriguez'";
CountOnly = False;

MyUsers = GetByFilter(DataType, Filter, CountOnly);
MyUser = MyUsers[0];

// Call DeleteDataItem and pass the data item.

DeleteDataItem(MyUser);
```

A shorter version of this example is as follows:

```
MyUsers = GetByFilter("User", "Name = 'John Rodriguez'", False);
DeleteDataItem(MyUsers[0]);
```

Example of deleting data items by filter

In this example, you delete all the data items in a data type named Node, where the value of Location is New York.

```
// Call BatchDelete and pass the name of the data type
// and a filter string that specifies which data items to delete

DataType = "Node";
Filter = "Location = 'New York'";
DataItems = NULL;

BatchDelete(DataType, Filter, DataItems);
```

A shorter version of this example is as follows:

```
BatchDelete("Node", "Location = 'New York'", NULL);
```

Example of deleting data items by item

The following example shows how to delete multiple data items by passing them directly to BatchDelete.

In this example, you delete all the data items in a data type named Customer, where the value of Location is London.

```
// Call GetByFilter and pass the name of the data type
// and a filter string

DataType = "Customer";
Filter = "Location = 'New York'";
CountOnly = False

MyCustomers = GetByFilter(DataType, Filter, CountOnly);

// Call BatchDelete and pass the array
// returned by GetByFilter

BatchUpdate(DataType, NULL, MyCustomers);
```

A shorter version of this example is as follows:

```
MyCustomers = GetByFilter("Customer", "Location = 'London'", False);
BatchDelete("Customers", NULL, MyCustomers);
```

Calling database functions

You can call functions that are defined in the underlying data source of an SQL database data type.

These functions allow you to obtain such useful data as the number of rows in the database that match a specified filter. To call a database function, you call CallDBFunction and pass the name of the data type, a filter string, and the function expression. CallDBFunction then returns the results of the function.

CallDBFunction uses the same SQL filter syntax as GetByFilter and BatchDelete. Complete syntax and additional examples for SQL filters are provided in the *Policy Reference Guide*.

The following example shows how to call the database COUNT function within a policy. In this example, you count the number of data items in the Node data type, where the value of the Location field is New York. Then, you print the number of items counted to the policy log.

```
// Call CallDBFunction and pass the name of the data type,
// a filter string and the function expression.

DataType = "Node";
Filter = "Location = 'New York'";
Function = "COUNT()";

NumItems = CallDBFunction(DataType, Filter, Function);

// Print the number of counted items to the policy log.

Log(NumItems);
```

A shorter version of this example is as follows:

```
NumItems = CallDBFunction("Node", "Location = 'New York'", "COUNT()");
Log(NumItems);
```

Chapter 16. Handling hibernations

Hibernations are policies that have been temporarily put to sleep. While a policy is asleep, it is stored internally at its current state and all processing is paused until it is woken by the hibernating policy activator service or by another policy. IPL and JavaScript languages support hibernation.

Hibernations overview

The `Hibernation` data type is a system data type that stores hibernating policies. You do not typically create or modify `Hibernation` data items using the Tivoli Integrated Portal GUI. However, you can use the GUI to delete stored hibernations in the case that an error condition occurs and the hibernations are not woken by the hibernation policy activator or another policy.

An action key is a string that uniquely identifies a hibernation. When you hibernate a policy, you must specify a unique action key.

The hibernation timeout value is the number of seconds that a policy hibernates before it can be woken by the hibernating policy activator. The hibernation timeout value does not affect the time at which the hibernation can be woken by another policy.

Hibernations are designed to be used in X events in Y time solutions. This type of solution monitors an event source for a certain number of same events to occur within a time frame, it takes the designated event management action (for example, notifying an administrator of a repeating event condition).

You can put a policy into hibernation. You can also activate a hibernating policy or remove a hibernating policy from the hibernation data type. Use the `RemoveHibernation` function to remove a policy from the hibernation data type and to remove it from the hibernation queue.

Hibernating a policy

To hibernate a policy, you call the `Hibernate` function, and pass an action key and the number of seconds for it to hibernate.

The action key can be any unique string you want to use to identify the policy. Typically, you obtain this string by performing any combination of the following tasks:

- Use the value of the `Identifier` field in an incoming ObjectServer event. The ObjectServer generates a unique `Identifier` value for each event.
- Use the `Random` function to generate a random value.
- Use the `GetDate` function to generate a value based on the current system time.

Examples of hibernating a policy

The following examples show how to hibernate a policy and work with IPL and JavaScript languages.

In this example, the action key is the value of the Identifier field in an incoming ObjectServer event. This policy will hibernate for 60 seconds before being woken by the hibernating policy activator.

```
// Call Hibernate and pass an action key and the timeout
// value for the hibernation.

ActionKey = EventContainer.Identifier;
Reason = null;
Timeout = 60;

Hibernate(ActionKey, Reason, Timeout);
```

A shorter version of this policy is as follows.

```
Hibernate(EventContainer.Identifier, null, 60);
```

In this example, the action key is a combination of the current system time and a random value. This policy will hibernate for 2 minutes before being woken by the hibernating policy activator.

```
// Call Hibernate and pass an action key and the timeout
// value for the hibernation.

ActionKey = GetDate() + "_" + Random(9999);
Reason = null;
Timeout = 120;

Hibernate(ActionKey, Reason, Timeout);
```

A shorter version of this policy is as follows.

```
Hibernate(GetDate() + Random(9999), null, 120);
```

Retrieving hibernations

Retrieving hibernations is the way that you get data items from the Hibernation data type. You must retrieve a hibernation before you can wake it from within a policy or remove it.

You can retrieve hibernations by:

- Action key search
- Filter

Retrieving hibernations by action key search

About this task

You can use the GetHibernatingPolicies function to retrieve hibernations using a lexicographical search of action key values. GetHibernatingPolicies returns an array of Hibernation data items whose action keys fall within the specified start and end action keys.

The following example shows how to retrieve hibernations using an action key search. This search returns all the Hibernation data items whose action keys fall between ActionKeyAAA and ActionKeyZZZ. The example also prints the contents of the policy context to the action tree log.

```
// Call GetHibernatingPolicies and pass the start action key
// and end action key values.

StartActionKey = "ActionKeyAAA";
EndActionKey = "ActionKeyZZZ";
```

```
MaxNum = 10000;

MyHibers = GetHibernatingPolicies(StartActionKey, EndActionKey, MaxNum);

Log(CurrentContext());
```

A shorter version of this example is as follows.

```
MyHibers = GetHibernatingPolicies("ActionKeyAAA", "ActionKeyZZZ", 10000);
Log(CurrentContext());
```

Retrieving hibernations by filter

About this task

You can use the GetByFilter function to retrieve hibernations using a filter. GetByFilter returns an array of Hibernation data items whose action keys match the specified filter string. The filter is an SQL filter as defined in "Retrieving data by filter" on page 113.

The following example shows how to retrieve hibernations using GetByFilter. In this example, you retrieve the Hibernation data item whose action key is 76486467. Then, you print the contents of the current policy context to the policy log.

```
// Call GetByFilter and pass the name of the data type
// and a filter string.

DataType = "Hibernation";
Filter = "ActionKey = '76486467'";
CountOnly = false;

MyHibers = GetByFilter(DataType, Filter, CountOnly);

Log(CurrentContext());
```

A shorter version of this example is as follows.

```
MyHibers = GetByFilter("Hibernation", "ActionKey = '76486467', false);
Log(CurrentContext());
```

Waking a hibernation

To wake a hibernation, you perform the following tasks:

- Retrieve the hibernation using GetHibernatingPolicies or GetByFilter
- Call ActivateHibernation

You must also run the RemoveHibernation function to remove the policy from the hibernation queue and to free up memory resources.

Retrieving the hibernation

About this task

The first step in waking a hibernation is to retrieve it from the Hibernation data type using GetHibernatingPolicies or GetByFilter. This step is described in the previous section of this guide.

Calling ActivateHibernation

About this task

After you have retrieved the hibernation, you can call the `ActivateHibernation` function and pass the data item as an input parameter.

Example

The following example shows how to wake a hibernation. In this example, you wake a hibernation policy whose action key value is `ActionKeyABC`.

```
// Call GetHibernatingPolicies and pass the start action key
// and end action key values.

StartActionKey = "ActionKeyAAA";
EndActionKey = "ActionKeyZZZ";
MaxNum = 10000;

MyHibers = GetHibernatingPolicies(StartActionKey, EndActionKey, MaxNum);
MyHiber = MyHibers[0];

// Call ActivateHibernation and pass the Hibernation data item as
// an input parameter.

ActivateHibernation(MyHiber);
```

Removing hibernations

Use the `RemoveHibernation` function to remove a policy from the hibernation data type and to remove it from the hibernation queue. To remove a hibernation from the internal data repository, you call the `RemoveHibernation` function and pass the action key of the hibernation as an input parameter.

The following example shows how to remove a hibernation. In this example, the action key for the hibernation is `ActionKeyABC`.

```
RemoveHibernation("ActionKeyABC");
```

Chapter 17. Sending e-mail

Netcool/Impact allows you to send e-mail from within a policy.

Sending e-mail overview

You can use the feature of sending e-mails from within a policy to send e-mail notification to administrators and users when a certain event or combination of events occur.

Netcool/Impact does not provide a built-in mail server. Before you can send e-mail, you must make sure that an SMTP server is available in your environment. The Netcool/Impact e-mail sender service must also be running before a policy can successfully send e-mail.

Sending an e-mail

About this task

To send e-mail you call the `SendEmail` function and pass the following information as input parameters:

Procedure

- The e-mail address of the recipient
- The subject line text for the e-mail
- The body content of the e-mail
- The name of the e-mail sender

Results

The following example shows how to send an e-mail. In this example, you send the e-mail to the address `srodriguez@example.com`.

```
// Call SendEmail and pass the address, subject and message text
// as input parameters

Address = "srodriguez@example.com";
Subject = "Netcool/Impact Notification";
Message = EventContainer.Node + " has reported the following error condition: "
+ EventContainer.Summary;
Sender = "impact";
ExecuteOnQueue = false;

SendEmail(null, Address, Subject, Message, Sender, ExecuteOnQueue);
```

Chapter 18. Instant messaging

Instant Messaging (IM) is a network service that allows two participants to communicate through text in real time. The most widely used Instant Messaging (IM) services are ICQ, AOL Instant Messenger (AIM), Yahoo! Messenger and Microsoft Messenger. You can send and receive instant messages from within an Impact policy.

Netcool/Impact IM

Netcool/Impact IM is a feature that allows you to send and receive instant messages from within a policy. Using this feature, Netcool/Impact can monitor an IM account on any of the most widely used services for incoming messages and perform operations when specific messages are received. Netcool/Impact can also send instant messages to any other IM account. This allows you to use IM to notify administrators, operators, and other users when certain events occur in your environment.

Netcool/Impact IM uses Jabber to send and receive instant messages. Jabber is a set of protocols and technologies that provide the means for two software entities to exchange streaming data over a network. For more information, see the Jabber Web site at `http://www.jabber.org`.

Netcool/Impact IM components

Netcool/Impact has two types of services that work together with your policies to provide IM functionality. The Jabber reader service listens for incoming instant messages and then starts a specified policy when a new message is received. The Jabber service sends messages to other IM accounts.

Netcool/Impact requires access to a Jabber server in order to send and receive instant messages. A list of public Jabber servers is available from the Jabber Web site at `http://www.jabber.org/user/publicservers.php`.

Netcool/Impact IM process

The Netcool/Impact IM process has the following phases:

- Message listening
- Message sending

Message listening

During the message listening phase, the Jabber reader service listens for new messages from one or more IM accounts. When a new message is received, the Jabber reader creates a new `EventContainer` and populates it with the contents of the incoming message. Then, the Jabber reader starts the policy specified in its configuration settings and passes it the `EventContainer`. Netcool/Impact then processes the policy.

Message sending

Message sending is the phase during which Netcool/Impact sends new messages through the Jabber service. Message sending occurs during the execution of a

policy when Netcool/Impact encounters a call to the SendInstantMessage function. When Netcool/Impact processes a call to SendInstantMessage, it passes the message content, recipient and other information to the Jabber service. The Jabber service then assembles the message and sends it to a Jabber server where it is routed to the specified recipient.

Setting up Netcool/Impact IM

Before you can send and receive instant messages using a policy, you must set up the Jabber service and the Jabber reader service as described in the User Interface Guide. After you have set up these services, you can start writing instant messaging policies using the information in "Writing instant messaging policies."

Writing instant messaging policies

About this task

You can perform the following tasks with instant messages in a Netcool/Impact policy:

Procedure

- Handle incoming messages
- Send messages

Results

Handling incoming messages

About this task

When the Jabber reader receives an incoming message, it starts the policy specified in the Jabber reader service configuration and passes the contents of the message to the policy using the EventContainer variable. The policy can then handle the incoming message in the same way it handles information passed in an incoming event.

When the Jabber reader receives an incoming message, it populates the following fields in the EventContainer variable: From and Body. The From field contains the user name of the account from which the message was sent. Body contains the contents of the message. You can access the contents of these fields using either the dot notation or the @ notation.

Sending messages

About this task

You send instant messages from within a policy using the SendInstantMessage function. This function requires you to specify the recipient and the body content of the message. You can also specify a subject, a chat room ID, and whether to send the message directly or put it on the message queue for processing by the command execution manager service. For a complete description of this function, see the Policy Reference Guide.

Example

The following example shows how to send and receive instant messages using Netcool/Impact IM.

In this example, the Jabber reader service calls the policy whenever an incoming message is received. The policy then confirms receipt of the message and performs a different set of actions, depending on whether the message sender is `NetcoolAdmin` or `NetcoolOps`.

```
// Call SendInstantMessage and pass the name of the recipient and the content
// of the message as message parameters
To = @From // Recipient is sender of the original message
TextMessage = "Message receipt confirmed.";
SendInstantMessage(To, NULL, NULL, TextMessage, False);
If (@From == "NetcoolAdmin") {
  Log("Message received from user NetcoolAdmin.");
  Log("Message contents: " + @Body);

If (@From == "NetcoolOps") {
  Log("Message    received from user NetcoolOps.");
  Log("Message contents: " + @Body);
} Else {
  Log("Message received from unrecognized user.");
  Log("Message contents: " + @Body);
}
```

Chapter 19. Executing external commands

External command execution is the process of running external commands, scripts, and applications from within a policy.

External command execution overview

You can execute external commands using the JRExec server or the command and response feature. The JRExec server is a runnable component of Netcool/Impact that allows you to run external commands on the system where the Netcool/Impact server is located. Command and response is a more advanced feature that lets you run interactive and non-interactive programs on both local and remote systems.

You can execute any type of external command that can be started from a command line. Including operating system commands, shell scripts, and many other types of applications.

JRExec server

You use the JRexec server to run external commands, scripts, and applications from within a policy.

Overview of the JRExec server

The JRExec server is a runnable server component of Netcool/Impact that you use to run external commands, scripts, and applications from within a policy, on the same system where Netcool/Impact is installed.

The JRExec server is automatically installed when you install Netcool/Impact. On Windows systems, you must also manually add the JRExec server as a Windows service. You run the JRExec server either using the JRExec server script or through the services administration tools, depending on the operating system. The server is configured through a property file.

You use the JRExecAction function to run external commands from within a policy. For more information about the JRExecAction function, see the *Policy Reference Guide*.

Starting the JRExec server

Use this procedure to start the JRExec server.

Procedure

- On UNIX systems you use the JRExec Server startup script, `nci_jrexec`, located in the `$IMPACT_HOME/bin` directory.

  Run this command in the terminal:

  ```
  ./nci_jrexec
  ```

- On Windows systems you start the JRExec server service, in the Services management console.

  Right-click **Netcool JRExec Server** in the Services window that opens, and select **Start**.

Stopping the JRExec server

Use this procedure to stop the the JRExec server.

Procedure

- On Windows systems you stop the JRExec server service, in the Services management console.

  Right-click **Netcool JRExec Server** in the Services window that opens, and select **Stop**.
- On UNIX systems you must manually terminate the process.

  Two processes are associated with the JRExec Server: the nci_jrexec process, and a JAVA process that was started by the nci_jrexec process.

  1. Obtain their IDs using these commands:
     - `ps -eaf | grep nci_jrexec`

       This command returns the PID of the nci_jrexec process.
     - `ps -eaf | grep java`

       Apart from the Impact Server, and the GUI Server process ID, this command should return this process:

       ```
       501      16053     1  1 13:58 pts/2
       00:00:02 /home/netcool_usr/IBM/tivoli/tipv2/java/bin/java
       -Dibm.tivoli.impact.propertiesDir=/home/netcool_usr/IBM/tivoli/impact/etc
        -Dbase.directory=/home/netcool_usr/IBM/tivoli/impact
        -Dnetcool.productname=impact
       -classpath /home/netcool_usr/IBM/tivoli/impact/lib/nciJmxClient.jar:
       /home/netcool_usr/IBM/tivoli/tipv2/lib/ext/log4j-1.2.15.jar
        com.micromuse.response.client.RemoteJRExecServerImpl
       ```

       This is only an example and the PID, and the path of the process. They will be different on your system.
  2. Kill both processes using this command:

     `kill -9 pid`

     where *pid* is one of the two process IDs associated with the JRExec Server.

The JRExec server configuration properties

The JRExec server properties file, `jrexecserver.props`, is located in the `$IMPACT_HOME/etc` directory.

The file may contain the following properties:

impact.jrexecserver.port

To change the port number used by the JRExec server. Default is 1345. If you change this property, you must also update the value of the `impact.jrexec.port` property in the *NCI*`_server.props` file, where *NCI* is the name of the Impact Server instance.

impact.jrexecserver.logfile

To enable logging for the JRExec server. Set its value to the path and filename of the target JRExec server log file. For example, `impact.jrexecserver.logfile=/opt/IBM/tivoli/impact/logs/jrexecserver.log`.

JRExec server logging

To enable logging for the JRExec server, add the **`impact.jrexecserver.logfile`** property to the JRExec Server properties file.

`impact.jrexecserver.logfile=`*`filename`*

Where *filename* is the path and filename of the target JRExec server log file.

The JRExec server properties file, `jrexecserver.props`, that is located in the `$IMPACT_HOME/etc/` directory.

Running commands using the JRExec server

To run a command using the JRExec server, you call the JRExecAction function and pass the name of the command and any command-line arguments as input parameters.

You can also pass a value that specifies whether you want the JRExec server to wait for the command to be completed before executing any other commands, or to continue processing without waiting.

The following example shows how to run an external command using the JRExec server. In this example, you send a page to an administrator using a paging application named `pageit` that is installed in the `/opt/pager/bin` directory on the system. The `pageit` application takes the phone number of the person paged and the return contact number as command-line arguments. In this application, the JRExec server waits for the application to finish before continuing to process any other commands.

```
// Call JRExecAction and pass the command string and
// other parameters

Command = "/opt/pager/bin/pageit";
Args = {"2125551212", "2126353131"};
ExecuteOnQueue = False;
Timeout = 60;

JRExecAction(Command, Args, ExecuteOnQueue, Timeout);
```

Using CommandResponse

Command and response is an advanced feature that lets you run interactive and non-interactive programs on both local and remote systems.

You can invoke this feature within a policy using the `CommandResponse` function. For more information about the syntax of the function, see *CommandResponse* in the Policy Reference Guide.

Chapter 20. Handling strings and arrays

This chapter contains information about handling strings and arrays in a policy.

Handling strings

The Netcool/Impact policy language allows you to manipulate strings in various ways. You can perform the following tasks with strings:

- Concatenate strings
- Find the length of a string
- Split a string into substrings
- Extract a substring from another string
- Replace a substring in a string
- Strip a substring from a string
- Trim white space from a string
- Change the case of a string
- Encrypt and decrypt strings

Concatenating strings

About this task

To concatenate strings, you use the addition operator (+). You can concatenate two strings or multiple strings at once. You can also concatenate a string with a numeric value.

The following example shows how to concatenate strings.

```
String1 = "This";
String2 = "is a test";
String3 = String1 + " " + String2;

Log(String3);

String4 = "The value of X is" + 5;

Log(String4);
```

When you run this example, it prints the following messages to the policy log:

```
This is a test
The value of X is 5
```

Finding the length of a string

About this task

You can find the length of a string using the `Length` function. The `Length` function returns the number of characters is any text string.

The following example shows how to use the `Length` function.

```
NumChars = Length("This is a test.");
Log(NumChars);
```

When you run this example, it prints the following message to the policy log:

```
15
```

Splitting a string into substrings

About this task

You can split a string into substrings using the `Split` function. The `Split` function takes a string and a set of delimiter characters as input parameters. It returns an array in which each element is a substring.

The following example shows how to use the `Split` function.

```
MyString = "One, Two, Three, Four.";
Delimiters = ",.";

MyArray = Split(MyString, Delimiters);

Count = Length(MyArray);

While (Count > 0) {
 Index = Count - 1;
 Log(MyArray[Index]);
 Count = Count - 1;
 }
```

When you run this example, it prints the following message to the policy log:

```
Four
Three
Two
One
```

Extracting a substring from another string

About this task

You can extract a substring from another string in the following ways:

Procedure

- Using the word position
- Using regular expression matching

Results

Extracting a substring using the word position

To extract a substring using the word position, you call the `Extract` function, and pass the string and the word position of the substring.

The following example shows how to extract a string in this way.

```
MyString = "This is a test.";
MySubstring = Extract(MyString, 2);
Log(MySubstring);
```

When you run this example, it prints the following message to the policy log:

```
is
```

Extracting a substring using regular expression matching

You can use regular expression matching to retrieve a single substring or all substrings from a string.

To extract a single substring, you use the `RExtract` function. The `RExtract` function takes a string and a regular expressions pattern as input parameters. It returns the first matching substring it finds in the string.

To extract all matching substrings, you use the `RExtractAll` function. As with `RExtract`, The `RExtractAll` function takes a string and a regular expressions pattern as input parameters. It returns an array that contains all the matching substrings.

Replacing a substring in a string

About this task

You can replace a substring in a string using the `Replace` function. The `Replace` function takes the string, the substring to replace and its replacement as input parameters. It returns the string after the replacement has been made.

The following example shows how to replace a substring.

```
MyString = "This is a test.";
Substring1 = "is a";
Substring2 = "is not a";

MyString = Replace(MyString, Substring1, Substring2);

Log(MyString);
```

When you run this example, it prints the following message to the policy log:

```
This is not a test.
```

Stripping a substring from a string

About this task

You can strip a substring from a string using the `Strip` function. The `Strip` function takes the string and the substring you want to strip as input parameters. It returns the string after the substring has been removed.

The following example shows how to strip a substring from a string.

```
MyString = "This is not a test.";
Substring = " not";

MyString = Strip(MyString, Substring);

Log(MyString);
```

When you run this example, it prints the following message to the policy log:

```
This is a test.
```

Trimming white space from a string

About this task

You can trim leading and trailing white space from a string using the `Trim` function. The `Trim` function takes the string as an input parameter and returns it without any leading or trailing white space.

The following example shows how to trim the white space from a string.

```
MyString = "     This is a test.     ";
MyString = Trim(MyString);
Log(MyString);
```

When you run this example, it prints the following message to the policy log:

```
This is a test.
```

Changing the case of a string

About this task

You can change the case of a string to all lowercase using the `ToLower` function.

The following example shows how to change a string to lowercase.

```
Log(ToLower("THIS IS A TEST.");
```

When you run this example, it prints the following message to the policy log:

```
this is a test.
```

You can change the case of a string to all uppercase using the `ToUpper` function.

The following example shows how to change a string to uppercase.

```
Log(ToUpper("this is a test.");
```

When you run this example, it prints the following message to the policy log:

```
THIS IS A TEST.
```

Encrypting and decrypting strings

About this task

The policy language provides a feature that allows you to encrypt and decrypt strings. This feature is useful if you want to handle password data within a Netcool/Impact policy.

You can encrypt a string using the `Encrypt` function. This function takes the string as an input parameter and returns an encrypted version.

The following example shows how to encrypt a string.

```
MyString = Encrypt("password");
```

You can decrypt a string that you have previously encrypted using the Decrypt function. This function takes an encrypted string as an input parameter and returns the plaintext version.

The following example shows how to decrypt a string.

```
MyString = Decrypt("AB953E4925B39218F390AD2E9242E81A");
```

Handling arrays

The Netcool/Impact policy language allows you to perform the following tasks with arrays:

- Find the length of an array
- Find the distinct values in an array.

Finding the length of an array

About this task

You can find the number of elements in an array using the `Length` function. The `Length` function takes the array as an input parameter and returns its number of elements.

The following example shows how to find the number of elements in an array in IPL.

```
Elements = Length({"One", "Two", "Three"});
Log(Elements);
```

The following example shows how to find the number of elements in an array in JavaScript.

```
Elements = Length(["One", "Two", "Three"];
Log(Elements);
```

When you run the example in either language, it prints the following message to the policy log:

```
3
```

Finding the distinct values in an array

About this task

You can find the distinct values in an array using the `Distinct` function. The Distinct function takes the array as an input parameter and returns another array that consists only of the unique, or non-duplicate, elements.

The following example shows how to find the distinct values in an array.

```
MyArray = {"One", "One", "Two", "Three", "Three", "Four"};
MyArray = Distinct(MyArray};
Log(MyArray);
```

When you run this example, it prints the following message to the policy log:

```
{One, Two, Three}
```

Chapter 21. Event enrichment tutorial

The goal of this tutorial is to develop an event enrichment solution to enhance the value of an existing Netcool/Impact installation.

This solution automates common tasks performed manually by the network operators and helps to integrate related business data with alerts in the Netcool OMNIbus ObjectServer.

Tutorial overview

This tutorial uses a sample environment that provides the background for understanding various event enrichment concepts and tasks.

The environment is a network operations center for a large enterprise where the company has installed and configured Netcool OMNIbus and is currently using it to manage devices on its network. The sample environment is a scaled down representation of what you might actually find in a real world operations center. It contains only the network elements and business data needed for this tutorial.

This tutorial leads you through the following steps:

- Understanding the Netcool/Impact installation
- Understanding the business data
- Analyzing the workflow in the environment
- Creating a project
- Setting up a data model
- Setting up services
- Writing an event enrichment policy
- Configuring the OMNIbus event reader to run the policy
- Running the complete solution

Understanding the Netcool/Impact installation

The first step in this tutorial is to understand the current Netcool installation. Generally, before you start developing any Netcool solution, you must find out which products in the Netcool suite you have installed and which devices, systems, or applications are being monitored in the environment.

The Netcool installation in the sample environment consists of Netcool OMNIbus and a collection of probes that monitor devices on the network. This installation uses two instances of an ObjectServer database named `NCOMS` that have been set up in a backup/failover configuration. These ObjectServers are located on host systems named `NCO_HOST_01` and `NCO_HOST_02`, and run on the default port of `4100`.

The probes in this installation monitor various network devices. The details of the devices are not important in this tutorial, but each probe sends the basic set of alert fields to the ObjectServer database, including: `Node`, `Summary`, `Severity`, `AlertKey`, `Identifier`, and `Count`

Understanding the business data

The next step in this tutorial is to understand the location and structure of the business data in your environment.

In the sample environment, the company uses instances of the Oracle database to store network inventory information, customer service information, and general organizational information about the business.

The information you want to use is stored in two databases named `ORA_01` and `ORA_02`. `ORA_01` is a network inventory database that stores information about the devices in the network, including their technical specification, facility locations, and rack numbers. `ORA_01` is located on a system named `ORA_HOST_01`. `ORA_02` is a database that contains information about the various departments in the business. `ORA_02` is located on a system named `ORA_HOST_02`. They both run on port `1521`

Analyzing the workflow

After you have found the location and structure of the business data, the next step is to analyze the current event management workflow in your environment.

The tutorial work environment is a network operations center. In this center, a number of operators are on duty at all times. They sit in an open work area and each one has access to a console that displays a Netcool OMNIbus event list. On large projector screens on one wall of the operation center are large map visualizations that provide geographical views into the current network status.

As alerts flow to the ObjectServer from the various Netcool probes and monitors installed in the environment, they appear in the event lists available to the operators. Depending on the severity of the alerts, the operators manually perform a set of tasks using the event list tools, third-party applications, and typical office tools like cell phones and e-mail.

For the sake of this tutorial, we assume that, among other tasks, the operators are performing the following actions for each alert whose severity is critical or higher. The operators:

- Manually acknowledge the alert using the event list.
- Use an in-house database tool to find information about the device causing the alert. This tool runs a query against the network inventory database and returns technical specifications, the location, and other information.
- Use another in-house tool to look up the business department being served by the device that caused the alert.
- If the business department is part of a mission critical business function, they increase the severity of the alert and update it in the ObjectServer database.

The operators might perform other actions, like looking up the administrators on call at the facility where the device is located and contacting them by phone or pager. After the problem that caused the alert has been addressed, the operators might also record the resolution in a problem log and delete the alert from the ObjectServer. For this tutorial, however, only use the workflow tasks listed.

Creating the project

About this task

After you have finished analyzing the workflow, the next step is to create a project in the Tivoli Integrated Portal GUI. You can use this project to store the data model, services, and policies used in this solution. The name of this project is `NCI_TUT_01`.

To create a project:

Procedure

1. Open the Tivoli Integrated Portal in a web browser and log in.
2. In the navigation tree, expand **System Configuration** > **Event Automation** click on one of the links, for example **Data Model**, to view the project and cluster selection lists on the **Data Model** tab.
3. Select a cluster from the **Cluster** list. From the **Project** list, select **Global**.
4. Click the **New Project** icon on the toolbar to open the New Project window.
5. Use the New Project window to configure your new project.
6. In the **Project Name** field, type `NCI_TUT_01`.
7. Click **OK** then click **Close**.

Setting up the data model

After you have created a project for this tutorial, the next step is to set up a Netcool/Impact data model. This data model consists of the event sources, data sources, and data types required by the event enrichment solution. It also consists of a dynamic link used to define the relationship between the data types.

You perform all the tasks in this step using the Tivoli Integrated Portal GUI.

To set up the data model, you perform the following tasks:

- Create the event source
- Create the data sources
- Create the data types
- Create the dynamic link

Creating the event source

About this task

The first task in setting up the data model is to create the event source. As you learned when you investigated the details of the Netcool installation, the example environment has one event source, an ObjectServer named `NCOMS`. Because you want to tap into the alerts that are stored in this ObjectServer, you must create an event source that represents it in Netcool/Impact.

An event source is a special type of data source that Netcool/Impact can use to represent a physical source of event data in the environment. Since your source of event data is an ObjectServer database, you must create an ObjectServer data source and configure it with the connection information you discovered when you investigated the details of the Netcool installation.

To create the event source:

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Select a cluster from the **Cluster** list. From the **Project** list, select **NCI_TUT_01**.
3. Click the **New Data Source** icon and select **ObjectServer** from the list. The New Data Source opens.
4. Type `NCOMS` in the **Data Source Name** field.
5. Type the name and password of an ObjectServer user in the **Username** and **Password** fields.
6. Type `NCO_HOST_01` in the **Primary Host Name** field.
7. Type `4100` in the **Primary Port** field.
8. Click **Test Connection** to test the ObjectServer connection.
9. Type `NCO_HOST_02` in the **Backup Host Name** field.
10. Type `4100` in the **Backup Port** field.
11. Click **Test Connection** to test the ObjectServer connection.
12. Click **OK**.

Creating the data sources

About this task

The next task in setting up the data model is to create the data sources. As you learned when you discovered the location and structure of the business data in your environment, the data you want to use in this solution is located in two Oracle databases named `ORA_01` and `ORA_02`. Since you want to access these databases, you must create a data source that corresponds to each one.

To create the data sources:

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Data Model** to open the **Data Model** tab.
2. Click the **New Data Source** icon and select **Oracle** from the list. The New Data Source window opens.
3. Type `ORACLE_01` in the **Data Source Name** field.
4. Type an Oracle user name and password in the **Username** and **Password** fields.
5. Type `ORA_HOST_01` in the **Primary Host Name** field.
6. Type `1521` in the **Primary Port** field.
7. Type `ORA_01` in the **SID** field.
8. Click **Test Connection** to test the ObjectServer connection.
9. Click **OK**.

Results

Repeat these steps to create another data source that corresponds to the `ORA_02` database. Name this data source `ORACLE_02`.

Creating the data types

About this task

The next task in setting up the data model is to create the data types. As you learned when you discovered the location and structure of the business data in your environment, the data that you want to use is contained in two tables.

The first table is called `Device` and is located in the `ORA_01` database. This table contains information about each device on the network. Columns in this table include `Hostname`, `DeviceID`, `HardwareID`, `Facility`, and `RackNumber`.

The second table is called `Department` and is located in the `ORA_02` database. This table contains information about each functional department in the business. Columns in this table include `DeptName`, `DeptID`, and `Location`.

Since you want to access the data in both of these tables, you must create a data type for each. Name these data types `Device` and `Department`.

To create the data types:

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Data Model** to open the **Data Model** tab.
2. Select `Oracle_01` from the data sources list.
3. Click the **New Data Type** icon.

   A new Data Type Editor tab opens.
4. Type `Device` in the **Data Type Name** field.
5. Type `ORACLE_01` in the **Data Source Name** field.
6. Select the **Enabled** option.
7. Scroll down the Data Type Editor tab so that the **Table Description** area is visible.
8. Select `Device` from the **Base Table** list.
9. Click **Refresh**.

   Netcool/Impact queries the Oracle database and populates the **Table Description** browser with the names of each column in the `Device` table.
10. Specify that the `DeviceID` field is the key field for the data type by selecting the **Key** option in the **DeviceID** row.
11. Select `Hostname` from the **Display Name Field** list.
12. Click **Save** in the Data Type Editor tab.
13. Click **Close** in the Data Type Editor tab.

Results

Repeat these steps to create another data type that corresponds to the `Department` table in the `ORA_02` database. Name this data type `Department`.

Creating a dynamic link

About this task

The next step is to create a dynamic link between the `Device` and `Department` data types.

One property of the business data that you are using in this solution is that there is a relationship between devices in the environment and departments in the business. All the devices that reside in a certain facility serve the business departments in the same location. You can make this relationship part of the data model by creating a dynamic link between the `Device` and `Department` data types. Once you have created the dynamic link, you can traverse it within a policy using the `GetByLinks` function.

In this relationship, `Device` is the source data type and `Department` is the target data type. When you create the link between the two data types, you can define it using the following syntax:

```
Location = '%Facility%'
```

This filter tells Netcool/Impact that `Device` data items are linked to `Department` data items if the value of the `Location` field in the `Department` is equal to the value of the `Facility` field in the `Device`.

To create the dynamic link:

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Data Model** to open the **Data Model** tab.
2. Click the name of the `Device` data type.

   A new Data Type Editor tab opens in the Main Work panel of the GUI. This editor displays configuration information for the `Device` data type.
3. Select the **Dynamic Links** tab in the editor.

   The **Links From This Data Type** area opens in the editor.
4. Click the **New Link By Filter** button to open the Link By Filter window.
5. Select `Department` from the **Target Data Type** list.
6. In the **Filter ...** Field, type the filter string that defines the relationship between the `Device` and `Department` list. As noted in the description of this task above, the filter string is `Location = '%Facility%'`. This means that you want `Device` data items to be linked to `Department` data items if the `Location` field in the `Department` is the same as the `Facility` field in the `Device`.
7. Click **OK**.
8. Click the **Save** button in the Data Type Editor tab.
9. Click the **Close** button in the Data Type Editor tab.

Reviewing the data model

About this task

After you have created the dynamic links, you can review the data model using the Tivoli Integrated Portal GUI to verify that you have performed all the tasks correctly. You can review the data model by opening the Data Source and Data Type task panes in the Navigation panel, and by making sure that the event source, data sources, and data types that you created are visible.

Setting up services

The next step in this tutorial is to set up the OMNIbus event reader required by the solution.

Creating the event reader

About this task

The OMNIbus event reader for this solution must check the NCOMS ObjectServer every 3 seconds and retrieve any new events.

To create the event reader:

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Services** to open the **Services** tab.
2. Click the **Create New Service** icon and select **Database EventReader** from the list.
3. Type **TUT_READER_01** in the **Service Name** field.
4. Select **NCOMS** from the **Data Source**list.
5. Type 3000 in the **Polling Interval** field.
6. Select the **Startup** option. This option specifies whether the service starts automatically when you run Netcool/Impact.
7. Click **OK**.

Reviewing the services

About this task

After you have created the event reader, you can use the Tivoli Integrated Portal GUI to verify that you have performed all the tasks correctly. To review the service that you created, click the Services task pane in the Navigation panel, and make sure that the TUT_READER_01 OMNIbus event reader is visible. You can also check to make sure that the event reader appears in the Service Status panel.

Writing the policy

After you have set up the OMNIbus event reader service, the next step is to write the policy for the solution. This policy is named EnrichEvent and it automatically performs the tasks that you discovered when you analyzed the workflow in the environment.

The EnrichEvent policy performs the following tasks:

- Look up information about the device causing the alert.
- Look up the business departments that are served by the device.
- If one of the business department is part of a mission critical business function, the policy increases the severity of the alert to critical.

This section assumes that you already know how to create, edit, and save a policy using the policy editor tools in the Tivoli Integrated Portal GUI. For more information about these tools, see the User Interface Guide.

Looking up device information

About this task

The first task that you want the policy to perform is to look up device information related to the alert in the network inventory database. Specifically, you want the

policy to retrieve technical specifications for the device causing the alert, as well as information about the facility and the rack number where the device is located.

To do this, the policy has to perform a SELECT at the database level on the table that contains the device data and return those rows that are related to the incoming alert. Viewed from the data model perspective, the policy must get data items from the Device data type where the value of the Hostname field is the same as the value of the Node field in the alert.

In order to retrieve the data items, you type the following code into the Netcool/Impact policy editor tab:

```
DataType = "Device";
Filter = "Hostname = '" + @Node + "'";
CountOnly = False;

MyDevices = GetByFilter(DataType, Filter, CountOnly);
MyDevice = MyDevices[0];

If (Length(MyDevices) < 1) { Log("No matching device found."); }
If (Length(MyDevices) > 1) { Log("More than one matching device found."); }
```

Here, GetByFilter is retrieving data items from the Device data type where the value of the Hostname field is equal to the value of the Node field in the incoming alert. The data items are stored in an array named MyDevices.

Although GetByFilter able to return more than one data items in the array, you only expect the array to contain one data item in this situation, as each device in the database has a unique Hostname. The first element of the MyDevices array is assigned to the MyDevice variable so that MyDevice can be used as shorthand later in the policy.

Because you want to retrieve only one data item from the data type, the policy also prints error messages to the policy log if GetByFilter retrieves less than or more than one.

Looking up business departments

About this task

The next task that you want the policy to perform is to look up the business departments that are served by the device that caused the alert.

When you set up the data model for this solution, you created a dynamic link. This link defined the relationship between the devices in the environment and departments in the business. To look up the business departments that are served by the device, the policy has to take the data item that it previous retrieved from the Device data type and traverse the links between it and the Department data type.

In order to retrieve the Department data items that are linked to the Device, type the following text into the policy editor below the code you entered previously:

```
DataTypes = {"Department};
Filter = NULL;
MaxNum = 10000;

MyDepts = GetByLinks(DataTypes, Filter, MaxNum, MyDevices);

If (Length(MyDepts) < 1) { Log("No linked departments found."); }
```

Here, GetByLinks retrieves up to 10,000 Department data items linked to data items in the MyDevices array. Since you are certain that the business has less than 10,000 departments, you can use a large value such as this one to make sure that all Department data items are returned.

The returned data items are stored in the MyDepts array. Because you want at least one data item from the data type, the policy also prints an error message to the policy log if GetByLinks does not return any.

Increasing the alert severity

About this task

The final task that you want the policy to perform is to increase the severity of the alert. For example, if the department that it affects has a mission critical function in the business. For the purposes of this tutorial, the departments in the business whose function are mission critical are the Data Center and Transaction Processing units.

To perform this task, the policy has to iterate through each of the Department data items retrieved in the previous step. For each Department, it must test the value of the Name field against the names of the two departments in the business that have mission critical functions. If the Department name is that of one of the two departments, the policy must increase the severity of the alert to Critical.

```
Count = Length(MyDepts);

While (Count > 0) {

    Index = Count - 1;
    MyDept = MyDepts[Index];

    If (MyDept.Name == "Data Center" || MyDept.Name == "Transaction Processing") {
        @Severity = 5;
    }

    Count = Count - 1;

}
```

Here, you use a While loop to iterate through the elements in the MyDepts array. MyDepts is the array of Department data items returned previously in the policy by a call the GetByLinks.

Before the While loop begins, you set the value of the Count variable to the number of elements in the MyDepts array. Each time the loop runs, it tests the value of Count. If Count is greater than zero, the statements inside the loop are executed. If Count is less than or equal to zero, the statements are not executed. Because Count is decremented by one each time the loop is performed, the While loop runs once for each data item in MyDepts.

A variable named Index is used to refer the current element in the array. The value of Index is the value of Count minus one, as Netcool/Impact arrays are zero-based structures whose first element is counted as zero instead of one.

Inside the loop, the policy uses an If statement to test the name of the current Department in the array against the name of the two mission-critical business departments. If the name of the current Department matches the mission-critical

departments, the policy sets the value of the Severity field in the alert to 5, which signifies a critical severity.

Reviewing the policy

About this task

After you have finished writing the policy, you can review it for accuracy and completeness. The following example shows the entire text of this policy.

```
// Look up device information

DataType = "Device";
Filter = "Hostname = '" + @Node + "'";
CountOnly = False;

MyDevices = GetByFilter(DataType, Filter, CountOnly);
MyDevice = MyDevices[0];

If (Length(MyDevices) < 1) { Log("No matching device found."); }

// Look up business departments

DataTypes = {"Department"};
Filter = NULL;
MaxNum = 10000;

MyDepts = GetByLinks(DataTypes, Filter, MaxNum, MyDevices);

If (Length(MyDepts) < 1) { Log("No linked departments found."); }

// If department is mission-critical, update severity of alert

Count = Length(MyDepts);

While (Count > 0) {

    Index = Count - 1;
    MyDept = MyDepts[Index];

    If (MyDept.Name == "Data Center" || MyDept.Name == "Transaction Processing") {
        @Severity = 5;
    }

    Count = Count - 1;

}
```

Running the solution

The final step in this tutorial is to run the event enrichment solution.

Procedure

To start the solution, you simply start the OMNIbus event reader service.
The event reader then begins to monitor the ObjectServer and retrieves any new events that appear. When a new event appears, the event reader brings it back into Netcool/Impact, where it is processed by running the EnrichEvent policy.

Chapter 22. Configuring the Impact policy PasstoTBSM

In this scenario, you configure the Impact policy PasstoTBSM. You will create and configure an Impact policy, and create a TBSM service model to receive the data. You will create a custom portlet to view the data. When you have created the custom portlet, you will create a freeform page to display the data.

Expected Result

When you have completed this scenario, you will have a freeform custom page displaying data in TBSM, gathered from an Impact policy.

Overview

Use the `PasstoTBSM` function to send event information from Netcool/Impact to TBSM.

Netcool/Impact uses the function `PasstoTBSM` to send event information to TBSM. In an Impact policy, you can add the `PassToTBSM` function to the policy. When you activate the policy using a Netcool/Impact service, the event information is sent to TBSM.

In TBSM, you can manually configure an **Incoming status rule** to look for events coming from Netcool/Impact. The **Data Feed** list menu shows the Netcool/Impact service used to run the policy containing the `PasstoTBSM` function. To show the fields available for the selected Netcool/Impact service, you must manually customize the field names in Customize Fields window to match the fields in the policy.

You can also use the `PasstoTBSM` feature to transfer event information from a remote Netcool/Impact cluster to TBSM. To do this some additional configuration is required.

Configuration

You can use the `PassToTBSM` function on both local and remote installations. The syntax for `PassToTBSM` is the same as it is for a policy running on a TBSM server. For a remote installation the following tasks must be completed:

- The TBSM server must share a clustered name server with the remote Impact Server to view the Netcool/Impact services in the **Data Feed** list menu in the Edit Incoming status rule window.
- In TBSM an administration user configures `impact.sla.remoteimpactcluster=<cluster name of remote impact server>` in `etc/TBSM_sla.props` on the TBSM server.
- In Netcool/Impact, an administrator user exports the project, **For ImpactMigration** from TBSM and imports it into the remote version of Netcool/Impact. Netcool/Impact needs the **For ImpactMigration** project to access the TBSM data sources, and data types.

To call the `PassToTBSM` function from a remote Impact Server, the remote Netcool/Impactcluster needs the data type **ImpactEvents**. The **ImpactEvents** data type points to the **ImpactEvents** table in the DB2 database that TBSM uses. This

data type uses a data source called **TBSMDatabase**. The **TBSMDatabase** data source and the **ImpactEvents** data type belong to the project called **ForImpactMigration** in the TBSM server.

You can export this project from the TBSM server and import it into the remote Impact Server to provide the Impact Server with the data sources and data types required from TBSM.

Exporting and Importing the ForImpactMigration project

To call the `PassToTBSM` function from a remote Impact server, the remote Impact server needs to import the **ForImpactMigration** project and its contents from TBSM.

Before you begin

The **ForImpactMigration** project displays in the **Projects** list in the version of Impact that is contained within TBSM. The **ForImpactMigration** project has the data sources and data types necessary for remote Impact server to send events using `PassToTBSM`. To send events to TBSM from a remote Impact server, an administrator user needs to export the **ForImpactMigration** project from the TBSM server and import it into their Impact server.

About this task

Before you complete the export and import to the Impact server. Use the **Unlock all** button on the Global projects toolbar to unlock any locked items and check the `etc/<instance_name>_versioncontrol.locks` file for locked items before completing the export and import steps.

Procedure

1. In the TBSM server instance, run the nci export command.
   `<INSTALL_DIR>/tbsm/bin/nci_export TBSM --project ForMigration <exported dir>`
2. Copy the exported directory to the remote Impact server or to a location where the Impact server can access the directory.
3. In the Impact server instance, run the nci import command.
   `<INSTALL_DIR>/impact/bin/nci_import NCI <exported dir>` to import the **ForImpactMigration** into the remote Impact server.

Creating a policy

A policy example to use for `PassToTBSM` using the Web Services wizard to create the policy

About this task

The role of this policy is to monitor a web service that provides weather data on temperature and humidity about a particular city. In this example, create the policy using the Web service option in the policy Wizard.

Procedure

1. In the **Policies** tab, select the arrow next to the **New Policy** icon. Select **Use Wizard** > **Web Services** to open the Web Service Invoke-Introduction window.

2. In the Web Service Invoke-Introduction window, type in the policy name in the **Policy Name** field, for example `Weather` and click **Next** to continue.
3. In the Web Service Invoke-WSDL file and client stub window, in the **URL or Path to WSDL** field, enter the URL or a path for the target WSDL file. For example `http://wsf.cdyne.com/WeatherWS/Weather.asmx?wsdl`.

   In instances where the GUI server is installed separately from the Impact Server, the file path for the WSDL file refers to the Impact Server file system, not the GUI server file system. If you enter a URL for the WSDL file, that URL must be accessible to the Impact Server host and the GUI server host.
4. In the **Client Stub** area, select **Provide a package name for the new client stub**.
5. Enter a name for the package, for example **getWeatherInfoPkg**. Click **Next**.
6. In the Web Service Invoke-Web Service Name, Port and Method window, the general web service information is prepopulated for the following items; **Web Service** `Weather`, **Web Service Port Type** `WeatherSoap`, and **Web Service Method**. Select the option you want from the list, for example, `GetCityWeatherByZIP`. Click **Next**.
7. In the Web Service Invocation- Web Service Method parameters window, enter the parameters required by the target Web service method. For example, enter the name the zip code of the city you want to get weather information for. Click **Next**. When the wizard is complete it creates a policy which gets weather information from the selected web site for the specified city.
8. In the Web Service Invoke-Web Service EndPoint window, you can optionally edit the **URL or Path to WSDL** by selecting the edit check box. To enable web service security, select the **Enable web service security service** check box. Select one of the following authentication types:
   - **HTTP user name authentication**
   - **SOAP message user name authentication**

   Add the **User name** and **Password**. Click **Next.**
9. The Web Service Invoke-Summary and Finish window is displayed. It shows details relating to the policy. Click **Finish** to create the policy. When the wizard is completed, it generates the policy content. You can run the policy in the usual way and verify the results in the policy logger.
10. To extract the data from the policy and send it to TBSM. You must manually edit the policy and add the following lines to the policy. `PassToTBSM(ec);`

    Important: This policy uses a `NewEvent` object to pass the data. If you create an object to send the event data to PassToTBSM, use `NewEvent`, not `NewObject`. If your policy is driven by an event reader or listener, the `EventContainer` object can be sent directly into PassToTBSM. A `PolicyActivator` service does not pass any event object to its policy, so you must create a `NewEvent("EventSourceName")` including the name of the service which points to the event source from where events are read and sent. For example, `MyEvent = NewEvent("DefaultPolicyActivator");`

    An example of the web service policy generated by the web services wizard.

```
//This policy generated by Impact Wizard.

//This policy is based on wsdl file at
http://wsf.cdyne.com/WeatherWS/Weather.asmx?wsdl

log("Start policy 'getWeather'...");
//Specify package name as defined when compiling WSDL in Impact
WSSetDefaultPKGName('getWeatherInfoPkg');
```

```
//Specify parameters
GetCityWeatherByZIPDocument=WSNewObject
("com.cdyne.ws.weatherws.GetCityWeatherByZIPDocument");
_GetCityWeatherByZIP=WSNewSubObject
(GetCityWeatherByZIPDocument,"GetCityWeatherByZIP");

_ZIP = '27513';
_GetCityWeatherByZIP['ZIP'] = _ZIP;

WSParams = {GetCityWeatherByZIPDocument};

//Specify web service name, end point and method
WSService = 'Weather';
WSEndPoint = 'http://wsf.cdyne.com/WeatherWS/Weather.asmx';
WSMethod = 'getCityWeatherByZIP';

log("About to invoke Web Service call GetCityWeatherByZIP ......");

WSInvokeDLResult = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams);
log("Web Service call GetCityWeatherByZIP return result:
 " +WSInvokeDLResult);

//Added for PasstoTBSM

city = WSInvokeDLResult.GetCityWeatherByZIPResponse.
GetCityWeatherByZIPResult.City;
temperature=WSInvokeDLResult.GetCityWeatherByZIPResponse.
GetCityWeatherByZIPResult.Temperature;
humidity=WSInvokeDLResult.GetCityWeatherByZIPResponse.
GetCityWeatherByZIPResult.RelativeHumidity;

ec = NewEvent("WeatherActivator");
// Using a Policy Activator called WeatherActivator

ec.city=city;
ec.temperature=temperature;
ec.humidity=humidity;

log(" City : " + ec.city + "  Temp : " + ec.temperature + " Humid :
 " + ec.humidity);

PassToTBSM(ec);
```

Creating a policy activator service

Create the policy activator service to call the policy to get updates and to pass the updates to TBSM.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation** click **Services** to open the **Services** tab.
2. In the **Services** tab, click the **Create New Service** icon.
3. From the menu, select a template for the service that you want to create. In this instance, select **Policy Activator**.
4. Add the **Service Name** for example `WeatherActivator`, the **Activation Interval** in seconds, for example `300` and select the policy **Weather** you created earlier policy from the **Policy** list menu.
5. **Startup: Automatically when server starts** Select the checkbox to automatically start the service when the server starts. You can also start and stop the service from the GUI.
6. **Service log: Write to file** Select the checkbox to write log information to a file.

7. Click the **Save Service** icon.
8. Start the service.

Create a new template and rule to collect weather data

In this topic, you create a service structure to organize the weather data by city. You create a regional service template and an aggregation rule that depends on the City service template.

About this task

To create the service structure:

Procedure

1. From the **Service Navigation** pull-down menu, select **Templates**.
2. Click **Create New Template** button. The **Edit Template** tab opens in the Service Editor.
3. Enter `CityWeather` in the **Template Name** field.
4. Enter `Weather data by city` in the **Description** field.
5. In the **Rules** tab after Children, click **Create Incoming Status rule** button.
6. Select the **Based on a Numeric Value** radio button and click the **OK** button.
   The Edit Incoming status rule window opens.
7. Type `CityTemperature` in the **Rule Name** field.
8. Select **WeatherActivator** from the **Data Feed** drop-down list.
9. Click the **Customize Fields** button. The Customized Fields window opens.
10. Make sure the values for the fields based on the match table below:

Table 18. Default Field Names and Types

Field Name	Field Type
`EventClass`	**String**
`EventType`	**String**
`ResourceName`	**String**
`SecondaryResourceName`	**String**
`Summary`	**String**
`Value1`	**Float**
`Value2`	**Float**
`Value3`	**Float**

11. Add the following fields:

Table 19. New Custom Fields and Types

Field Name	Field Type
`city`	**String**
`temperature`	**Float**
`humidity`	**Float**

12. Click **OK**.
13. Enter the remaining values for the incoming status rule using the table below as your guide.

Table 20. Settings for CityTemperature rule

Entry fields	Value
Instance Name	`city`
Expression	`temperature <`
Select the **Status** checkbox.	
Marginal Bad	80 95
Select the **Store data for this rule for TWA** checkbox.	

14. Click **OK**. The **CityTemperature** rule is listed in the **Rules** tab.
15. You can repeat the same steps to create a **CityHumidity** incoming status rule to collect humidity data from **WeatherActivator** data feed. Select humidity as the output value and choose values between 0 and 100 for status thresholds.
16. To save the rule, click the **Save** button in the **Edit Template** tool bar.

Create the CityHumidity rule for the CityWeather template

In this topic you will create a rule to collect data for the template.

Procedure

1. From the Service Navigation pull-down menu, select **Templates**.
2. If it is not already open, click the **Edit Template 'CityWeather'** tab.
3. Click the **Incoming Status Rule** button to open the Edit Incoming Status Rule Type window.

Figure 5. Incoming status rule button

4. Select the **Based on a Good, Marginal, and Bad Threshold** radio button and click the **OK** button.

   The Create Incoming Status Rule window opens.
5. Type `CityHumidity` in the **Rule Name** field.
6. Select **weatherActivator** from the **Data Feed** drop-down list.
7. Click the **Customize Fields** button. The Customized Fields window opens.
8. Make sure the values for the fields based on the match table below:

Table 21. Default Field Names and Types

Field Name	Field Type
`EventClass`	**String**
`EventType`	**String**
`ResourceName`	**String**
`SecondaryResourceName`	**String**
`Summary`	**String**
`Value1`	**Float**
`Value2`	**Float**
`Value3`	**Float**

9. Add the following fields:

Table 22. New Custom Fields and Types

Field Name	Field Type
City	**String**
Temperature	**Float**
Humidity	**Float**

10. Click **OK**.
11. Enter the remaining values for the incoming status rule using the table below as your guide.

Table 23. Settings for CityTemperature rule

Entry fields	Value
Instance Name	City
Expression	Humidity <
Select the **Status** checkbox.	
Marginal	60
Bad	85
Select the **Store data for this rule for TWA** checkbox.	

12. Click the **OK** button.
13. Click the **Save** button in **Edit Template 'CityWeather'** tab.

    Note: The rule will not be saved to the TBSM database until you click the **Save** button.

    The CityHumidity rule displays in the **Rules** tab.

What to do next

Next: Create the service by hand.

In this topic, you create a service for city weather.

Create a city service

In this topic, you will create a service.

About this task

To create a service called Cary, complete the following steps:

Procedure

1. From the Service Navigation pull down menu, select **Services**.
2. Click the **Create New Service** button.

   The **Edit Service** tab opens in the Service Editor.
3. In the **Service Name** field type Cary.
4. In the **Templates** tab, click the template CityWeather in the **Available Templates** list and click the >> button.

The `CityWeather` template moves to the **Selected Templates** list.

5. Click the **Save** button in the **Edit Service** tab toolbar.
6. The Service Navigation portlet displays the new service in the **Services** tree.
7. In order to have a custom service tree with just the cities containing weather information, create another service (let's say, **Weather**) and make **Cary** a dependent of it.
8. Create a new tree template **CityWeather**, adding the **Temperature** and **Humidity** columns for the **CityWeather** template. Associate the new columns to `@CityTemperature` and `@CityHumidity`, respectively.

   For information on creating custom trees, see the *Service Configuration Guide* > *Custom service trees*.

Results

Next: Customize a Service Tree portlet

When you have created the service, you can customize a Service Tree portlet to only show the City weather information.

Customizing a Service Tree portlet

In this topic, you will be creating a customized Service Tree portlet.

Procedure

1. Click **Settings –> Portlets** in the navigation pane. A list of all navigation nodes in the console are displayed, grouped the same way as they are in the console navigation. The page includes all the portlets you can choose to customize.
2. Click **New**. The welcome page of the Create Widget wizard opens. Click **Next**. The next page is launched with the title Select a Base Widget.
3. Select the **Services** portlet. Click **Next**.
4. On the General page, enter `Weather by City` in the **Name** field.
5. Scroll through the thumbnail icon choices for the portlet, and choose according to the figure below.

6. Choose the Description Image for the new portlet as shown in the figure below:

7. Select **TBSM** and click the **Add >** button to add the new portlet to the TBSM catalog.
8. Click **Next**. The Security page is launched.
9. On the Security page, select **User** from the Selected Roles list.
10. Click **Add** to view a list of roles that can access this page.
11. Select these roles from the list of Available Roles:
    - **tbsmReadOnlyUser**

- **tbsmAdminUser**
-

12. Select **User** from the Selected Roles drop down list for user access levels. Click **Add**.
13. From the list of Available Roles, select **tbsmAdminUser**, select **Privileged User** from the Selected Roles drop down list for user access levels.
14. Click **Add**.
15. Click **Next** The Customize section opens.
16. On the General page, enter `Weather by City` for the portlet title.
17. Click **Next**. The Context page opens. Select **Weather** as starting instance.
18. Click **Next**. The View page opens.
19. In the Tree Template drop-down list, select **CityWeather**. Keep the defaults for the other fields.
20. Click **Next**. The Summary page displays.
21. Click **Finish**.
22. Verify in **Settings –> Portlets** that the new portlet is listed.

Results

Next: Adding a custom Services portlet to a freeform page

When you have customized a Service Tree portlet, you can add to a new page.

Adding a custom Services portlet to a freeform page

In this topic, you can add a custom Service Tree to a new freeform page.

Before you begin

To create a custom page, you need administrator privileges in TBSM.

About this task

To create a custom page, complete the following steps:

Procedure

1. Click **Settings –> Pages** in the navigation pane. A list of all navigation nodes in the console are displayed, grouped the same way as they are in the console navigation.
2. Click **New Page**. A new page is launched with the title Page Settings.
3. Enter `Weather Service` in the **Page name** field.
4. In the **Page location** field, click **Location** to browse for the location you want your page. `console/Availability/`. This value page specifies that the page will be listed under **Availability** in the console task list. Keep the defaults for the other fields.
5. In the **Page location** field, click **Location** to browse for where the new page will be listed in the console task list. Drag the new page into the **Availability** folder. This page is for read-only users who will not need to edit services. As a result, you add the page to the **Availability** group. The Location field is updated with `console/Availability/`. Keep the defaults for the other fields.
6. Click **OK**.

7. Select **Freeform** option under Page Layout.
8. Expand **Optional setting** to add roles access to this page.
9. Select **User** from the Selected Roles list.
10. Click **Add** to view a list of roles that can access this page.
11. Select these roles from the list of Available Roles:
    - **tbsmReadOnlyUser**
    - **tbsmAdminUser**
    -
12. Click **Add** .
13. For **tbsmReadOnlyUser**, select **User** from the Access Level drop-down list.
14. For **tbsmAdminUser**, select **Privileged User** from the Selected Roles list.
15. Click **OK**. The Portlet palette displays, which is used to select portlet content.
16. Select the **All** folder.
17. Use the arrows at the bottom of the Portlet palette to find and select the **Weather by City** portlet.
18. Drag the **City Weather Tree** portlet into the empty space below the Portlet palette. **--> Weather by City**
19. Drag a **Time Window Analyzer** and place it under the **Weather by City**.
20. In the **Time Window Analyzer**, click **Add Service**.
21. Search for `Cary`, and click on it. You can edit Shared Preferences to make **Cary** the default service for the Time Window Analyzer portlet.
22. Click **Edit Options > Skin** to customize the look of your portlet. The **Skin** option controls how the border of the portlet looks.
23. Click **Done**. The new page will open.

    Note: After you click **Done**, you will not be able to change or arrange your portlets.
24. Log out and log in as the `OSManager1` user to verify that the new user can see the page.

Chapter 23. Maintenance Window Management

Maintenance Window Management (MWM) is an add-on for managing Netcool OMNIbus maintenance windows.

MWM can be used with OMNIbus versions 7.1, 7.2, and 7.3. A maintenance time window is a prescheduled period of downtime for a particular asset. Faults and alarms, also known as events, are often generated by assets undergoing maintenance, but these events can be ignored by operations. MWM creates maintenance time windows and ties them to Netcool OMNIbus events that are based on OMNIbus fields values such as **Node** or **Location**.Netcool/Impact watches the Netcool OMNIbus event stream and puts these events into maintenance according to the maintenance time windows. The Netcool/Impact **MWMActivator** service located in the **System Configuration** > **Event Automation** > **Services** in the **MWM** project must be running to use this feature. For more information about maintenance windows, see "About MWM maintenance windows" on page 171.

Activating MWM in a Netcool/Impact cluster

Maintenance Window Management (MWM) interacts with Netcool OMNIbus using an Netcool/Impact policy activator service called MWMActivator. This service is turned off by default in Netcool/Impact.

About this task

Use the following steps to activate MWM in the Netcool/Impact cluster *NCICLUSTER*.

Procedure

1. Log on to Netcool/Impact.
2. Expand **System Configuration > Event Automation**. Click **Policies**.
3. From the **Cluster** list, select **NCICLUSTER**. From the **Project** list, select **MWM**.
4. In the **Policies** tab, select the **MWM_Properties** policy, right click, and select **Edit** or click the **Edit policy** icon to view the policy and make any required changes. For more information, see "Configure the MWM_Properties policy."
5. Click **Services**.
6. In the **Services** tab, select **MWMActivator**, right click, and select **Edit** or click the **Edit services** icon to open the **MWMActivator** service properties. Make any required changes. For information about these properties, see "Configuring MWMActivator service properties" on page 170.
7. To start the service, in the service status pane, select **MWMActivator** and either right click and select **Start** or click the **Start Service** arrow in the **Services** toolbar.

   When the service is running it puts OMNIbus events into maintenance based on schedules entered into the MWM GUI.

Configure the MWM_Properties policy

Configure the MWM_Properties policy for use with the MWM add-on.

The following configurable options are available in the Maintenance Window Management **MWM_Properties** policy.

- Maintenance window expiration
  - By default, MWM clears the "in maintenance" flag from corresponding OMNIbus events when a window expires. You can edit the policy so that MWM leaves those events flagged as "in maintenance" after the maintenance window expires.
- Flagging existing events when a maintenance window starts
  - By default, any matching events in OMNIbus are flagged, regardless of when they came into OMNIbus. You can modify the policy so that MWM flags only events arrive or deduplicate while the maintenance window is running.

You can change these options by editing the **MWM_Properties** policy in the **MWM** project.

1. Expand **System Configuration > Event Automation**, click **Services**.
2. In the **Projects** list, select **MWM**.
3. In the **Policies** tab, select **MWM_Properties**, right click and select **Edit** to open the policy. **MWM_Properties** is a small policy with a single function called `getProperties()`. Other MWM policies call this function to retrieve configuration information.
4. To change the MWM options, change the function and the given values to `TRUE` or `FALSE` if required.

See the following information in the policy for `clearFlag` options. `clearFlag = TRUE` is the default option.

```
Use clearFlag = TRUE if you want the maintenance flag
on events cleared when windows expire.
Use clearFlag = FALSE if you want Impact to leave the events tagged
as in maintenance after the window expires.
```

See the following information in the policy for `flagExistingEvents` options. `flagExistingEvents = TRUE` is the default option.

```
Use flagExistingEvents = TRUE if you want Impact to flag as "in maintenance"
events which last came in (based on LastOccurrence) before the time window started.
Use flagExistingEvents = FALSE if you want Impact to NOT flag events as
 "in maintenance" unless they come in during the maintenance window.

function getProperties(propsContext)
{
 propsContext = newobject();

//SET YOUR VALUES HERE//////////////////////
 clearFlag = TRUE;
 flagExistingEvents = TRUE;
//THANKS :)


 propsContext.clearFlag = clearFlag;
 propsContext.flagExistingEvents = flagExistingEvents;

}
```

Configuring MWMActivator service properties

Configure the **MWMActivator** service to check for OMNIbus events that require maintenance.

Procedure

1. Expand **System Configuration > Event Automation**, click **Services**.
2. In the **Services** tab, right click **MWMActivator** and select **Edit** or click the **Edit services** icon to open the properties for the **MWMActivator** service.
3. By default, the MWMActivator **Activation Interval** is set to 7 seconds. The **MWMActivator** service checks OMNIbus every seven seconds for events that require maintenance. Select the interval time you want to use. If possible use prime numbers.
4. Change the **Policy** value only if you have created your own policy to replace the **MWM_Properties** policy.
5. Select **Startup** to start the **MWMActivator** service when Netcool/Impact starts.
6. Select the **Service Log** to create a file of the service log.

Logging on to Maintenance Window Management

Use theTivoli Integrated Portal to access Maintenance Window Management (MWM).

Procedure

1. In the Tivoli Integrated Portal, expand **Troubleshooting and Support > Event Automation**.
2. Click **Maintenance Window Management** to open MWM. The main menu options are **Add One Time**, **Add Recurring**, and **View Windows**. There is also a **Time Zone** menu for setting your time zone. For more information about using these options, see "About MWM maintenance windows."

About MWM maintenance windows

Use the Maintenance Window Management (MWM) web interface to create maintenance time windows and associate them withNetcool OMNIbus events.

Netcool OMNIbus events are based on OMNIbus field values such as **Node** or **Location**. The Netcool OMNIbusevents are then put into maintenance according to these maintenance time windows. If events occur during a maintenance window, MWM flags them as being in maintenance by changing the value of the OMNIbus field, integer field, `SuppressEscl` to 6 in the alerts.status table.

A maintenance time window is prescheduled downtime for a particular asset. Faults and alarms (events) are often generated by assets that are undergoing maintenance, but these events can be ignored by operations. MWM tags OMNIbus events in maintenance so that operations know not to focus on them. You can use MWM to enter one time and recurring maintenance time windows.

- **One time windows** are maintenance time windows that run once and do not recur. **One Time Windows** can be used for emergency maintenance situations that fall outside regularly scheduled maintenance periods. You can use them all the time if you do not have a regular maintenance schedule.
- **Recurring time windows** are maintenance time windows that occur at regular intervals. MWM supports three types of recurring time windows:
  - **Recurring Day of Week**
  - **Recurring Date of Month**
  - **Every nth Weekday**

Maintenance time windows must be linked to OMNIbus events in order for MWM to mark events as being in maintenance. When you configure a time window, you

also define which events are to be associated with the time window. The MWM supports the use of **Node**, **AlertGroup**, **AlertKey**, and **Location** fields for linking events to time windows.

Creating a one time maintenance window

Create a one time maintenance time window for a particular asset.

Procedure

1. Click the **Add One Time** link to view the form to create a one time maintenance window.
2. Enter the appropriate values in the fields **Node**, **AlertGroup**, **AlertKey**, and **Location**.

   Select the **Equals** or **Like** options next to each field.
3. Click the calendar icon to select the**Start Time** and **End Time** for the maintenance time window.
4. Click **Add Window** to create the window.
5. Click **View Windows** to see the configured window.

Creating a recurring maintenance window

Create a recurring maintenance time window for a particular asset.

Procedure

1. Click the **Add Recurring** link to view the form for creating the different types of recurring time windows.
2. Enter the appropriate values in the fields **Node**, **AlertGroup**, **AlertKey**, and **Location**.

   Select the **Equals** or **Like** options next to each field.
3. Select the **Start Time** and **End Time** for the maintenance time window.
4. Select the type of recurring window and complete the details.
   - **Recurring Day of Week** These windows occur every week on the same day and at the same time of day. For example, you can set the window to every Saturday from 5 p.m. to 12 a.m. Or you can set the window for multiple days such as Saturday, Sunday, and Monday from 5 p.m. to 12 a.m.
   - **Recurring Day of Month** These windows occur every month on the same date at the same time of day. For example, you can set the window to every month on the 15th from 7 a.m. to 8 a.m. Or you can set the window for multiple months.
   - **Every nth Weekday** These windows occur every month on the same day of the week at the same time. For example, you can set the window to the first and third Saturday of the month from 5 p.m. to 12 a.m.
5. Click **Add Window** to create the window.
6. Click **View Windows** to verify that your time window has been added.

Viewing maintenance windows

Click the **View Windows** link to view a toolbar which contains links to the different types of windows. Your viewing options are:

- **One Time**
- **Day of Week**
- **Day of Month**
- **nth Weekday**
- **Active Windows**

If maintenance windows are defined in any of these window categories, click a link to view a list of defined maintenance windows.

The color of the status icon indicates whether the window is active (green), expired (red), or has not started yet (blue, future).

You can use the delete icon to delete a maintenance window.

Maintenance Window Management and other Netcool/Impact policies

Maintenance Window Management (MWM) runs independently from other Netcool/Impact policies or OMNIbus automations. Every seven seconds, MWM checks for open maintenance windows and marks the appropriate events as being in maintenance. Take this feature into consideration when you add your own policies and automations.

Known shortcomings

The MWM GUI interacts with the primary Impact server in the Impact cluster. Data is stored in the HSQL database of the primary Impact server. Data is not replicated between HSQL instances in the Impact cluster. If your primary Impact server changes, the MWM GUI and policies cannot interact with previously entered data.

If there are overlapping time windows, there is a chance that an event could be temporarily flagged as out of maintenance when the first window ends. If this situation occurs, the event is flagged as in maintenance the next time the **MWMActivator** Service runs. The `clearFlag` property comes to play here. If the `clearFlag = FALSE`, then the event is never marked as out of maintenance.

Maintenance Window Management does not work properly if the default cluster name, `NCICLUSTER`, is not used. When the MWM main page opens, you see the following message:

```
Could not retrieve a client for accessing the Impact server, under cluster:
clustername
```

For information about how to resolve this issue, see the Troubleshooting Guide.

Chapter 24. Event Isolation and Correlation

Event Isolation and Correlation is provided as an additional component of the Netcool/Impact product. Event Isolation and Correlation is developed using the operator view technology in Netcool/Impact. You can set up Event Isolation and Correlation to isolate an event that has caused a problem. You can also view the events dependent on the isolated event.

Overview

Netcool/Impact has a predefined project, **EventIsolationAndCorrelation** that contains predefined data sources, data types, policies, and operator views. When all the required databases and schemas are installed and configured you must set up the data sources. Then, you can create the event rules using the objectserver sql in the Event Isolation and Correlation configuration view from the Tivoli Integrated Portal. You can view the event analysis in the operator view, **EIC_Analyze**.

To set up and run the Event Isolation and Correlation feature the following steps need to be completed.

1. Install Netcool/Impact.
2. Install DB2 or use an existing DB2 installation.
3. Configure the DB2 database with the DB2 Schema in the Netcool/Impact launchpad.
4. Install Discovery Library toolkit from the Netcool/Impact launchpad.

   If you already have a Tivoli® Application Dependency Discovery Manager (TADDM) installation, configure the discovery library toolkit to consume the relationship data from TADDM. You can also consume the data is through the loading of Identity Markup Language (IDML) books. For additional information about the discovery library toolkit, see the *Tivoli Business Service Manager Administrator's Guide* and the *Tivoli Business Service Manager Customization Guide*. These guides are available in the Tivoli Business Service Manager 6.1.0.1 information center available from the following url, https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/Tivoli+Business+Service+Manager.
5. In the Tivoli Integrated Portal, configure the data sources and data types in the **EventIsolationAndCorrelation** project to use with the Impact Server.
6. Create the event rules in the UI to connect to the Impact Server.
7. Configure WebGUI to add a new launch point.

Detailed information about setting up and configuring Event Isolation and Correlation, is in the *Netcool/Impact Solutions Guide*.

Installing Netcool/Impact and the DB2 database

To run the Event Isolation and Correlation feature, install Netcool/Impact and the DB2 database and configure the DB2 Schema.

Procedure

1. Install Netcool/Impact. Refer to *Netcool/Impact Administration Guide, Chapter 2 Installation and migration*.
2. Install DB2. Netcool/Impact and Tivoli Business Service Manager support DB2 version 9.5 or higher. For information about installing and using DB2, see the information center listed here for the version you are using: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.common.doc/doc/t0021844.html.
   - In a z/Linux environment, you have to manually install the DB2 schema from the command line and not from the launchpad. Run the following command from the command line: `launchpad/zlinux/setup-dbconfig-zlinux.bin`.
3. Configure the DB2 database with the DB2 schema. A user who has permissions to run the DB2 command-line tools completes this step.
   - For Unix, use the user ID `db2inst1`.
   - For Windows, use the user ID `db2admin`.

   You can install the DB2 schema from the Netcool/Impact launchpad.

Installing the Discovery Library Toolkit

Install the discovery library toolkit, to import the discovered resources and relationships into the Services Component Registry database.

About this task

For information about the Services Component Registry see the *Services Component Registry API* information in the *Tivoli Business Service Manager Customization Guide* available from the following url, https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/Tivoli+Business+Service+Manager.

Use the discovery library toolkit to import data from Tivoli® Application Dependency Discovery Manager 7.1 or later to Tivoli Business Service Manager. The toolkit also provides the capability of reading discovery library books in environments that do not have a Tivoli Application Dependency Discovery Manager installation.

- If you are using Tivoli Business Service Manager and Netcool/Impact, use the information in *Installing the Discovery Library Toolkit* in the *Tivoli Business Service Manager Installation Guide* available in the Tivoli Business Service Manager 6.1.0.1 information center available from the following url, https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/Tivoli+Business+Service+Manager.
- For a Netcool/Impact implementation that does not use Tivoli Business Service Manager, the discovery library toolkit can be installed from the Netcool/Impact launchpad. For the Tivoli Business Service Manager related information, the data source must be configured to access the db2 database. This information is not required for an Netcool/Impact installation.

Procedure

1. From the Netcool/Impact launchpad, select **Install Discovery Library Toolkit**.
2. Unzip `DiscoveryLibraryToolkit.zip`, to a local directory where you are installing the database schema and or discovery library toolkit.
3. Navigate to the OS directory in which you are installing the discovery library toolkit.

4. Execute the **setup-dbconfig-<osname>.bin** file to install the database schema.
5. To install the discovery library toolkit, execute the **setup-dltoolkit-<osname>.bin**file. Where **osname** is either `Linux, Windows, Aix, or Solaris.`
6. During the installation of the discovery library toolkit, there are options to configure the Tivoli Business Service Manager data server. You can add any values you want. These values are not used in Netcool/Impact.

Event Isolation and Correlation policies

The **EventIsolationAndCorrelation** project has a list of predefined polices that are specific to Event Isolation and Correlation.

The following policies are in the **EventIsolationAndCorrelation** project and support the Event Isolation and Correlation feature and must not be modified:

- **EIC_IsolateAndCorrelate**
- **EIC_eventrule_config**
- **EIC_utils**
- **Opview_EIC_Analyze**
- **Opview_EIC_confSubmit**
- **Opview_EIC_configure**
- **Opview_EIC_requestHandler**

Event Isolation and Correlation operator views

The **EventIsolationAndCorrelation** project has a list of predefined operator views that are specific to Event Isolation and Correlation.

- **EIC_Analyze** shows the analysis of an event query.
- **EIC_confSubmit** supports the configuration of Event Isolation and Configuration.
- **EIC_configure** configures the event rules for Event Isolation and Configuration.
- **EIC_requestHandler** supports the configuration of Event Isolation and Configuration.

Configuring Event Isolation and Correlation data sources

All the Event Isolation and Correlation-related features are associated with the project, **EventIsolationAndCorrelation**. Configure the necessary data sources, data types, and data items for the event isolation and correlation.

Procedure

1. From the Tivoli Integrated Portal, click **System Configuration** > **Event Automation** > **Data Model**.
2. From the project list, select the project **EventIsolationAndCorrelation**. A list of data sources specific to the **EventIsolationAndCorrelation** feature display.
   - **EIC_alertsdb**
   - **SCR_DB**
   - **EventrulesDB**
3. For each data source, update the connection information, user ID, and password and save it.
4. Configure **EIC_alertsdb** to the object server where the events are to be correlated and isolated.

5. Configure **SCR_DB** to the Services Component Registry database.

   Note: When configuring the Services Component Registry (SCR) data sources, you must point the data sources to what is commonly called the SCR. The SCR is a schema within the TBSM database that is created when you run the DB2 schema configuration step. The schema is called **TBSMSCR**. The database has a default name of **TBSM**.
6. Configure **EventRulesDB** to the Services Component Registry database.

Configuring Event Isolation and Correlation data types

The **EventIsolationAndCorrelation** project has a list of predefined data types that are specific to Event Isolation and Correlation. Except for the data type **EIC_alertquery** which you must configure, the remaining data types are preconfigured and operate correctly once the parent data sources are configured.

About this task

The following list shows the Event Isolation and Correlation data sources and their data types:

- **EIC_alertsdb**
  - **EIC_alertquery**
- **SCR_DB**

  The following data types are used to retrieve relationship information from the Services Component Registry.
  - **bsmidenties**
  - **getDependents**
  - **getRscInfo**
- **EventRulesDB**

  The following data types used by the database contain the end user configuration for Event Isolation and Correlation.
  - **EVENTRULES**
  - **EIC_PARAMETERS**

Procedure

1. To configure the **EIC_alertquery** data type, right click on the data type and select **Edit**.
2. The **Data Type Name** and **Data Source Name** are prepopulated.
3. The **State** check box is automatically selected as **Enabled** to activate the data type so that it is available for use in policies.
4. **Base Table:** Specifies the underlying database and table where the data in the data type is stored.
5. Click **Refresh** to populate the table. The table columns are displayed as fields in a table. To make database access as efficient as possible, delete any fields that are not used in policies. For information about adding and removing fields from the data type see “SQL data type configuration window - Table Description tab” on page 30.
6. Click **Save** to implement the changes.

Creating, editing, and deleting event rules

How to create, edit, and delete an event rule for Event Isolation and Correlation.

Procedure

1. Select **System Configuration** > **Event Automation** > **Event Isolation and Correlation** to open the Event Isolation and Correlation page tab.
2. Click the **Create New Rule** icon to create an Event Rule. While creating this item the configure page has empty values for various properties.
3. Click the **Edit the Selected Rule** icon to edit the existing event rules.
4. Click the **Delete the Selected Rule** icon to delete an event rule from the system and the list.

Creating an event rule

Complete the following fields to create an event rule.

Procedure

1. **Event Rule Name:** Specify the event rule name. The event rule name must be unique across this system. When you select **Edit** or **New** if you specify an existing event rule name, the existing event rule is updated. When you edit an event rule and change the event rule name, a new event rule is created with the new name.
2. **Primary Event:** Enter the SQL to be executed against the objectserver configured in the data source **EIC_alerts db**. The primary event is the event selected for analysis.

   The primary event filter is used to identify if the event that was selected for analysis has a rule associated with it. The primary event filter is also is used to identify the object in the Services Component Registry database that has the event associated with it. The object may or may not have dependent entities. During analysis, the event isolation and correlation feature finds all the dependent entities and there associated events.

   For example, the primary event has 3 dependent or child entities and each of these entities has 3 events has associated with it. In total there are 9 dependent events. Any of these secondary events could be the cause of the primary event. This list of events is what is termed the list of secondary events. The secondary event filter is used to isolate one or more of these events to be the root cause of the issue.
3. **Test SQL:** Click **Test SQL** to test the SQL syntax specified in the primary event. Modify the query so that only one row is returned. If there are multiple rows, you can still configure the rule. However, during analysis only the first row from the query is used to do the analysis.
4. **Secondary Events:** The text area is for the SQL to identify the dependent events. When you specify the dependent events, you can specify variables or parameters which can be substituted from the primary event information. The variables are specified with the @ sign. For example, if the variable name is *dbname*, it must be specified as *@dbname@*. An example is `Identifier = 'BusSys Level 1.2.4.4' and Serial = @ser@`. The variables are replaced during the analysis step. The information is retrieved from the primary event based on the configuration in the parameters table and displays in the **Variables Assignment** section of the page.
5. **Extract parameters:** Click **Extract Parameters** to extract the variable name between **@** and populate the parameter table. Once the variable information is extracted into the table, you can edit each column.

a. Select the field against the regular expression you want to execute, and a substitution value is extracted.
b. Enter the regular expression in the regular expression column. The regular expression follows the IPL Syntax and is executed using the `RExtract` function.
c. When the regular expression is specified, click **Refresh** to validate the regular expression and check that the correct value is extracted. The table contains the parameters.

6. **Limit Analysis results to related configuration items in the Service Component Registry:** Select this check box if the analysis is to be limited to related configuration items only. If the check box is not selected, the dependent query will be returned.
7. **Primary Event is a root cause event:** Select this check box to identify whether the primary event is the cause event and rest of events, are symptom only events.
8. **Event Field:** Identifies the field in the event which contains the resource identifier in the Services Component Registry. Select the field from the drop-down menu that holds the resource identifier in the event.
9. **Time window in seconds to correlate events:** Add the time period the event is to analyze. The default value is 600 seconds. The events that occurred 600 seconds prior to the primary event are analyzed.
10. Click **Save Configuration** to add the configuration to the backend database.
11. Now the event rules are configured, the event is ready to be analyzed. You can view the event analysis in the in the **EIC_Analyze** page.

Configuring WebGUI to add a new launch point

Configure the WebGUI with a launch out context to launch the analysis page.

About this task

WebGUI can be configured to launch the analysis page. Refer to the procedure for launch out integration described in the following URL, http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?topic=/com.ibm.netcool_OMNIbus.doc_7.3.1/webtop/wip/task/web_con_integrating.html.

The URL you need for Event Isolation and Correlation is `<TIPHOSTNAME>:<TIPPORT>/opview/displays/NCICLUSTER-EIC_Analyze.html`. Pass the serial number of the selected row for the event.

Note: `NCICLUSTER` is the name of the cluster configured during the installation of Netcool/Impact. You must use the name of your cluster whatever it is, in the URL. For example, in Tivoli Business Service Manager the default cluster name is `TBSMCLUSTER`. To launch from Tivoli Business Service Manager, you would need to use the following html file, `TBSMCLUSTER-EIC_Analyze.html`.

Launching the Event Isolation and Correlation analysis page

How to launch the Event Isolation and Correlation analysis page.

About this task

There are two ways to launch the Event Isolation and Correlation analysis page.

- Manually by using the webpage and Event Serial number.
- Using the launch out functionality on Active Event List (AEL) or Lightweight Event List (LEL) from WebGUI in the Tivoli Enterprise Portal.

Procedure

Open a browser on Netcool/Impact. Use one of the following options:

- Point to `<TIPServer>:<TIPPort>/opview/displays/NCICLUSTER-EIC_Analyze.html?serialNum=<EventSerialNumber>`. Where `<TIPServer>` and `<TIPPort>`are the Netcool/Impact GUI Server and port and `EventSerialNumber` is the serial number of the event you want to analyze. To launch the analysis page outside of the AEL (Action Event List), you can add `serialNum=<Serial Number>` as the parameter.
- The Event Isolation and Correlation analysis page can be configured to launch from the Active Event List (AEL) or LEL (Lightweight Event List) within WebGUI. For more information see, "Configuring WebGUI to add a new launch point" on page 180. When you create the tool you have to specify only `<TIPSERVER>:port/opview/displays/NCICLSTER-EIC_Analyze.html.` You do not have to specify **`SerialNum`** as the parameter, the parameter is added by the AEL tool.

Viewing the Event Analysis

View the analysis of an Event query in the **EIC_Analyze** page.

About this task

The input for the **EIC_IsolateAndCorrelate** policy is the serial number of the event through the serialNum variable. The policy looks up the primary event to retrieve the resource identifier. The policy then looks up the dependent events based on the configuration. The dependent events are further filtered using the related resources, if the user has chosen to limit the analysis to the related resources. Once the serial number has been passed as the parameter in WebGUI, you can view the event from the AEL or LEL and launch the Analyze page.

Procedure

Select the event from the AEL or LEL and launch the Analyze page. The **EIC_Analyze** page contains three sections:

- **Primary Event Information:** shows the information on the selected event. This is the event on which the event isolation and correlation analysis takes place.
- **Correlated Events:** shows information about the dependent events identified by the tool. Dependant events are identified as the events that are associated with the dependant child resources of the device or object that is associated with the primary event. These events are displayed in the context of dependent resources that were identified from the Services Component Registry.
- **Event Rule process:** shows the rule which was identified and processed when this primary event was analyzed.

Appendix. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features you can use with *Netcool/Impact* when accessing it on the *IBM Personal Communications* terminal emulator:

- You can operate all features using the keyboard instead of the mouse.
- You can read text through interaction with assistive technology.
- You can use system settings for font, size, and color for all user interface controls.
- You can magnify what is displayed on your screen.

For more information about viewing PDFs from Adobe, go to the following web site: http://www.adobe.com/enterprise/accessibility/main.html

Glossary

This glossary includes terms and definitions for Netcool/Impact.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

assignment operator
An operator that sets or resets a value to a variable. See also operator.

B

Boolean operator
A built-in function that specifies a logical operation of AND, OR or NOT when sets of operations are evaluated. The Boolean operators are &&, || and !. See also operator.

C

command execution manager
The service that manages remote command execution through a function in the policies.

command line manager
The service that manages the command-line interface.

Common Object Request Broker Architecture (CORBA)
An architecture and a specification for distributed object-oriented computing that separates client and server programs with a formal interface definition.

comparison operator
A built-in function that is used to compare two values. The comparison operators are ==, !=, <, >, <= and >=. See also operator.

control structure
A statement block in the policy that is executed when the terms of the control condition are satisfied.

CORBA
See Common Object Request Broker Architecture.

D

database (DB)
A collection of interrelated or independent data items that are stored together to serve one or more applications. See also database server.

database event listener
A service that listens for incoming messages from an SQL database data source and then triggers policies based on the incoming message data.

database event reader
An event reader that monitors an SQL database event source for new and modified events and triggers policies based on the event information. See also event reader.

database server
A software program that uses a database manager to provide database services to other software programs or computers. See also database.

data item
A unit of information to be processed.

data model
An abstract representation of the business data and metadata used in an installation. A data model contains data sources, data types, links, and event sources.

data source
A repository of data to which a federated server can connect and then retrieve data by using wrappers. A data source can contain relational databases, XML files, Excel spreadsheets, table-structured files, or other objects. In a federated system, data sources seem to be a single collective database.

data source adapter (DSA)
A component that allows the application to access data stored in an external source.

data type
An element of a data model that represents a set of data stored in a data source, for example, a table or view in a relational database.

DB See database.

DSA See data source adapter.

dynamic link
An element of a data model that represents a dynamic relationship between data items in data types. See also link.

E

email reader
A service that polls a Post Office Protocol (POP) mail server at intervals for incoming email and then triggers policies based on the incoming email data.

email sender
A service that sends email through an Simple Mail Transfer Protocol (SMTP) mail server.

event An occurrence of significance to a task or system. Events can include completion or failure of an operation, a user action, or the change in state of a process.

event processor
The service responsible for managing events through event reader, event

listener and email reader services. The event processor manages the incoming event queue and is responsible for sending queued events to the policy engine for processing.

event reader
A service that monitors an event source for new, updated, and deleted events, and triggers policies based on the event data. See also database event reader, standard event reader.

event source
A data source that stores and manages events.

exception
A condition or event that cannot be handled by a normal process.

F

field A set of one or more adjacent characters comprising a unit of data in an event or data item.

filter A device or program that separates data, signals, or material in accordance with specified criteria. See also LDAP filter, SQL filter.

function
Any instruction or set of related instructions that performs a specific operation. See also user-defined function.

G

generic event listener
A service that listens to an external data source for incoming events and triggers policies based on the event data.

graphical user interface (GUI)
A computer interface that presents a visual metaphor of a real-world scene, often of a desktop, by combining high-resolution graphics, pointing devices, menu bars and other menus, overlapping windows, icons and the object-action relationship. See also graphical user interface server.

graphical user interface server (GUI server)
A component that serves the web-based graphical user interface to web browsers through HTTP. See also graphical user interface.

GUI See graphical user interface.

GUI server
See graphical user interface server.

H

hibernating policy activator
A service that is responsible for waking hibernating policies.

I

instant messaging reader
A service that listens to external instant messaging servers for messages and triggers policies based on the incoming message data.

instant messaging service
A service that sends instant messages to instant messaging clients through a Jabber server.

IPL See Netcool/Impact policy language.

J

Java Database Connectivity (JDBC)
An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call level interface for SQL-based and XQuery-based database access.

Java Message Service (JMS)
An application programming interface that provides Java language functions for handling messages.

JDBC See Java Database Connectivity.

JMS See Java Message Service.

JMS data source adapter (JMS DSA)
A data source adapter that sends and receives Java Message Service (JMS) messages.

JMS DSA
See JMS data source adapter.

K

key expression
An expression that specifies the value that one or more key fields in a data item must have in order to be retrieved in the IPL.

key field
A field that uniquely identifies a data item in a data type.

L

LDAP See Lightweight Directory Access Protocol.

LDAP data source adapter (LDAP DSA)
A data source adapter that reads directory data managed by an LDAP server. See also Lightweight Directory Access Protocol.

LDAP DSA
See LDAP data source adapter.

LDAP filter
An expression that is used to select data elements located at a point in an LDAP directory tree. See also filter.

Lightweight Directory Access Protocol (LDAP)
An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory. See also LDAP data source adapter.

link An element of a data model that defines a relationship between data types and data items. See also dynamic link, static link.

M

mathematic operator
A built-in function that performs a mathematic operation on two values. The mathematic operators are +, -, *, / and %. See also operator.

mediator DSA
A type of data source adaptor that allows data provided by third-party systems, devices, and applications to be accessed.

N

Netcool/Impact policy language (IPL)
A programming language used to write policies.

O

operator
A built-in function that assigns a value to a variable, performs an operation on a value, or specifies how two values are to be compared in a policy. See also assignment operator, Boolean operator, comparison operator, mathematic operator, string operator.

P

policy A set of rules and actions that are required to be performed when certain events or status conditions occur in an environment.

policy activator
A service that runs a specified policy at intervals that the user defines.

policy engine
A feature that automates the tasks that the user specifies in the policy scripting language.

policy logger
The service that writes messages to the policy log.

POP See Post Office Protocol.

Post Office Protocol (POP)
A protocol that is used for exchanging network mail and accessing mailboxes.

precision event listener
A service that listens to the application for incoming messages and triggers policies based on the message data.

S

security manager
A component that is responsible for authenticating user logins.

self-monitoring service
A service that monitors memory and other status conditions and reports them as events.

server A component that is responsible for maintaining the data model, managing services, and running policies.

service
A runnable sub-component that the user controls from within the graphical user interface (GUI).

Simple Mail Transfer Protocol (SMTP)
An Internet application protocol for transferring mail among users of the Internet.

Simple Network Management Protocol (SNMP)
A set of protocols for monitoring systems and devices in complex networks. Information about managed devices is defined and stored in a Management Information Base (MIB). See also SNMP data source adapter.

SMTP See Simple Mail Transfer Protocol.

SNMP
See Simple Network Management Protocol.

SNMP data source adapter (SNMP DSA)
A data source adapter that allows management information stored by SNMP agents to be set and retrieved. It also allows SNMP traps and notifications to be sent to SNMP managers. See also Simple Network Management Protocol.

SNMP DSA
See SNMP data source adapter.

socket DSA
A data source adaptor that allows information to be exchanged with external applications using a socket server as the brokering agent.

SQL database DSA
A data source adaptor that retrieves information from relational databases and other data sources that provide a public interface through Java Database Connectivity (JDBC). SQL database DSAs also add, modify and delete information stored in these data sources.

SQL filter
An expression that is used to select rows in a database table. The syntax for the filter is similar to the contents of an SQL WHERE clause. See also filter.

standard event reader
A service that monitors a database for new, updated, and deleted events and triggers policies based on the event data. See also event reader.

static link
An element of a data model that defines a static relationship between data items in internal data types. See also link.

string concatenation
In REXX, an operation that joins two characters or strings in the order specified, forming one string whose length is equal to the sum of the lengths of the two characters or strings.

string operator
A built-in function that performs an operation on two strings. See also operator.

U

user-defined function
A custom function that can be used to organize code in a policy. See also function.

V

variable
A representation of a changeable value.

W

web services DSA
A data source adapter that exchanges information with external applications that provide a web services application programming interface (API).

X

XML data source adapter
A data source adapter that reads XML data from strings and files, and reads XML data from web servers over HTTP.

Index

F

G

H

I

J

K

L

M

N

O

P

Q

S

T

U

V

W

X

Printed in USA

SC23-8834-04